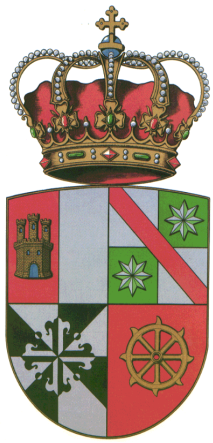
Universidad de Castilla-La Mancha

Departamento de Sistemas Informáticos



Desplegado de Programas Lógicos Difusos

TESIS DOCTORAL

Presentada por:

Jaime Penabad Vázquez

Dirigida por:

Ginés Moreno Valverde

Pascual Julián Iranzo

Desplegado de Programas Lógicos Difusos

Jaime Penabad Vázquez

Departamento de Sistemas Informáticos
Universidad de Castilla-La Mancha

Memoria presentada para optar al título de:

Doctor en Informática

Dirigida por:

Ginés Moreno Valverde
Pascual Julián Iranzo

Tribunal de lectura:

Presidenta:	**María Alpuente Frasnedo**	**U. P.**	**Valencia**
Vocales:	**Manuel Ojeda Aciego**	**U. M.**	**Málaga**
	Rafael Caballero Roldán	**U. C.**	**Madrid**
	Fernando Cuartero Gómez	**U. C. L. M.**	**Albacete**
Secretario :	**Guillermo Manjabacas Tendero**	**U. C. L. M.**	**Albacete**

Albacete, mayo de 2010.

Resumen

El objetivo general de esta tesis es la introducción de un conjunto de transformaciones, basadas en desplegado, para optimizar programas lógicos difusos. Introducimos por primera vez estas técnicas de transformación de programas, que por otra parte son clásicas en paradigmas declarativos precedentes como el lógico, funcional, lógico-funcional, etc., sobre distintos lenguajes de la programación lógica difusa (a los que también en ocasiones aportamos diferentes tipos de enriquecimientos), poniendo especial énfasis en el relativamente reciente marco lógico multi-adjunto. En este potente paradigma difuso se extiende la noción clásica de cláusula al etiquetar las reglas con pesos y permitir la presencia de distintos tipos de conectivas difusas en sus cuerpos, al tiempo que se dispone de una amplia gama de retículos para modelar la noción de grado de verdad en dominios más ricos que el intervalo $[0, 1]$.

Hemos estudiado las principales semánticas de la programación lógica multi-adjunta y aportado resultados relevantes sobre las relaciones que mantienen entre ellas. Además de formular mediante teoría de modelos la noción de modelo mínimo de Herbrand difuso en este contexto, también mejoramos su semántica procedural a dos niveles complementarios. Por un lado, demostramos sobre la fase operacional una extensión del resultado clásico de independencia de la regla de computación, lo que resulta crucial para demostrar las propiedades formales de nuestra noción de desplegado operacional. Por otra parte, diseñamos la fase interpretativa en términos de un sistema de transición de estados, lo que resulta preceptivo para formular el desplegado interpretativo en este marco y posteriormente abordar rigurosos análisis del coste computacional asociados a este tipo de computaciones y transformaciones.

En este mismo contexto, adaptamos conceptos clásicos de evaluación parcial con la intención inicial de disponer de técnicas de especialización para programas lógicos multi-adjuntos. Un paso más allá, y con una motivación completamente diferente y original, mostramos que estas mismas técnicas admiten una fácil reutilización para el

cálculo eficiente de reductantes (reglas especiales que son necesarias para garantizar la completitud -aproximada- en este contexto). Una particularidad de los lenguajes difusos en general, y del lógico multi-adjunto en especial, es que buena parte de las técnicas de manipulación de programas se prestan a su optimización mediante técnicas de umbralización: se trata de evitar computaciones irrelevantes en función de la gestión de los grados de verdad que llevan asociadas. En este sentido, hemos diseñado un algoritmo refinado y ágil para computar PE-reductantes muy eficientes mediante técnicas de evaluación parcial basadas en desplegado con un conjunto de umbrales dinámico.

Todos los desarrollos descritos en esta tesis vienen acompañados de demostraciones formales de sus propiedades fundamentales (incluyendo invariablemente las de corrección, completitud y eficiencia), al tiempo que se proporcionan detalles técnicos sobre el prototipo FLOPER que se está implementando en nuestro grupo y que pretende servir de plataforma sobre la que implantar todos los avances relatados en la misma.

Agradecimientos

Dedico esta tesis a mi familia y, muy especialmente, a mis padres cuya entrega incondicional ha permitido que mis expectativas de optar a una vida mejor hayan resultado más amplias que las suyas. Y también a todos los amigos que he ido encontrando a lo largo de los años: su aliento y sus pautas hacen más fáciles las tareas profesionales y más amable la vida diaria.

Deseo agradecer sinceramente la colaboración de todas las personas que han hecho posible este proyecto. Es mi intención recordar especialmente:

- A Ginés Moreno, por la ayuda constante sin la que no hubiera prosperado esta tesis ni hubiera tenido la oportunidad de incorporarme a un área de investigación en la que he podido reorientar y mejorar mi formación, a la vez que contactar con personas de gran nivel científico y categoría humana. Su apoyo en estos años ha sido absoluto y su aportación en esta tesis decisiva, ofreciendo siempre ideas fecundas y procurando la finura en el estilo y la precisión en los detalles. Sólo el placer de haber trabajado con él me hubiera gratificado suficientemente por su inquietud profesional, dedicación y generosidad.

- A Pascual Julián que, pese a no mediar con él relación personal anterior, ha tenido total disponibilidad desde el principio para sugerir ideas, revisar cálculos, contrastar resultados, en definitiva para orientarme y ayudarme en lo que he necesitado. Su experiencia y amplia cultura en este campo me han aportado mucho, al igual que la exigencia que siempre procura en la formulación de conceptos y la presentación de trabajos. La confianza que me ha entregado, el rigor científico y la seriedad profesional son para mi ejemplo a seguir.

- A ambos, Ginés y Pascual, debo agradecer la ayuda inestimable en esta tesis. Las prolongadas sesiones de trabajo con ellos han hecho viable el proyecto y, so-

bre todo, me han abierto muchas oportunidades de ampliar mis conocimientos en un ambiente de cordialidad y de amistad muy gratos para mi.

- A los compañeros del Departamento de Matemáticas de la Universidad de Castilla-La Mancha, con quienes comparto docencia, que siempre me han entregado su colaboración y su apoyo en momentos de dificultades personales o profesionales. Con ellos resulta muy agradable el trabajo diario, por la ayuda, la complicidad y la amistad que siempre me ofrecen.

- A los organismos e instituciones que han colaborado en el desarrollo de los trabajos de esta tesis: la Universidad de Castilla-La Mancha, donde llevo a cabo mi labor docente e investigadora, el Ministerio de Educación y Ciencia, y el Ministerio de Ciencia e Innovación, que han financiado en gran medida las publicaciones de la misma.

- A los revisores, por sus orientaciones y correcciones, que han permitido depurar y mejorar la memoria inicial.

La Matemática es el único lenguaje realmente universal.
Carl Sagan.

Índice general

Capítulo 1

Introducción

En esta tesis nos proponemos, como objetivo principal, la introducción de un conjunto de transformaciones, basadas en desplegado, para optimizar programas lógicos difusos. Estas técnicas de transformación de programas, que son clásicas en la programación declarativa (lógica, funcional y lógico funcional), se introducen aquí por primera vez sobre distintos lenguajes de la programación lógica difusa, y se tratan con especial énfasis en el marco multi-adjunto.

Después de valorar distintos lenguajes de programación lógica difusa, optamos por escoger el lenguaje multi-adjunto como marco sobre el que abordar las transformaciones debido a su gran potencia y expresividad, que resulta de permitir distintas implicaciones en las reglas y conectivas muy generales en los cuerpos de las mismas. Como se documenta en las Secciones 1.5 y 2.5, este lenguaje mejora considerablemente otras propuestas de lenguajes difusos, ubicándose en un contexto muy general en el que muchos otros programas lógicos difusos resultan instancias de un programa multi-adjunto o pueden ser reescritos como un programa de esta clase.

Por tal motivo, la programación lógica multi-adjunta resulta muy atractiva para estudiar reglas de transformación de programas. En efecto, una vez caracterizadas en este ámbito, pueden ser adaptadas a los distintos lenguajes que cubre este estilo de programación difusa, hecho que, por otra parte, garantiza la relevancia y difusión de los resultados.

Además, dicho lenguaje es apropiado para formular el desplegado difuso debido, principalmente, a que disponemos de una sintaxis sencilla y una semántica operacional apropiada (formalizada como un sistema de transición de estados, que permite

la definición de reglas de desplegado y de técnicas de evaluación parcial) para dar cuenta de los grados de verdad. Es, por tanto, el marco (más general) que escogemos para definir la transformación de desplegado y demostrar sus propiedades de corrección y eficiencia; y para adaptar, asimismo, la evaluación parcial en contexto difuso y su aplicación posterior al cálculo de reductantes.

Previamente, realizamos dos extensiones del lenguaje f-Prolog debido a Vojtáš y Paulík [1996] a fin de hacer posible, de una parte, una interpretación más flexible de las conectivas y, de otra, permitir la codificación de los programas transformados, todo ello de forma que se conservan las semánticas operacional y declarativa del lenguaje original. Sobre el lenguaje que recoge las dos extensiones, que llamamos lef-Prolog, definimos también la regla de desplegado, y sus propiedades de corrección y eficiencia.

Hemos estudiado las semánticas de la programación lógica multi-adjunta, así como las relaciones entre ellas. Mejoramos la semántica procedural, añadiendo a la fase operacional (debida a los autores del lenguaje), el diseño de la fase interpretativa en términos de un sistema de transición de estados, lo que nos resulta preceptivo para formular el desplegado interpretativo en este marco y, además, abordar un análisis del coste computacional de los programas multi-adjuntos. Para la fase operacional, demostramos la independencia de la regla de computación, extendiendo el resultado conocido de la programación lógica.

Definimos, por primera vez, el concepto de *modelo mínimo de Herbrand difuso* adaptando, para este tipo de progamas, los conceptos clásicos de la programación lógica pura y expresamos a partir del mismo la semántica declarativa de los programas multi-adjuntos.

Probamos la equivalencia de esta semántica con la de punto fijo obtenida por Medina *et al.* [2004] y consideramos asimismo las relaciones de ambas con el conjunto de respuestas computadas difusas (semántica procedural).

Por otra parte, mostramos cómo los métodos tradicionales que estiman el número de pasos de computación de una derivación produce resultados no deseados cuando estimamos el esfuerzo computacional de la fase interpretativa de los programas multi-adjuntos. Para salvar esta dificultad definimos una medida de coste computacional adecuada para este tipo de programas que, además, usamos en el cálculo de *reductantes* para justificar que nuestro 1-*reductante* tiene la misma eficiencia que el primitivo. Estas medidas de coste podrán servir en el futuro para estimar la eficiencia de las técnicas de plegado/desplegado.

Posteriormente, definimos una regla de desplegado para programas lef-Prolog (y para otras variantes de este lenguaje) y para programas multi-adjuntos, y estudiamos sus propiedades de corrección y eficiencia. Para este lenguaje, estudiamos dos reglas de desplegado (que llamamos operacional e interpretativo) y demostramos los mejores resultados de corrección y eficiencia.

Además, adaptamos al marco multi-adjunto los conceptos clásicos de evaluación parcial, con la intención de disponer de técnicas de especialización de programas multi-adjuntos y proponermos asimismo el cálculo eficiente de reductantes (necesarios para garantizar la completitud de estos programas). En efecto, basándonos en las técnicas clásicas de evaluación parcial para programas declarativos y de nuestra experiencia en adaptar la transformación de desplegado en la programación lógica borrosa adaptamos el concepto de EP al marco de la programación lógica multi-adjunta para lograr una versión optimizada del programa original que se ejecuta más eficientemente.

Los *reductantes* son una herramienta introducida por Kifer y Subrahmanian [1992] para demostrar propiedades de corrección y completitud de los programas lógicos generalizados con anotaciones, y adaptados recientemente por Medina *et al.* [2001c] al marco multi-adjunto ya que un programa lógico multi-adjunto, interpretado en un retículo completo, precisa contener estas reglas para que esté garantizada la completitud (aproximada).

Por nuestra parte, introducimos un nuevo concepto de reductante con pocas diferencias sintácticas con el de Medina *et al.* [2004], pero que puede definirse y extenderse usando, de una forma muy novedosa, conceptos de evaluación parcial que facilitarán su cálculo. Surge así el concepto de *PE-reductante* que generaliza y mejora el reductante de Medina *et al.* [2004] y para los que, además, pueden usarse técnicas de evaluación parcial con umbralización, que permiten simplificar su expresión y reducir las secuencias de derivación en las que intervienen. Finalmente, diseñamos un algoritmo eficiente para computar *PE-reductantes* mediante técnicas de evaluación parcial basadas en desplegado con un conjunto de umbrales dinámico.

1.1. Contribuciones

Las principales contribuciones de esta tesis son las siguientes.

Lenguajes

Entre los distintos lenguajes de programación lógica difusa el f-Prolog descrito en [Vojtáš y Paulík, 1996], es ya una referencia clásica y lo escogemos para adaptar el concepto de desplegado de programas lógicos difusos [Julián *et al.*, 2004b,c]. Mejoramos este lenguaje f-Prolog con marcas etiquetadas, obteniendo lf-Prolog, extensión que resulta imprescindible para poder codificar los programas desplegados.

Además, extendemos los dos lenguajes mencionados a fin de permitir una interpretación flexible de las conectivas, de modo que se admitan diferentes lógicas en un mismo programa y se mejore la potencia expresiva. Se obtienen así los lenguajes resultantes ef-Prolog y lef-Prolog (que conservan la misma semántica operacional y declarativa que los primitivos) que tratamos en [Julián *et al.*, 2004a, 2005c].

Además, adaptamos el concepto clásico de *regla de computación* para estos lenguajes difusos, y probamos la independencia de la regla de computación para programas y objetivos lf-Prolog [Julián *et al.*, 2004b,c], y para los correspondientes del lenguaje lef-Prolog [Julián *et al.*, 2004a, 2005c].

En cuanto al lenguaje multi-adjunto, implementamos, en el grupo DEC-TAU, un prototipo de intérprete/compilador para traducir programas lógicos multi-adjuntos a código Prolog, que aspira a ser una plataforma útil para optimizar programas difusos. Esta herramienta permite ya en la actualidad compilar, ejecutar y manipular programas lógicos difusos, según hemos diseñado en [Julián *et al.*, 2006c], y también generar y visualizar árboles de desplegado.

Presentamos estas contribuciones en [Julián *et al.*, 2004b], [Julián *et al.*, 2004c], [Julián *et al.*, 2004a], [Julián *et al.*, 2005c] y [Julián *et al.*, 2006c].

Semánticas

Definimos, por primera vez en la literatura, una semántica declarativa para la programación lógica multi-adjunta en términos del *modelo mínimo de Herbrand difuso*, adaptando a nuestro marco (tal como figura en [Julián *et al.*, 2009]) la construcción clásica de modelo mínimo de Herbrand de la programación lógica pura, que ha sido aceptada tradicionalmente como la semántica declarativa de programas lógicos. Este modelo mínimo permite también caracterizar las respuestas correctas.

Asimismo, proponiéndonos una revisión de las semánticas conocidas del lenguaje multi-adjunto, hemos estudiado todas las relaciones entre nuestra noción de modelo mínimo difuso y las ya existentes semántica procedural y de punto fijo, demostrando

en particular la equivalencia entre la semántica declarativa por modelo mínimo y la de punto fijo debida a Medina *et al.* [2004]. Incluimos una demostración original de la corrección de la semántica procedural de la programación multi-adjunta. Aportamos también ejemplos reveladores en los que nuestra semántica declarativa sigue teniendo sentido mientras las anteriormente mencionadas no están definidas.

Además, clarificamos la semántica procedural del lenguaje, mejorando la fase interpretativa, tal como se recoge en [Julián *et al.*, 2006c], que hemos diseñado como un sistema de transición de estados, hecho preceptivo para la formalización del desplegado interpretativo en este marco. Esta reformulación de la fase interpretativa, permite además el análisis del coste computacional de estos programas multi-adjuntos que referimos a continuación.

Demostramos, tal como se justifica en [Julián *et al.*, 2005a], la independencia de la regla de computación, adaptando a nuestro contexto el resultado conocido de la programación lógica pura.

Por otro lado, definimos una medida de coste computacional para los programas lógicos multi-adjuntos, que estima convenientemente el coste de la fase interpretativa de estos programas, después de observar que el cómputo de los pasos de derivación es una medida ingenua que no resulta apropiada en este ámbito.

Hemos definido, en [Julián *et al.*, 2007c], [Julián *et al.*, 2007a], una medida de coste computacional para dichos programas, que contabiliza el número de conectivas y operadores primitivos que aparecen dentro de la definición de los agregadores que son evaluados en cada paso (interpretativo) de una derivación dada. Posteriormente, aplicamos este criterio de coste para contrastar (véanse de nuevo [Julián *et al.*, 2007c], [Julián *et al.*, 2007a]) la eficiencia de dos nociones semánticamente equivalentes de *reductantes* (la original, introducida por Medina *et al.* [2004] y nuestra versión refinada de 1-*reductante*).

Presentamos la mayoría de estas contribuciones en [Julián *et al.*, 2005a], [Julián *et al.*, 2006c], [Julián *et al.*, 2007a], [Julián *et al.*, 2007c] y [Julián *et al.*, 2009].

Transformaciones

Adaptamos al contexto difuso (para varios lenguajes) la noción clásica de desplegado, manteniendo el comportamiento de esta transformación en entornos no difusos. Definimos, en primer lugar, el desplegado difuso para el lenguaje f-Prolog de Vojtáš y Paulík [1996] (véase [Julián *et al.*, 2004b,c]) y para el lenguaje ef-Prolog.

Introducimos, además, (véase [Julián *et al.*, 2004a, 2005c]) las reglas de reem-

plazamiento de t-norma para el lenguaje lef-Prolog, que acelera el cálculo de los grados de verdad, y es también un precedente de la de desplegado interpretativo que referimos a continuación.

Con especial atención abordamos la transformación de desplegado en el marco de la programación lógica multi-adjunta, contemplando dos tipos de desplegado muy relacionados: el operacional (primeramente introducido en [Julián *et al.*, 2005a]) y el interpretativo (introducido por vez primera en [Julián *et al.*, 2006c]).

Este último enriquece significativamente las reglas de reemplazamiento de t-norma referidas, complementa el desplegado operacional (exhibiendo propiedades de corrección fuerte análogas a las de éste) y mejora la evaluación de los grados de verdad en la fase interpretativa, tal como justificamos en [Julián *et al.*, 2006c].

Finalmente, enfatizamos el papel central de la transformación de desplegado (en el marco multi-adjunto) tanto para producir evaluadores parciales y mejorar el cálculo de reductantes como para obtener semánticas por desplegado para este lenguaje.

En [Julián *et al.*, 2004b], [Julián *et al.*, 2004c], [Julián *et al.*, 2005a], [Julián *et al.*, 2004a], [Julián *et al.*, 2005c] y [Julián *et al.*, 2006c] se encuentran estas aportaciones.

Propiedades de corrección

Para las transformaciones de desplegado difuso consideradas en el apartado anterior sobre distintos dialectos de Prolog difuso y el lenguaje multi-adjunto, obtenemos los mejores resultados de corrección y eficiencia que cabe esperar. En efecto, demostramos que los programas residuales obtienen las mismas respuestas computadas difusas que el original y, además, que se obtiene mejora en eficiencia cuando ejecutamos los programas transformados.

Para el lenguaje lef-Prolog probamos la corrección total fuerte (véase [Julián *et al.*, 2004a], [Julián *et al.*, 2004b] y [Julián *et al.*, 2005c]), y para el lenguaje multi-adjunto, demostramos las propiedades de corrección del desplegado operacional, del desplegado interpretativo y, finalmente, los resultados de corrección del sistema de transformación determinado por la combinación de ambos tipos de desplegado. Estas propiedades del desplegado operacional se recogen en [Julián *et al.*, 2005a], y las correspondientes al desplegado interpretativo y al sistema de transformación en [Julián *et al.*, 2005b] y en [Julián *et al.*, 2006c].

En cuanto a la eficiencia, vemos que las secuencias de transformación pueden ser dirigidas de manera arbitraria, dado que cualquier paso de computación ejecu-

tado con alguna de las transformaciones consideradas produce una ganancia en los programas residuales.

Observamos que los resultados correspondientes al desplegado de programas lef-Prolog no son corolarios inmediatos de los correspondientes del lenguaje multi-adjunto porque estos lenguajes tienen su propia semántica operacional, aparte de otros rasgos distintivos a nivel sintáctico y procedural.

Estos resultados pueden suponer el punto de partida para la optimización de programas lógicos difusos y constituyen un referente en la construcción de un sistema global de plegado/desplegado (incluyendo más reglas de transformación y estrategias) para la optimización de los programas referidos.

Presentamos estas contribuciones en [Julián *et al.*, 2004a], [Julián *et al.*, 2004b], [Julián *et al.*, 2005c], [Julián *et al.*, 2005a] y [Julián *et al.*, 2006c].

Evaluación parcial y reductantes

Construimos, por primera vez (en [Julián *et al.*, 2009]), un marco de evaluación parcial para programas multi-adjuntos que permite especializar programas de manera semejante a cómo se ha procedido en otros contextos declarativos (apoyándonos en las técnicas de deducción parcial de programas lógicos, pero usando ahora la regla de desplegado difuso para esta clase de programas) para obtener un programa más eficiente.

Introducimos (como puede verse en [Julián *et al.*, 2006b], [Julián *et al.*, 2009]) una reformulación del concepto de *reductante* que aporta las mismas respuestas computadas difusas que el primitivo, tiene la misma eficiencia y es también semánticamente equivalente (véase [Julián *et al.*, 2007c] y [Julián *et al.*, 2009]). Este nuevo reductante puede generalizarse y posee la ventaja de que puede obtenerse mediante técnicas de evaluación parcial (*PE-reductante*), lo que representa una aplicación singular de éstas. Dichas técnicas permiten, además, obtener reductantes generales más fáciles de construir (admiten una expresión más simple si usamos técnicas de EP con umbralización). En efecto, proponemos la obtención del *PE-reductante* a partir de un *árbol de desplegado* umbralizado para reducir su expresión y beneficiarnos de otras ventajas cuando damos pasos de computación con estas reglas (véanse [Julián *et al.*, 2006a, 2007b]).

Relacionando, de este modo, el concepto de EP de un átomo en un programa lógico multi-adjunto con la construcción de un *reductante* para dicho átomo, logramos diseñar un cálculo eficiente de reductantes, rebajando el impacto obtenido por

la incorporación de reductantes a un programa lógico multi-adjunto mediante un cálculo previo en el que éstos son parcialmente evaluados antes de añadirlos al programa final: dado que la fase de EP produce un conjunto refinado de reductantes, el esfuerzo computacional realizado (sólo una vez) en tiempo de generación es evitado (muchas veces) en tiempo de ejecución.

Diseñamos un algoritmo eficiente para el cálculo de *PE-reductantes*, que reduce significativamente el tamaño de los árboles de desplegado a partir de los que obtenemos dichos reductantes usando técnicas de evaluación parcial con umbralización.

En [Julián *et al.*, 2006b], [Julián *et al.*, 2006a], [Julián *et al.*, 2007b], [Julián *et al.*, 2007c] y [Julián *et al.*, 2009], se encuentran estas aportaciones.

1.2. Organización de la memoria

Esta memoria ha sido estructurada en nueve capítulos. En el Capítulo 1 presentamos un resumen de las nociones básicas (de lógica difusa y programación lógica) necesarias para abordar los conceptos estudiados en la tesis, así como una panorámica general de la programación lógica difusa. En el Capítulo 2 se describen los lenguajes lógicos difusos que hemos considerado apropiados para formular la transformación de desplegado. Se dedica especial atención al lenguaje multi-adjunto, el marco más general sobre el que trabajamos aquí y en capítulos posteriores. En el Capítulo 3 se abordan las semánticas del lenguaje multi-adjunto aportando, por nuestra parte, una semántica declarativa en términos del *modelo mínimo de Herbrand difuso* y estudiando las relaciones con el resto de semánticas. En el Capítulo 4 hemos formulado la transformación de desplegado para distintos lenguajes difusos, con especial detalle para el lenguaje lef-Prolog y el lenguaje multi-adjunto. En el Capítulo 5 se recogen los resultados de corrección, completitud y eficiencia que hemos obtenido para las transformaciones de desplegado estudiadas en el capítulo anterior. En el Capítulo 6 estudiamos, para programas multi-adjuntos, el concepto de *reductante* introduciendo una reformulación equivalente del mismo (que llamamos 1-*reductante*). Este nuevo *reductante* lo extendemos posteriormente al concepto general de *PE-reductante* en el Capítulo 7, mediante nociones clásicas de evaluación parcial, que también hemos adaptado previamente para este tipo de programas, al objeto de optimizar su cálculo. Al final de todos los capítulos (salvo del primero, naturalmente) se incluyen las conclusiones parciales. Finalmente, en el Capítulo 8 se analizan con detalle todas las conclusiones y en el Capítulo 9 se enuncian las líneas de trabajo futuro.

1.3. Lógica difusa

Las nociones básicas de la lógica difusa han sido formuladas por Zadeh [1965b,a], Goguen [1969] y Pavelka [1979] con la intención de incorporar a la lógica formal los predicados de carácter vago del lenguaje ordinario, que han permitido iniciar la construcción del razonamiento aproximado.

Para Zadeh [1996], creador de esta disciplina, el término lógica difusa o lógica borrosa, tiene dos significados diferentes. En el sentido más estricto, la lógica difusa constituye un sistema lógico que se ocupa de la formalización de modos de razonamiento aproximados. En este aspecto, resulta una extensión de los sistemas lógicos polivalentes y comparte la agenda de la lógica simbólica en el sentido en que busca establecer la corrección y completitud de los sistemas que estudia [Hajek, 2006], aunque sus objetivos son distintos. [Hájek, 1998; Novak *et al.*, 1999; Pavelka, 1979] son las referencias fundamentales en esta primera orientación. En el sentido más amplio, en el que mayoritariamente se concibe, la lógica borrosa coexiste con la teoría de conjuntos borrosos, que es una teoría de clases con fronteras no nítidas.

La lógica clásica, la teoría de conjuntos clásica y la probabilidad no resultan apropiadas para tratar la vaguedad, la imprecisión, la incertidumbre, la falta de especificidad, la inconsistencia y la complejidad del mundo real. Según Trillas *et al.* [1995], desde la lógica clásica sólo se han ofrecido soluciones insuficientes: o se ha pretendido ganar espacios de precisión frente a lo impreciso (Frege o Russel) o se ha querido aislar lo impreciso para evitarlo cuidadosamente (Platón, Hume o ciertas estrategias contemporáneas). Esta limitación de las herramientas tradicionales motiva el nacimiento de los conjuntos difusos y, paralelamente, de la lógica difusa.

En el marco discreto (bivalente) de las matemáticas clásicas no se tolera la vaguedad y la verdad parcial. En este ámbito, todo concepto debe admitir una definición precisa, que divida a los objetos del universo considerado en dos subconjuntos: el conjunto de los objetos que lo satisfacen, y el conjunto de los objetos que no lo satisfacen, sin admitir que se presenten casos dudosos.

En claro contraste con este mundo idealizado, la percepción de la realidad está invadida por conceptos sin fronteras nítidas [Terricabras y Trillas, 1989], como por ejemplo, alto, grande, muchos, la mayoría, lentamente, joven, saludable, relevante, mucho mayor que, amable, etc. Una hipótesis esencial en la lógica difusa es que tales conceptos determinan conjuntos difusos, es decir, clases de objetos en los que la transición de la pertenencia a la no pertenencia es gradual y no abrupta.

Además, el esfuerzo inicial por adquirir conocimiento del mundo real está de-

jando paso al esfuerzo por conocer aspectos del propio conocimiento. Hoy en día, no preocupa tanto la mera adquisición de conocimiento como la delimitación de su alcance y validez. Es preciso asignar un grado de certeza al conocimiento, saber en qué medida conocemos algo. La incertidumbre está asociada, de forma inseparable, con la información. Aunque existen diferentes formas de incertidumbre, cabe destacar la que se produce como consecuencia de la imprecisión y subjetividad propias de la actividad humana [Zadeh, 2008]. En muchas ocasiones, es conveniente sacrificar parte de la información precisa disponible por otra más vaga pero más potente, a fin de tratar de forma eficiente la complejidad del mundo real. Muchos de los conceptos manejados habitualmente tienen naturaleza vaga, es decir, no están bien delimitados, y no por ello carecen de significación. La lógica borrosa y la teoría de conjuntos borrosos ofrecen un método natural para representar la vaguedad y la imprecisión.

En [Zadeh, 1965b], L. A. Zadeh introduce una teoría sobre nuevos objetos, los conjuntos difusos, que son conjuntos de frontera no precisa y cuya función de pertenencia indica un grado.

Uno de los objetivos de esta lógica es proporcionar las bases del razonamiento aproximado que utiliza hipótesis vagas como herramienta para formular el conocimiento. Aunque de naturaleza distinta a las lógicas clásicas, en opinión de Trillas *et al.* [1995], esta lógica con infinitos grados de verdad puede verse como extensión de la lógica bivalente, de la trivalente definida por Lukasiewicz en 1922 y, en general, de la multivalorada [Ackerman, 1967; Chang, 1958].

Puede plantearse su objetivo como el intento de construir un modelo de razonamiento que perfile el aspecto cualitativo o aproximado, y tiene gran capacidad para tratar problemas muy complejos o mal definidos.

Su base natural la constituyen, como decíamos, los conjuntos difusos que son el instrumento matemático con el que se formalizan los predicados difusos y la posibilidad de realizar con ellos un cálculo lógico necesario para realizar inferencias.

La aparición de esta lógica inicia una de las teorías recientes más sugestivas y útiles para modelar un abundante número de problemas, para los que el marco de la lógica clásica y de la probabilidad resultaba insuficiente o poco apropiado. Para esta modelización, la lógica difusa ofrece desde símbolos y operadores que recogen el concepto de vaguedad hasta reglas de inferencia que conservan o acotan los grados de verdad asignados a las premisas.

1.3.1. Conjuntos difusos

La noción de conjunto difuso es introducida por Zadeh [1965b] y sobre ella puede formalizarse el concepto de interpretación difusa, las operaciones lógicas, los modificadores lingüísticos, etc. [Pedrycz y Gomide, 1998]. Mientras que para conjuntos ordinarios como

$$A = \{x \in \mathbb{Z} : x \text{ es primo}\},\ A = \{x \in \mathbb{N} : x \text{ es par}\},\ A = \{x : x \text{ es mortal}\}$$

la relación de pertenencia tiene un carácter discreto, es decir, un elemento (del correspondiente universo) pertenece o no pertenece al conjunto:

$$\forall x \in \mathcal{U}, \quad x \in A \ \vee \ x \notin A; \quad A \subset \mathcal{U},$$

en los conjuntos borrosos la pertenencia tiene un grado; es el caso de conjuntos del tipo:

$$A = \{x \in \mathbb{Z} : x \text{ es grande}\}$$

$$A = \{x : x \text{ es día caluroso}\}$$

$$A = \{x : x \text{ es país desarrollado}\}$$

para los que la naturaleza de la propiedad (predicado) que los caracteriza ya no es precisa, como en los conjuntos ordinarios, sino vaga, difusa. Los elementos del universo dejan de satisfacer/no satisfacer el predicado asociado, sino que lo satisfacen en un determinado grado. Para formalizar estos conjuntos es esencial dar sentido a la nueva pertenencia, lo que se hace del siguiente modo.

Un conjunto difuso (o borroso) A, en un universo $\mathcal{U}$, se expresa como:

$$A = \{x | \mu_A(x) : \mu_A(x) \neq 0, x \in \mathcal{U}\},$$

donde la aplicación

$$\mu_A : \mathcal{U} \to [0, 1]$$

es la función grado de pertenencia.

Es decir, un conjunto difuso está determinado por la función μ_A. Para cada $x \in \mathcal{U}$, $\mu_A(x) \in [0, 1]$ es un número real que mide la compatibilidad de x con el conjunto A, con la característica (predicado) que define A.

Si observamos que la pertenencia ordinaria está determinada por la función característica

$$\chi_A : \mathcal{U} \to \{0, 1\}, \quad \chi_A(x) = \begin{cases} 1, & \text{si } x \in A \\ 0, & \text{si } x \notin A \end{cases}$$

o sea,

$$x \in A \Leftrightarrow \chi_A(x) = 1, \quad x \notin A \Leftrightarrow \chi_A(x) = 0, \quad \forall x \in \mathcal{U}$$

la pertenencia borrosa, graduada por μ_A, es una generalización de la clásica ya que χ_A es una función μ_A particular.

De esta observación, sencilla pero relevante, se deriva el hecho de que todo conjunto ordinario es un conjunto difuso; es decir, la noción de conjunto difuso extiende a la de conjunto clásico. En particular, el universo $\mathcal{U}$ (que tomamos como ordinario en la definición de A) y el conjunto vacío $\varnothing$ son borrosos. En efecto,

$$\mu_\varnothing = \chi_\varnothing \text{ tal que } \mu_\varnothing(x) = \chi_\varnothing(x) = 0, \ \forall x \in \mathcal{U}$$

$$\mu_\mathcal{U} = \chi_\mathcal{U} \text{ tal que } \mu_\mathcal{U}(x) = \chi_\mathcal{U}(x) = 1, \ \forall x \in \mathcal{U}$$

Caracterizado el conjunto difuso por su función grado de pertenencia, μ_A, todas sus propiedades se remiten a ésta, por lo que el contenido, el complementario, las operaciones, etc., se expresan a partir de las correspondientes funciones grado de pertenencia.

La definición esencial (al menos desde un punto de vista semántico) de la lógica difusa es –naturalmente– la de interpretación, en la que se asocia a cada fórmula (atómica) un número real que se toma habitualmente[1] en el intervalo $[0, 1]$. En efecto, se asigna (un grado de) verdad a una proposición difusa mediante el concepto de conjunto difuso del siguiente modo:

Dado un predicado $A(x)$ en un universo $\mathcal{U}$ y un elemento $x_0 \in \mathcal{U}$, la fórmula $A(x_0)$ se interpreta como verdadera con grado de verdad $\mu_A(x_0)$. En tal caso escribimos:

$$\mathcal{I}(A(x_0)) = \mu_A(x_0)$$

Por tanto, venimos a decir que la proposición $A(x_0)$ se satisface con grado $\mu_A(x_0)$, el grado de pertenencia de x_0 al conjunto difuso A. Y este conjunto es, necesariamente,

$$A = \{x \in \mathcal{U} : A(x)\},$$

es decir, el conjunto cuyo predicado asociado es $A(x)$.

Igualmente, podríamos asumir que la definición anterior constituye la formalización (interpretación) del predicado $A(x)$ mediante el conjunto difuso A. En efecto, es lícito plantearla en los siguientes términos:

[1] De manera más general puede tomarse en un cierto conjunto ordenado [Zadeh, 2008], como consideraremos posteriormente en este mismo capítulo.

x_0 satisface el predicado $A(x)$ con grado $\mu_A(x_0)$, esto es, el grado de pertenencia de x_0 al conjunto difuso $A = \{x \in \mathcal{U} : A(x)\}$.

Debemos advertir que, posteriormente, a través de los modificadores ligüísticos, será posible contemplar las proposiciones difusas como falsas, muy verdaderas, muy falsas, algo verdaderas, algo falsas, etc. De este modo, podremos incorporar también el carácter difuso al concepto de interpretación: más concretamente, podrá asociarse una nueva interpretación a cada modificador de predicado (no, muy, algo, aproximadamente, etc.) que se formalice. Este hecho aportará otro rasgo diferenciador de la lógica difusa.

Finalmente, debemos distinguir la vaguedad de un enunciado (en cuanto a lo que enuncia), de la certidumbre (en cuanto a su cumplimiento). Es decir, ésta no es una lógica posibilística que considera proposiciones no difusas sobre las que cabe alguna incertidumbre no aleatoria.

1.3.2. t-normas, t-conormas y agregadores

A grandes rasgos, la sintaxis de la lógica difusa no presenta muchas novedades. Una vez que se ha interpretado una expresión elemental, fórmulas al efecto otorgan verdad a las expresiones compuestas [Lee, 1972]. Así, por ejemplo, la conjunción se define habitualmente por la expresión

$$\mathcal{I}(A(x_0) \wedge B(x_0)) = min\{\mathcal{I}(A(x_0)), \mathcal{I}(B(y_0))\},$$

donde $A(x), B(y)$ son predicados cualesquiera en universos $\mathcal{U}, \mathcal{V}$, respectivamente, y $x_0 \in \mathcal{U}, y_0 \in \mathcal{V}$.

Si tomamos predicados $A(x), B(x)$ sobre el mismo universo $\mathcal{U}$ y definen, respectivamente, los conjuntos borrosos $A, B \subset \mathcal{U}$, se tiene además

$$\mathcal{I}(A(x_0) \wedge B(x_0)) = \mu_{A \cap B}(x_0),$$

donde $\mu_{A \cap B}(x_0)$ es el grado de pertenencia de x_0 al conjunto intersección $A \cap B$. Es decir, se permite definir la conjunción lógica borrosa a partir de la correspondiente intersección conjuntista.

Pero, de manera más general, la (función de verdad de la) conjunción difusa se puede definir también por todo un conjunto de funciones: las normas triangulares, introducidas por Schweizer y Sklar [1983] para modelar las distancias en espacios métricos probabilísticos (definidos por K. Menger en 1942) y los semigrupos de funciones de distribución.

La definición de estas funciones, en el intervalo $[0,1]$, es la que sigue.

Definición 1.3.1 ([Nguyen y Walker, 2006]). *Una operación* $T : [0,1] \times [0,1] \longrightarrow [0,1]$ *es una norma triangular o t-norma si, y sólo si, verifica*

i) *es conmutativa, es decir,* $T(x,y) = T(y,x), \forall x, y \in [0,1]$.

ii) *es asociativa, es decir,* $T(x, T(y,z)) = T(T(x,y), z), \forall x, y, z \in [0,1]$.

iii) $T(x,1) = x, \forall x \in [0,1]$.

iv) *es monótona en cada componente, es decir*[2]*, si* $x_1 \leq x_2$*, entonces* $T(x_1, y) \leq T(x_2, y), \forall x_1, x_2, y \in [0,1]$.

De manera análoga, la disyunción se caracteriza habitualmente por la expresión

$$\mathcal{I}(A(x_0) \vee B(x_0)) = max\{\mathcal{I}(A(x_0)), \mathcal{I}(B(y_0))\},$$

y si consideramos predicados $A(x), B(x)$ sobre el mismo universo $\mathcal{U}$ definiendo, respectivamente, los conjuntos borrosos $A, B \subset \mathcal{U}$, se tiene además

$$I(A(x_0) \vee B(x_0)) = \mu_{A \cup B}(x_0),$$

donde $\mu_{A \cup B}(x_0)$ es el grado de pertenencia de x_0 al conjunto unión $A \cup B$. En consecuencia, esta operación lógica aparece asociada a la unión conjuntista. Más exactamente, se puede formalizar la disyunción borrosa (de proposiciones y también de predicados), mediante la unión de conjuntos borrosos.

Además, como para la conjunción, la (función de verdad de la) disyunción difusa se puede definir también por todo un conjunto de funciones: las t-conormas, caracterizadas del siguiente modo en el intervalo cerrado $[0,1]$.

Definición 1.3.2 ([Nguyen y Walker, 2006]). *Una operación* $S : [0,1] \times [0,1] \longrightarrow [0,1]$ *es una co-norma triangular o t-conorma si, y sólo si, verifica*

i) *es conmutativa, es decir,* $S(x,y) = S(y,x), \forall x, y \in [0,1]$.

ii) *es asociativa, es decir,* $S(x, S(y,z)) = S(S(x,y), z), \forall x, y, z \in [0,1]$.

iii) $S(x,0) = x, \forall x \in [0,1]$.

[2] De la caracterización dada (sólo para la primera componente) se sigue también la monotonía en la segunda componente usando las condiciones *i)* y *iv)*.

iv) es monótona en cada componente, es decir[3]*, si* $x_1 \leq x_2$*, entonces* $S(x_1, y) \leq S(x_2, y), \forall x_1, x_2, y \in [0,1]$.

Si T es una t-norma en $[0,1]$, entonces la igualdad $S(x,y) = 1 - T(1-x, 1-y)$ define una t-conorma y se dice que S deriva de T. Más generalmente, dada una t-norma T y una negación fuerte[4] N, la función $S_N : [0,1] \times [0,1] \longrightarrow [0,1]$, definida como $S_N(x,y) = N(T(N(x), N(y)))$, es una t-conorma a la que denominaremos t-conorma N-dual de T.

Haciendo uso de propiedades elementales de la negación se tiene que $T(x,y) = N(S_N(N(x), N(y)))$, es decir T es la t-norma N-dual de S_N.

Dada una t-conorma S y una negación fuerte N, la función $T_N : [0,1] \times [0,1] \longrightarrow [0,1]$, definida como $T_N(x,y) = N(S(N(x), N(y)))$, es una t-norma a la que denominaremos t-norma N-dual de S.

De nuevo, por ser N una negación, $S(x,y) = N(T_N(N(x), N(y)))$, es decir, es a su vez la t-norma N-dual de T_N.

En conclusión, diremos que T y S son N-duales si $\forall x, y \in [0,1]$ se cumple:

$T(x,y) = N(S(N(x), N(y)))$ $\qquad$ $S(x,y) = N(T(N(x), N(y)))$

En particular, tomando la negación usual $N(x) = 1-x$, T y S son duales si $\forall x \in [0,1]$ se cumple:

$$T(x,y) = 1 - S(1-x, 1-y) \qquad S(x,y) = 1 - T(1-x, 1-y)$$

Los pares de t-normas y t-conormas (duales) básicos son los siguientes (véase [Carlsson *et al.*, 1997]):

- De Zadeh (o del Mínimo/Máximo) definidos por

$$T(x,y) = min\{x,y\} \qquad S(x,y) = max\{x,y\}$$

- De Łukasiewicz definidos por

$$T(x,y) = max\{x + y - 1, 0\} \qquad S(x,y) = min\{x + y, 1\}$$

[3] La monotonía en la segunda componente resulta también de *i)* y *iv)*.

[4] Una negación fuerte en $[0,1]$ es una función $N : [0,1] \longrightarrow [0,1]$ continua, estrictamente decreciente y tal que $N(0) = 1, N(N(x)) = x$.

- Producto definidos por

$$T(x,y) = xy \qquad S(x,y) = x + y - xy$$

- Débil/Fuerte definidos por

$$T(x,y) = \begin{cases} min\{x,y\}, & \text{si } max\{x,y\} = 1 \\ 0, & \text{en otro caso} \end{cases} \qquad S(x,y) = x + y - xy$$

- De Hamacher definidos, para cada $\gamma \geq 0$ por

$$T_\gamma(x,y) = \frac{xy}{\gamma + (1-\gamma)(x + y - xy)} \qquad S(x,y) = \frac{x + y - (2-\gamma)xy}{1 - (1-\gamma)xy}$$

- De Yager definidos, para cada $p > 0$, por

$$T_p(x,y) = 1 - min\{1, \sqrt[p]{(1-x)^p + (1-y)^p}\} \quad S_p(x,y) = min\{1, \sqrt[p]{x^p + y^p}\}$$

Es bien conocido el uso de t-normas y t-conormas para la generación de nuevas conectivas [Mizumoto, 1989a,b; Turksen, 1992; Fodor y Calvo, 1998; Durante *et al.*, 2007; Klement *et al.*, 2004]. t-normas y t-conormas son casos particulares de los operadores de agregación[5] (estudiados estos últimos por Dubois y Prade [1984, 1985, 1986], Fodor y Yager [1994]; Fodor y Roubens [1992] y Yager [1993a,b, 1994b,a]) y, además, ciertas combinaciones de éstos originan nuevos operadores de agregación [Calvo *et al.*, 1999, 2002].

Una posibilidad de combinar agregadores (véase [Ling, 1965; Mizumoto, 1989b; Turksen, 1992; Moser *et al.*, 1999; Jenei y Montagna, 2003; Jenei, 2004, 2006]) es la que ofrecen las combinaciones convexas de una t-norma T y una t-conorma S, es decir tomar el agregador $@(x,y) = \alpha T(x,y) + (1-\alpha)S(x,y)$, que preservan la simetría y la idempotencia.

Encontramos agregadores en el desarrollo de múltiples sistemas inteligentes. En efecto, se usan los operadores de agregación en las redes neuronales, en los controladores difusos, en los sistemas expertos y, muy especialmente, en la teoría de la

[5] Véase la Definición 1.3.3 para una caracterización de éstos.

decisión. Mediante los operadores de agregación se gestiona la agregación de información de manera eficiente y flexible [Herrera *et al.*, 1996], aspecto éste que se ha convertido en tarea principal de los problemas de decisión multicriterio, en los que es necesario procesar mucha información de modo que su cantidad y precisión es muy variada.

La definición más general de operador de agregación, en el intervalo $[0,1]$, es la considerada en [Kolesárová y Komorníková, 1999], que damos en los siguientes términos.

Definición 1.3.3. *Un operador de agregación* $@ : [0,1]^n \longrightarrow [0,1]$ *que satisface:*

i) $@(0,\ldots,0) = 0, @(1,\ldots,1) = 1$ *(condiciones de frontera)*

ii) $\forall (x_1,\ldots,x_n),(y_1,\ldots,y_n) \in [0,1]^n$,
$(x_1,\ldots,x_n) \leq (y_1,\ldots,y_n)$[6] $\Longrightarrow @(x_1,\ldots,x_n) \leq @(y_1,\ldots,y_n)$ *(monotonía)*

A las condiciones de la definición anterior se añaden, en ocasiones, otras como la continuidad, simetría e idempotencia. En particular, $@$ es simétrico si, y sólo si, para toda permutación σ de $\{1,\ldots,n\}$ y toda n-upla $(x_1,\ldots,x_n) \in [0,1]^n$ se cumple $@(x_1,\ldots,x_n) = @(x_{\sigma(1)},\ldots,x_{\sigma(n)})$; además, $@$ es idempotente (es decir, $@(x,\ldots,x) = x$) si, y sólo si, para toda n-upla $(x_1,\ldots,x_n) \in [0,1]^n$ se cumple $min\{x_1,\ldots,x_n\} \leq @(x_1,\ldots,x_n) \leq max\{x_1,\ldots,x_n\}$.

Ejemplos bien conocidos de operadores de agregación son las t-normas y t-conormas (ya considerados anteriormente), las medias con peso cuasi-lineales [Aczél, 1948; Yager, 1994b] (si, además, son simétricos dan las medias cuasi-aritméticas, como la media aritmética, la media geométrica, le media armónica y la media cuadrática), los operadores OWA [Yager, 1988] (la media aritmética es también un caso particular de estos operadores), las funciones de agregación extendidas [Mayor y Calvo, 1997], los γ-operadores de Zimmermann y Zysno [1980], entre otros.

1.3.3. Implicaciones difusas

La implicación difusa constituye la expresión compuesta más interesante de la lógica difusa (como de la lógica clásica). Su función de verdad admite diversas formulaciones, no equivalentes; obtendremos así distintas implicaciones difusas, que no siempre extienden la implicación ordinaria [Trillas *et al.*, 2000], [Carlsson y Fullér, 1995].

[6] donde $(x_1,\ldots,x_n) \leq (y_1,\ldots,y_n)$ si, y sólo si, $x_i \leq y_i, i = 1,\ldots,n$.

La manera más frecuente de interpretar la implicación difusa

$$A(x_0) \Rightarrow B(y_0)$$

es la que da la fórmula

$$\mathcal{I}(A(x_0) \Rightarrow B(y_0)) = max\{min\{\mathcal{I}(A(x_0)), \mathcal{I}(B(y_0))\}, 1 - \mathcal{I}(A(x_0))\},$$

donde $A(x), B(y)$ son predicados arbritrarios en los universos $\mathcal{U}, \mathcal{V}$ respectivamente y $x_0 \in \mathcal{U}, y_0 \in \mathcal{V}$. Si los predicados $A(x), B(y)$ definen los conjuntos borrosos $A \subset \mathcal{U}, B \subset \mathcal{V}$, se tiene además

$$\mathcal{I}(A(x_0) \Rightarrow B(y_0)) = max\{min\{\mu_A(x_0), \mu_B(y_0)\}, 1 - \mu_A(x_0)\}; \quad x_0 \in \mathcal{U}, y_0 \in \mathcal{V}$$

Con esta función de verdad, debida a Zadeh, se comprueba fácilmente que la implicación difusa generaliza la implicación clásica.

Otros ejemplos destacados son los debidos a Mamdani y Larsen que dan, respectivamente, la siguiente interpretación de implicación difusa[7]

$$\mathcal{I}(A(x_0) \Rightarrow B(y_0)) = min\{\mathcal{I}(A(x_0)), \mathcal{I}(B(y_0))\}$$

$$\mathcal{I}(A(x_0) \Rightarrow B(y_0)) = \mathcal{I}(A(x_0)) \cdot \mathcal{I}(B(y_0))$$

Pero, en lo que sigue nos planteamos presentar la implicación borrosa de modo más general. Empecemos considerando que, dada un álgebra de Boole $(A, \wedge, \vee, ', 0, I)$, una operación $\rightarrow$: $A \times A \longrightarrow A$ es una implicación si para cada $x, y \in A$, se tiene $x \wedge (x \rightarrow y) \leq y$. Es bien conocido que $p \rightarrow q = (p \wedge q')'$, $p \rightarrow q = (p \wedge q) \vee (p' \wedge q) \vee (p \wedge q')$ son implicaciones y por las propiedades del álgebra de Boole se tiene $(p \wedge q) \vee (p' \wedge q) \vee (p \wedge q') = (p' \wedge q') \vee q = p' \vee (p \wedge q) = p' \vee q$, y también $(p \wedge q')' = p' \vee q$.

Si, en el contexto difuso, nos planteamos elegir una t-norma T en lugar de la conjunción $\wedge$, una t-conorma S en lugar de la disyunción $\vee$ y una negación fuerte N en lugar de la negación $'$, obtenemos los siguientes modelos de implicación difusa [Trillas *et al.*, 2000]

$$\begin{aligned} J_1(x,y) &= N(T(x, N(y))) \\ J_2(x,y) &= S(N(x), y) \\ J_3(x,y) &= S(N(x), T(x,y)) \\ J_4(x,y) &= S(T(N(x), N(y)), y) \end{aligned}$$

[7] Ambas implicaciones lógicas son muy usadas en el control difuso.

Ocupándonos de su caracterización, una implicación difusa en el intervalo $[0,1]$, se define por la siguiente función de verdad[8] [Trillas *et al.*, 2000; Trillas y Valverde, 1985].

Definición 1.3.4. *Una función implicación J es una aplicación $J : [0,1]^n \longrightarrow [0,1]$ que satisface:*

(1) *Si $x_1 \leq x_2$, entonces $J(x_1, y) \geq J(x_2, y)$*

(2) *Si $y_1 \leq y_2$, entonces $J(x, y_1) \leq J(x, y_2)$*

(3) $J(0, y) = 1$

(4) $J(1, y) = y$

(5) $J(y, J(x, z)) = J(x, J(y, z)), \forall x, y, z \in [0,1]$

Además, es frecuente que se exija alguna de las siguientes condiciones

(6) $J(x, 0) = N(x)$

(7) J es continua

(8) $J(x, y) = J(N(y), N(x))$, para alguna negación fuerte N

(9) $J(x, x) = 1$

(10) $x \leq y$ si, y sólo si, $J(x, y) = 1$

Existen tres clases muy relevantes de (funciones de verdad de) implicaciones [Carlsson *et al.*, 1997; Alsina *et al.*, 1995]:

- S-implicaciones definidas por

$$x \longrightarrow y = S(N(x), y)$$

  donde S es una t-conorma y N es una negación en [0,1]. Estas implicaciones están sugeridas por la equivalencia, en la lógica binaria, de las fórmulas $p \rightarrow q$ y $p' \vee q$. Ejemplos de S-implicaciones son las debidas a

[8] Para mayor comodidad, omitimos en lo que sigue toda mención a las expresiones lógicas que intervienen en la hipótesis y en la tesis de la implicación, para referirnos exclusivamente a la función de verdad de la conectiva implicación.

- Łukasiewicz definida por $x \longrightarrow y = min\{1 - x + y, 1\}$
- Kleene-Dienes, definida por $x \longrightarrow y = max\{1 - x, y\}$

- R-implicaciones definidas por residuación de una t-norma continua T del modo

$$x \longrightarrow y = sup\{z \in [0,1] : T(x,z) \leq y\}$$

Estas implicaciones surgen de la lógica intuicionista, entre ellas tenemos las debidas a

- Gödel, definida por $x \longrightarrow y = \begin{cases} 1, & \text{si } x \leq y \\ y, & \text{si } x > y \end{cases}$
- Łukasiewicz, definida por $x \longrightarrow y = min\{1 - x + y, 1\}$

Son admitidas como implicaciones las

- Implicaciones t-norma, definidas a partir de una t-norma T del modo

$$x \longrightarrow y = T(x,y)$$

de las que son ejemplos la implicación de Mamdani, muy usada en la teoría de control (difuso), y la que resulta de la t-norma del producto.

- QM-implicaciones [Trillas *et al.*, 2000; Ying, 2002], caracterizadas por una función de verdad $Q : [0,1]^2 \longrightarrow [0,1]$ dada por $Q(x,y) = S(N(x), T(x,y))$, a partir de una t-norma T, una t-conorma S y una negación fuerte N.

  Ejemplos de QM-operadores (en rigor, así debiéramos denominarlos por no determinar, en general, una implicación) son los dados por

  - $Q_1(x,y) = S(1-x, T(x,y)) = max\{1-x, y\}$, donde T es la t-norma de Łukasiewicz y S su t-conorma dual.
  - $Q_2(x,y) = S(1-x, T(x,y)) = 1 - x + x^2 y$, donde T es la t-norma Producto y S su t-conorma dual.
  - $Q_3(x,y) = \begin{cases} 1, & \text{si } y = 1 \\ y, & \text{si } x = 1 \\ 1-x, & \text{en otro caso} \end{cases}$

Nótese que en lógica clásica la S-implicación $p' \vee q$ y la QM-implicación $p' \vee (p \wedge q)$ son equivalentes y definen la implicación lógica ordinaria, pero son diferentes como operadores difusos.

Para la inferencia difusa (y la implicación borrosa) es esencial la propiedad del modus ponens difuso o generalizado, cuya primera propuesta ha sido formulada por L. A. Zadeh y transmite la verdad aproximada de las premisas a la conclusión usando una composición de relaciones borrosas. Tiene el siguiente esquema general:

$$\left.\begin{array}{c} \text{Si } A(x) \text{ entonces } B(y) \\ \text{y} \\ A'(x) \end{array}\right\} \text{ entonces } B'(y)$$

donde $A(x)$, $A'(x)$ son predicados difusos arbitrarios (en un universo arbitrario $\mathcal{U}$) lo mismo que $B(y)$, $B'(y)$ (en un universo $\mathcal{V}$). Tales predicados están asociados a los correspondientes conjuntos borrosos A, A', B, B'.

El grado de verdad que se toma para esta expresión hace uso de la regla de composición

$$\left.\begin{array}{c} \text{Si } A(x) \\ y \\ R(x,y) \end{array}\right\} \text{ entonces } (A \circ R)(y)$$

siendo:

$$\mu_{A \circ R}(y) = \underset{x \in \mathcal{U}}{max}\{min\{\mu_A(x), \mu_R(x,y)\}$$

donde R es una relación binaria difusa sobre $\mathcal{U} \times \mathcal{U}$, y $\circ$ denota la composición (unaria) del conjunto difuso A –caracterizado por el predicado $A(x)$ en el universo $\mathcal{U}$– y la relación difusa R.

A diferencia de la lógica ordinaria, el antecedente $A(x)$ no tiene por qué coincidir con $A'(x)$, y este modus ponens difuso puede verse como un caso particular de la regla de composición, sin más que elegir como relación R el producto cartesiano borroso $A \times B$.

Un modus ponens difuso aparece, por ejemplo, en [Vojtáš y Paulík, 1996], en el lenguaje f-Prolog que estudiamos en la Sección 2.2, aunque en su formalización no se contemplen relaciones borrosas.

Otras reglas de inferencia, que son la base del razonamiento aproximado, además de la regla de la composición y (el caso particular) del modus ponens son el principio de herencia, la regla de la intersección, la regla del producto cartesiano y la regla de la proyección, entre las más habituales. No las describimos aquí.

Los predicados clásicos sólo pueden modificarse por la negación. Pero si $A(x)$ es un predicado difuso, tiene sentido considerar

no $A(x)$, muy $A(x)$, algo $A(x)$, más o menos $A(x)$, aproximadamente $A(x)$,...

En efecto, pueden formalizarse estos llamados modificadores de predicado (o cercas semánticas, para los lingüistas), que son adverbios del lenguaje que matizan el uso de la propiedad $A(x)$. Así por ejemplo, podemos definir el modificador "muy" de la forma (suponiendo que $A(x)$ está definido en $\mathcal{U}$, y $x_0 \in \mathcal{U}$).

$$\mathcal{I}(muyA(x_0)) = [\mathcal{I}(A(x_0))]^2 \quad \text{o equivalentemente} \quad \mathcal{I}(muyA(x_0)) = [\mu_A(x_0)]^2$$

y el modificador "algo"

$$\mathcal{I}(algoA(x_0)) = [\mathcal{I}(A(x_0))]^{1/2} \quad \text{o equivalentemente} \quad \mathcal{I}(algoA(x_0)) = [\mu_A(x_0)]^{1/2}$$

Es de gran utilidad en el razonamiento aproximado el concepto lógico de variable lingüística, desarrollada por Zadeh [1975], cuyos valores son términos o expresiones del lenguaje natural o formal, etiquetas ligüísticas en el contexto considerado. Ejemplo de variable lingüística puede considerarse la variable *velocidad* y son valores de ésta: baja, media, alta, elevada, entre otros. Pero, ejemplo es también la variable *verdad* con valores posibles: muy verdadero, algo verdadero, falso (no verdadero), muy falso, etc., es decir, los asociados a los correspondientes modificadores que se hayan formalizado.

Por último, dejemos constancia de que, además de las diferencias ya observadas con la lógica ordinaria:

- carácter vago de los predicados
- infinitos valores de verdad
- presencia modificadores lingüísticos
- diferentes interpretaciones

es muy relevante también la presencia de cuantificadores específicos (véase [Dubois y Prade, 1980]) como son: casi, algunos, la mayoría, aproximadamente.

1.3.4. Aplicaciones. Razonamiento aproximado, control difuso y sistemas expertos.

Las nociones básicas de la lógica difusa se perfilan, por primera vez, en el artículo de Zadeh [1965b] sobre conjuntos borrosos. Aunque algunos científicos, como los matemáticos R. Bellman y G. Moisil, reciben las nuevas ideas con entusiasmo, de forma mayoritaria generan recelo y, a veces, hostilidad. Hoy en día, la controversia sobre esta lógica continúa, aunque no en el mismo grado: las numerosas aplicaciones de la lógica borrosa son contundentes. Quizá la tradición cartesiana, preocupada en ocasiones por atender a lo cuantitativo y preciso a la vez que ignora lo cualitativo e impreciso, pueda ir aceptando la extraordinaria potencia de esta materia para interpretar la realidad.

Japón es el país en el que la lógica borrosa y sus aplicaciones han tenido mayor auge, siendo los precursores, en 1968, los profesores K. Asai, K. Tanaka y T. Terano con sus trabajos sobre autómatas borrosos y sistemas de aprendizaje.

En Europa, el interés por la lógica borrosa comenzó a principios de los 70 y las aportaciones más significativas se centran en desarrollos teóricos. Hay que destacar la aportación de Mamdani y Assilian [1975] en control borroso que propicia la primera aplicación industrial importante: el control de un horno de cemento en Copenhague a finales de los 70.

El profesor E. Trillas fue quien inició en España la investigación sobre la lógica borrosa y sus aplicaciones y, en opinión de Zadeh [1996], sus aportaciones han influído notablemente para que nuestro país sea uno de los núcleos punteros en Europa en este área.

Un hito importante en el desarrollo de la lógica difusa lo marcó el seminario EE.UU.-Japón sobre conjuntos borrosos y sus aplicaciones que se celebró en Berkeley en 1974 y el Congreso IFSA (de la Asociación Internacional de Sistemas Borrosos) de Tokio en 1987. En ese congreso, Matsushita anunció el primer producto de consumo basado en lógica borrosa (un cabezal de ducha). Simultáneamente, en otro campo, comenzó a operar el sistema del metro de Sendai, que empleaba un controlador basado en lógica borrosa y está considerado como una de las aplicaciones de mayor éxito de esta lógica [Valverde y Zadeh, 1996].

La aplicación de la lógica difusa a la teoría de control ha sido natural, de hecho el calificativo "fuzzy" nace íntimamente ligado a este área. Y la evolución del control borroso ha sido espectacular. Su crecimiento ha sido muy rápido porque las aplicaciones de la lógica borrosa al control automático son sencillas de realizar al no

exigir más que reglas "si entonces" borrosas para manipular comandos. En efecto, los fundamentos del control borroso se centran en que las reglas del proceso, elaboradas generalmente por los expertos, son implicaciones borrosas en cuyos términos (antecedente y consecuente) figuran proposiciones borrosas, simples o compuestas [Driankov *et al.*, 1996; Palmn *et al.*, 1997].

Además, los controladores borrosos son simples y robustos, y en muchos casos no cabe alternativa para los controles tradicionales por no existir modelo matemático tradicional efectivo o ser inviable en la práctica [Mamdani, 1993]. Ebrahim Mamdani es sin duda el pionero en el diseño de métodos de control borroso.

Ya hemos visto cómo la lógica difusa se ocupa de la formalización de predicados vagos. Pero esta lógica aporta, además, modelos para el estudio de formas de razonamiento usuales. La lógica clásica aborda razonamientos con formulaciones muy precisas y en la lógica borrosa éste es un caso límite del razonamiento aproximado. Si ya detallamos cómo la lógica difusa facilita símbolos y operadores que recogen el concepto de vaguedad, ahora mencionaremos cómo proporciona, asimismo, reglas de inferencia que permiten acotar o extender los grados de verdad que aportan la premisas. Y es que uno de los objetivos básicos de esta lógica es procurar un cálculo adecuado para la representación del conocimiento y la obtención de inferencias en un entorno de vaguedad e imprecisión [Valverde y Zadeh, 1996].

Zadeh inicia también la teoría del razonamiento aproximado en el contexto de la inteligencia artificial; buscando herramientas más eficaces para la construcción de sistemas expertos, introduce este área.

En [Zadeh, 1973] enuncia el llamado principio de incompatibilidad, por el que resultan antagónicos la precisión y la complejidad, a la hora de describir la conducta de un sistema, por lo cual los programas convencionales tienen muy poca efectividad para modelar el comportamiento humano. Buscando herramientas eficaces para tratar la información imprecisa se plantea:

i) la representación de la información (imprecisa), mediante conjuntos difusos.

ii) la inferencia sobre información imprecisa, basada en el uso de la implicación borrosa y de la propiedad más relevante: el modus ponens generalizado, formalizado éste en la composición unaria de un conjunto borroso y una relación borrosa (la llamada regla composicional de inferencia).

Esta propiedad particular de la composición de dos relaciones borrosas desencadena la posibilidad de aplicar la lógica difusa al control difuso y permite el desarrollo de los sistemas de razonamiento y su implementación.

La selección de la función de implicación que ha de emplearse es cuestión esencial en el diseño de los sistemas de inferencia. La efectividad de las diferentes funciones de implicación, a la hora de reproducir el razonamiento humano y facilitar métodos de inferencia adecuados ha sido muy estudiado en la literatura.

El fenómeno de la vaguedad, que no es sólo lingüístico, invade el razonamiento ordinario, porque el discurso ordinario está lleno de predicados vagos, no precisables, que pierden matices al intentar delimitarlos arbitrariamente, y la inteligencia artificial se interesa por los distintos modos de este conocimiento [Trillas *et al.*, 1995].

La lógica difusa tiene gran relevancia en la ingeniería del conocimiento por dos aspectos principales: por representar formalmente el conocimiento impreciso y por gestionar adecuadamente la incertidumbre en algunos sistemas expertos. En general, esta lógica es un instrumento útil y sencillo para el tratamiento de distintas modalidades de razonamiento aproximado [Valverde y Zadeh, 1996]. En [Dubois *et al.*, 1991a; Dubois y Prade, 1991] puede encontrarse amplio detalle del interés de la lógica difusa para el razonamiento aproximado.

Un sistema experto es un sistema basado en el conocimiento al que se le incorpora información proveniente de expertos del dominio [Pajares y Santos, 2005]. Su finalidad es la resolución de problemas de este dominio, aplicando técnicas de razonamiento sobre el conocimiento que alberga su base de conocimiento.

Aunque la teoría de la probabilidad es el modelo formal clásico para representar la incertidumbre, no ha sido universalmente aceptado en el diseño de sistemas expertos para gestionar la incertidumbre. Es bien sabido que requiere una amplia colección de datos y gran número de operaciones para que resulte apropiada [Shortliffe y Buchanan, 1975; Adams, 1976]. Varios métodos, extensiones especializadas del cálculo de probabilidades, han sido desarrollados para salvar estas limitaciones, tales como los factores de certeza de MYCIN[9] [Shortliffe y Buchanan, 1975], el Bayesianismo subjetivo [Duda *et al.*, 1990], la teoría de la evidencia de Dempster-Shafer [Shafer, 1976] y la teoría de *endorsements* [Cohen, 1985]. En contraste con los métodos anteriores, carentes de un marco semántico bien conocido, L. A. Zadeh propone, como ya ha sido considerado, una lógica formal basada en la teoría de conjuntos difusos muy adecuada para tratar la incertidumbre.

Los sistemas basados en reglas difusas (FRBSs, acrónimo de Fuzzy Rule-Based Systems) son una extensión de los sistemas clásicos de representación del conocimiento basados en reglas [Cordón *et al.*, 2001]. Al igual que estos últimos, los FRBSs se

[9] MYCIN es un sistema experto escrito en Prolog.

componen de reglas condicionales de la forma "si entonces", con la particularidad de que el antecedente y el consecuente son expresiones difusas.

Son destacables las siguientes ventajas de los sistemas expertos difusos:

- Son una forma fácil de codificar un sistema no lineal.
- Tienen buena correspondencia con los esquemas de razonamiento humano sobre una gran clase de problemas matemáticos.
- Se ejecutan rápidamente sobre computadoras convencionales.
- Se ejecutan a velocidades extremadamente altas sobre hardware especializado.

La aplicación de la lógica borrosa a los sistemas basados en reglas se ha desarrollado principalmente en, de una parte, generalizar el modelo de los factores de certeza y, de otra, en el uso de predicados difusos en la descripción de las reglas y de la realidad.

En general, los métodos de razonamiento aproximado para el tratamiento de la incertidumbre se pueden agrupar en dos grandes tipos: los cualitativos, como los basados en lógicas no monótonas (si no se dispone de suficiente información se realizan hipótesis que pueden ser corregidas con nueva información) y los numéricos.

Entre éstos últimos, de interés marginal aquí, hay que considerar los probabilísticos y muy especialmente las redes bayesianas, introducidas a finales de los años 80 que originan el resurgimiento del uso de la probabilidad en los sistemas expertos. Este modelo de red bayesiana, que usan modelos gráficos para especificar distribuciones de probabilidad, no es el único modelo gráfico para modelar sistemas expertos probabilísticos, porque también lo son los grafos de Markov y las redes cadena, pero sí son los más usados (véase [Castillo *et al.*, 1996; Giarratano y Riley, 1994]).

Relacionamos, para finalizar, algunas de las múltiples aplicaciones concretas de esta lógica (en el campo del control difuso y de los sistemas expertos, principalmente).

-Electrónica de consumo: lavadoras inteligentes (Matsuhita Electronic Industrial), hornos microondas, sistemas térmicos, cámaras de vídeo, televisores, estabilizadores de imágenes digitales (Matsuhita) y sistemas de foco automático en cámaras fotográficas.

-Sistemas: sistemas de pilotaje automático de aeronaves, sistemas de depuración de aguas, sistemas de frenado automático, de suspensión automática y de conducción automática de vehículos, sistemas industriales de control de combustión, controladores de tráfico, sistemas de refrigeración/calefacción, sistemas de predicción del clima,

sistemas de predicción de fenómenos atmosféricos y sistemas de reconocimiento de escritura.

-Software de: diagnóstico médico ((CADAG, Adlssnig), Arita, OMRON), seguridad (Yamaichi, Hitachi), traducción lingüística, comprensión de datos, tecnología informática y bases de datos difusas para almacenar y consultar información imprecisa (uso del lenguaje FSQL).

En resumen, y atendiendo a la opinión de Valverde y Zadeh [1996], las aplicaciones de la lógica borrosa se dan, primordialmente, en dos campos bien diferenciados: de una parte las aplicaciones a la teoría de control y de otra las aplicaciones en el desarrollo de sistemas expertos.

Ahora bien, son de destacar finalmente las contribuciones de la lógica al *soft computing*. La aparición de la neurocomputación y los algoritmos genéticos (a mediados de los 80), tuvo un impacto significativo en el desarrollo de la lógica borrosa. La probabilidad y la lógica borrosa pueden usarse conjuntamente en las metodologías de neurocomputación y algoritmos genéticos. Esta constatación sugiere a Zadeh [1996] el concepto de *soft computing* entendido como "una especie de consorcio o sociedad entre la lógica borrosa, la neurocomputación y el razonamiento probabilístico". En este ámbito, la lógica borrosa aporta una metodología para el tratamiento de la imprecisión, el razonamiento aproximado y la computación con palabras. Lo importante del soft computing es que sugiere la posibilidad de emplear la lógica borrosa, la neurocomputación y los algoritmos genéticos de forma combinada en vez de aislada. Una de las combinaciones más destacadas en la actualidad es la de los sistemas "neuro-fuzzy" (para una introducción puede verse [Nguyen, 2002]). El empleo creciente del soft computing ha supuesto una contribución importante para la concepción, el diseño y el desarrollo de sistemas inteligentes.

En opinión de Valverde y Zadeh [1996], estamos entrando en una era de sistemas inteligentes que tendrán un impacto profundo en las formas en que nos comunicamos, tomamos decisiones y utilizamos las máquinas, y la lógica difusa, junto con sus socios en el soft computing (y también, a nuestro parecer, los lenguajes declarativos difusos), jugarán un papel importante en conseguir que la era de los sistemas inteligentes sea una realidad.

En la siguiente sección, entramos en otra de las grandes (prometedoras) aplicaciones de la lógica difusa: el diseño de lenguajes declarativos que permitan codificar con mayor facilidad las aplicaciones con rasgos difusos de los campos que referidos (sistemas basados en reglas difusas, inteligencia artificial, soft computing,...).

1.4. Programación lógica

La programación lógica ([Lloyd, 1987; Apt, 1990, 1997; Julián y Alpuente, 2007]) se basa en fragmentos de la lógica de predicados, siendo el más popular el de Cláusulas de Horn que puede emplearse como base para un lenguaje de programación al poseer una semántica operacional para la que cabe una implementación eficiente (la resolución-SLD).

Su rasgo principal es, en efecto, el uso de la lógica como lenguaje de programación, lo que puede concebirse como sigue:

- un programa es una teoría formal en una cierta lógica, y
- la computación se entiende como una forma de deducción en esta lógica.

Los principales requisitos que debe cumplir la lógica empleada son (veáse [Julián, 2000]): *i*) disponer de un lenguaje que sea suficientemente expresivo para cubrir un campo de aplicación interesante; *ii*) disponer de una semántica operacional, esto es, un mecanismo de cómputo que permita ejecutar los programas; *iii*) disponer de una semántica declarativa que permita dar un significado a los programas de forma independiente a su posible ejecución, y *iv*) poseer resultados de corrección y completitud que aseguren que lo que se computa coincide con aquello que es considerado como verdadero de acuerdo con la noción de verdad que sirve de base a la semántica declarativa. Además, esta semántica declarativa especifica el significado de los objetos sintácticos del lenguaje por medio de su traducción en elementos y estructuras de un dominio (generalmente matemático) conocido.

El mecanismo operacional de los programas lógicos se fundamente en una particularización de la estrategia de resolución, llamada resolución-SLD, que es un método de prueba por refutación que emplea el algoritmo de unificación como mecanismo de base y permite la extracción de respuestas, es decir el enlace de un valor a una variable lógica. Es un refinamiento de la resolución de Robinson, que ha sido descrito por primera vez por Kowalski [1974], y cuyas siglas hacen mención a las iniciales de "resolución lineal con función de selección para programas definidos". Además, es un método de prueba correcto y completo para la lógica referida.

Como semántica declarativa puede usarse la teoría de modelos que toma como dominio un universo puramente sintáctico: el universo de Herbrand.

Introducimos a continuación algunas nociones básicas de la programación lógica que usaremos en capítulos posteriores. El lector que desee mayor detalle, puede ver [Lloyd, 1987; Julián y Alpuente, 2007].

Denotamos por $\mathcal{V}$ un conjunto infinito (numerable) de variables y por Σ un conjunto de símbolos de función f/n, cada uno con una aridad n asociada. $\mathcal{T}(\Sigma, \mathcal{V})$[10] y $\mathcal{T}(\Sigma)$ denotan, respectivamente, el conjunto de términos y términos básicos (sin variables) construidos sobre $\Sigma \cup \mathcal{V}$ y Σ, respectivamente. El conjunto de variables que aparecen en una expresión E se denota por $\mathcal{V}ar(E)$. Un término t es básico si $\mathcal{V}ar(t) = \varnothing$.

Definición 1.4.1 ([Julián y Alpuente, 2007])**.** *Una sustitución σ es una aplicación $\sigma : \mathcal{V} \longrightarrow \mathcal{T}$ que asigna a cada variable x del conjunto de variables $\mathcal{V}$ de un lenguaje de primer orden $\mathcal{L}$, un término $\sigma(x)$ del conjunto de términos $\mathcal{T}$.*

Es costumbre exigir que $\sigma(x) \neq x$ sólo para un número finito de variables, y también expresar la sustitución en términos conjuntistas, identificando (en cierto sentido) la aplicación σ con el conjunto de imágenes. Es decir, escribiremos $\sigma = \{x_1/t_1, \ldots, x_n/t_n\}$, donde $t_i = \sigma(x_i)$ es un término diferente de x_i y a cada par x_i/t_i le diremos enlace o elemento de la sustitución.

Llamaremos dominio de σ al conjunto $\mathcal{D}om(\sigma) = \{x \in \mathcal{V} : \sigma(x) \neq x\} = \{x_1, \ldots, x_n\}$ y rango de σ al conjunto $\mathcal{R}an(\sigma) = \{\sigma(x) : x \in \mathcal{D}om(\sigma)\} = \{t_1, \ldots, t_n\}$. Además, representaremos por id la sustitución identidad que puede entenderse como el conjunto vacío de enlaces, de modo que $\mathcal{D}om(id) = \varnothing$, es decir, $id(x) = x$, para toda $x \in \mathcal{V}$. Diremos también que σ es básica si los términos t_i son básicos (no contienen variables).

Definición 1.4.2. *Dada una expresión E y una sustitución σ, se llama instancia $\sigma(E)$ al resultado de aplicar σ sobre E, reemplazando simultáneamente cada ocurrencia de x_i en E por el correspondiente término t_i, siendo x_i/t_i un elemento de la sustitución σ.*

Es costumbre denotar la instancia anterior por $E\sigma$ en lugar de $\sigma(E)$. Cuando la sustitución se aplica a fórmulas más generales del lenguaje $\mathcal{L}$ y no sólo a expresiones de un lenguaje clausal, conviene renombrar las variables ligadas antes de aplicar la sustitución (véase, [Julián, 2004]).

Definición 1.4.3. *Dadas las sustituciones $\sigma = \{x_1/t_1, \ldots, x_n/t_n\}$, $\theta = \{y_1/s_1, \ldots, y_m/s_m\}$, la composición $\sigma \circ \theta$[11] es la sustitución determinada a partir el conjunto $\sigma \circ \theta = \{x_1/\theta(t_1), \ldots, x_n/\theta(t_n), y_1/s_1, \ldots, y_m/s_m\}$ eliminando los enlaces $x_i/\theta(t_i)$ tal que $x_i = \theta(t_i)$ y eliminando en θ los enlaces y_j/s_j tal que $y_j \in \{x_1, \ldots, x_n\}$.*

[10] En ocasiones escribiremos sólo $\mathcal{T}$.

[11] En ocasiones escribiremos $\sigma\theta$ en lugar de $\sigma \circ \theta$.

Esta composición verifica sobre una expresión E que $(\sigma \circ \theta)(E) = \sigma(\theta(E))$, es asociativa y la sustitución identidad es el elemento neutro (por la izquierda y la derecha). Por otra parte, dadas σ, θ con $\mathcal{V}ar(\sigma) \cap \mathcal{V}ar(\theta) = \varnothing$, se define la unión $\sigma \cup \theta$ por el conjunto unión de ambas, es decir, $(\sigma \cup \theta)(x) = \sigma(x), x \in \mathcal{D}om(\sigma)$ y $(\sigma \cup \theta)(x) = \theta(x), x \in \mathcal{D}om(\theta)$.

Una sustitución ρ se llama sustitución renombramiento o simplemente renombramiento si existe ρ^{-1} (sustitución inversa) tal que $\rho \circ \rho^{-1} = \rho^{-1} \circ \rho = id$. Dos expresiones E_1, E_2 son variantes si existen sustituciones de renombramiento ρ, ρ' tales que $E_1 = \rho(E_2)$ y $E_2 = \rho'(E_1)$. La composición de sustituciones induce el preorden (subsumción) habitual entre sustituciones: $\theta \leq \sigma$ si, y sólo si, existe γ tal que $\sigma = \theta \circ \gamma$ y decimos que θ es más general que σ. Este preorden induce un (pre-)orden parcial sobre términos dado por $t \leq t'$ si existe γ tal que $t' = t\gamma$.

Dos términos t y t' son variantes (uno de otro) si existe un renombramiento ρ tal que $t\rho = t'$. Dada una sustitución θ y un conjunto de variables $W \subseteq \mathcal{V}$, denotamos por $\theta_{\restriction W}$ la sustitución obtenida de θ restringiendo $Dom(\theta)$ a las variables de W. Escribimos $\theta = \sigma\ [W]$ si $\theta_{\restriction W} = \sigma_{\restriction W}$, y $\theta \leq \sigma\ [W]$ denota la existencia de una sustitución γ tal que $\theta \circ \gamma = \sigma\ [W]$.

En la siguiente definición, se aborda el concepto de unificación que resulta fundamental en la programación lógica y en la demostración automática ([Julián y Alpuente, 2007]). De forma intuitiva, unificar dos expresiones supone lograr que ambas sean sintácticamente iguales después de aplicar sobre ellas una sustitución que llamaremos unificador (es decir, serán sintácticamente iguales las instancias que resulten de ambas mediante alguna sustitución).

Definición 1.4.4. *Una sustitución θ es un unificador de las expresiones E_1, E_2 si, y sólo si, $\theta(E_1) = \theta(E_2)$.*

Esta definición se extiende de manera natural a un número finito de expresiones $E_1, \ldots, E_n$ y hablaremos también del unificador del conjunto $S = \{E_1, \ldots, E_n\}$.

Definición 1.4.5 (Julián y Alpuente [2007])**.** *Un unificador σ de un conjunto de expresiones S es un unificador más general (o de máxima generalidad) para S si, y sólo si, cualquier otro unificador θ es tal que $\sigma \leq \theta$.*

Denotaremos por *mgu* (del inglés, *most general unifier*) al *unificador más general* de un conjunto de expresiones, que siempre existe (si la expresiones unifican) y es único salvo (composición con sustituciones de) renombramiento (véase [Lassez *et al.*, 1988]).

1.5. Programación lógica difusa

Desde un punto de vista formal, la programación lógica difusa es un área de la lógica difusa dedicada al estudio de teorías difusas o programas difusos, que son un conjunto de expresiones lógicas difusas en un lenguaje de primer orden directamente ejecutables en un computador.

La programación lógica ([Lloyd, 1987; Apt, 1997; Julián y Alpuente, 2007] son referencias fundamentales en este contexto) se ha usado ampliamente para resolver problemas y para representación del conocimiento [Meritt, 1989; Sterling y Shapiro, 1986]. Sin embargo, los lenguajes tradicionales de la programación lógica no incorporan técnicas que permitan tratar, de manera explícita, la vaguedad y el razonamiento aproximado.

La programación lógica difusa es un área de investigación que aglutina el esfuerzo de introducir la lógica difusa en la programación lógica. Si, debido a su alto nivel de abstracción y expresividad, los lenguajes declarativos son ampliamente reconocidos como una de las mejores herramientas para el desarrollo de aplicaciones con una fuerte componente simbólica, como ocurre a menudo en el campo de la inteligencia artificial, los lenguaje basados en lógica difusa aportan una habilidad natural para modelar escenarios de naturaleza vaga o imprecisa y formular razonamiento aproximado.

Como observaremos en lo que sigue, el área se encuentra en un estado relativamente incipiente, no existiendo estándares claros sino distintas aproximaciones sin que, por el momento, ninguna de ellas sea totalmente hegemónica ni aglutine las restantes opciones.

La gestión de la incertidumbre y/o de la imprecisión en los procesos de deducción es una cuestión relevante una vez que en el mundo real la información a procesar es de naturaleza imperfecta. En programación lógica, este asunto ha atraído la atención de muchos investigadores y han sido propuestos distintos marcos para su formalización. En esencia, difieren en la noción subyacente de teoría de la incertidumbre y teoría de la imprecisión que contemplan:

- teoría de la probabilidad [Baral *et al.*, 2004; Dekhtyar y Dekhtyar, 2005; Dekhtyar y Subrahmanian, 2000; Fuhr, 2000; Lakshmanan y Sadri, 1994, 2001; Lukasiewicz, 1999b, 2001b, 1999a, 2001a; Muggleton, 1995; Ng y Subrahmanian, 1991, 1992; Ngo y Haddawy, 1997; Wüthrich, 1995],

- teoría de conjuntos difusos [Cao, 2000; Hinde, 1986; Ishizuka y Kanai, 1985;

Klawonn y Kruse, 1994; Martin *et al.*, 1987; Mukaidono *et al.*, 1989; Shapiro, 1983; van Emden, 1986; Vojtáš y Paulík, 1996; Vojtáš, 2001; Yasui *et al.*, 1995],

- lógica multi-valorada [Damásio *et al.*, 2004a,b; Damasio *et al.*, 2004; Damásio y Moniz-Pereira, 2001b,a, 2004; Fitting, 1991; Kifer y Li, 1988; Kifer y Subrahmanian, 1992; Lakshmanan y Sadri, 1997; Lakshmanan y Shiri, 2001; Loyer y Straccia, 2002a,b, 2003, 2006; Lu, 1996; Mateis, 2000; Medina *et al.*, 2004, 2001d,c; Medina y Ojeda-Aciego, 2004; Medina *et al.*, 2001d,b,a; Medina y Ojeda-Aciego, 2002; Damásio *et al.*, 2007; Straccia, 2005b,a],

- lógica posibilista [Alsinet *et al.*, 2008; Alsinet y Godo, 2000; Dubois *et al.*, 1991b, 1998; Nicolas *et al.*, 2005; Chesñevar *et al.*, 2004],

y en cómo son gestionados los valores de incertidumbre/imprecisión asociados a las reglas y hechos.

Los marcos actuales para gestionar la imprecisión en programación lógica difusa pueden clasificarse en basados en anotaciones y basados en implicaciones (véanse a tal efecto [Lakshmanan y Shiri, 2001; Loyer y Straccia, 2005]).

- Los basados en anotaciones (véase por ejemplo [Kifer y Li, 1988; Kifer y Subrahmanian, 1992; Ng y Subrahmanian, 1991, 1992; Lu, 1996; Cao, 2000; Krajči *et al.*, 2002]), admiten reglas de la forma $A : f(\beta_1, \ldots, \beta_n) \longleftarrow B_1 : \beta_1, \ldots, B_n : \beta_n$ cuyo significado puede entenderse del modo "el grado de verdad de A es al menos $f(\beta_1, \ldots, \beta_n)$ siempre que el grado de verdad de cada átomo B_i sea al menos $\beta_i, 1 \leq i \leq n$", siendo f una función computable y β_i es una constante o una variable sobre un dominio (conjunto de valores de verdad) apropiado.

- Los basados en implicaciones (véase por ejemplo [Damásio *et al.*, 2004a; Damásio y Moniz-Pereira, 2004; Lakshmanan y Sadri, 1994; Lakshmanan y Shiri, 2001; Medina *et al.*, 2001d,c,b,a; Medina y Ojeda-Aciego, 2002; Medina *et al.*, 2004; Medina y Ojeda-Aciego, 2004; Damásio *et al.*, 2007; van Emden, 1986; Vojtáš, 2001]) tienen reglas que podemos dar del modo $\langle A \leftarrow @(B_1, \ldots, B_n); v \rangle$ donde v es el grado de verdad asociado a la fórmula $A \leftarrow @(B_1, \ldots, B_n)$, en la que $@$ es una conectiva que combina las expresiones atómicas B_i.

  Computacionalmente, dada una interpretación $\mathcal{I}$, los valores de verdad $\mathcal{I}(B_i)$ se calculan conforme determina la función de verdad de la conectiva $@$ y posteriormente se propagan convenientemente al átomo A de la cabeza de la regla.

Además, los grados de verdad pueden tomarse en un retículo, es decir, la aplicación $\mathcal{I}$ puede tomar valores sobre un cierto retículo. Puede verse en [Damásio *et al.*, 2004a; Krajči *et al.*, 2004; Lakshmanan y Shiri, 2001; Vojtáš, 2001] cómo la mayoría de los marcos que manejan imprecisión y programación lógica pueden ser contemplados en este contexto.

Es de destacar que la mayoría de las aproximaciones no contemplan ningún tipo de razonamiento no-monótono ni admiten negación. No es el caso de [Damásio y Moniz-Pereira, 2001a; Loyer y Straccia, 2006, 2002a,b, 2003; Straccia, 2005a,b], en donde el dominio de valores de verdad es un retículo. También [Vaucheret *et al.*, 2002; Guadarrama *et al.*, 2004] admiten negación, pero aquí los grados de verdad son intervalos reales [Klir y Yuan, 1995] lo que resulta muy adecuado para formalizar variables lingüísticas como la variable edad o la variable velocidad.

Deteniéndonos en la negación, digamos que es el concepto lógico más relevante que no ha sido contemplado originariamente por la programación lógica difusa, debido a que su incorporación conlleva una complejidad significativa. Sin embargo, la negación juega un papel importante en la representación del conocimiento en donde muchos de sus usos no pueden ser simulados por programas positivos. La negación es también útil en la gestión de bases de datos, composición de programas, razonamiento por defecto, etc.

En cuanto al mecanismo operacional contemplado en las distintas aproximaciones, observamos que en muchos de ellos se reemplaza el mecanismo clásico de inferencia, la resolución-SLD, por una variante difusa que permita razonar con incertidumbre y evaluar grados de verdad. La mayoría de estos implementan el principio de resolución difuso introducido por Lee [1972] (extendido por Mukaidono [1982] y por Weigert *et al.* [1993]), como el sistema Prolog-Elf [Ishizuka y Kanai, 1985], el sistema Fril Prolog [Baldwin *et al.*, 1995] y el lenguaje F-Prolog [Li y Liu, 1990].

Sin embargo, no hay un método común para "difuminar" el principio de resolución de Prolog (véase [Vaucheret *et al.*, 2002]): lenguajes difusos como el lenguaje Likelog considerado en [Arcelli y Formato, 1999] sólo contemplan la componente borrosa en los predicados introduciendo la noción de similaridad entre ellos, Atanassov y Georgiev [1993] implementa una modalidad de resolución concebida para gestionar grados de verdad de las cláusulas que son intervalos (cada uno de sus extremos representa el grado de verdad y de falsedad), Li y Liu [1990] considera expresiones difusas que incorporan cercas semánticas, Mukaidono *et al.* [1989] contempla expresiones que tienen asociada una cierta confianza obtenida a partir de su grado de verdad

y se establece la confianza del resolvente, de la correspondiente resolución difusa, a partir de la confianza de las cláusulas originales, y otros sistemas como, por ejemplo [Vojtáš y Paulík, 1996; Medina *et al.*, 2001d,c, 2004; Medina y Ojeda-Aciego, 2004], consideran hechos difusos y/o reglas difusas etiquetando cláusulas de programa con números reales (o, más generalmente, elementos de un retículo) que representan el grado de verdad asociado. En resumen, en estos lenguajes, pueden ser diferentes los modos en que se representa el conocimiento difuso y también los métodos elegidos para gestionar dicho conocimiento.

Las propiedades de corrección y completitud para los diferentes tipos de semántica operacional han sido propuestas en relación con una semántica declarativa apropiada, que en muchos casos ha sido concebida como una extensión difusa del clásico modelo mínimo de Herbrand [Lloyd, 1987].

Respecto al principio de resolución clásico, basado en la unificación sintáctica, encontramos dos aproximaciones a la hora de enriquecerlo con rasgos difusos:

- La primera aproximación, representada por lenguajes que reemplazan el mecanismo de la unificación sintáctica, de la clásica resolución-SLD, por un algoritmo de unificación difusa.

  En esta línea se encuentran trabajos como el de Virtanen [1991] donde se presenta un algoritmo de unificación difuso basado en relaciones de equivalencia borrosas. Rios-Filho y Sandri [1995] hacen una distinción entre conocimiento específico y general con el objetivo de direccionar el algoritmo de unificación borrosa, el llamado algoritmo de *unificación contextual*. Otros trabajos intentan construir algoritmos de unificación borrosa basados en relaciones de compatibilidad difusas, con el objetivo de incluir el tipo de datos conjunto difuso en el marco de la programación lógica (véase por ejemplo [Alsinet y Godo, 1998]).

  Por otro lado están aquellos planteamientos de Arcelli y Formato [2002]; Formato *et al.* [1999, 2000]; Sessa [2000, 2002], menos expresivos pero más eficientes desde el punto de vista de su implementación, que sustituyen el mecanismo de unificación sintáctico de la resolución-SLD clásica, por un algoritmo de *unificación débil*, basado en relaciones de similaridad (puede verse en [Klawonn, 2003] el concepto de similaridad). Con ello se obtiene un mecanismo operacional denominado *resolución débil* o *resolución-SLD basada en similaridad* en [Sessa, 2002]. Mientras que el mecanismo global de resolución permanece básicamente inalterado, la unificación cambia drásticamente, contemplando la posibilidad de "igualar" sintácticamente símbolos (de constante, función o

predicado) distintos, siempre que exista un cierto grado de similaridad entre ellos. Este algoritmo, proporciona un *unificador más general débil*, así como un valor numérico, llamado *grado de unificación*. Intuitivamente, el grado de unificación representa el grado de verdad asociado con la instancia computada de la pregunta. Los programas escritos en esta clase de lenguajes consisten, en esencia, en un conjunto de cláusulas, junto a un conjunto de "ecuaciones de similaridad" que juegan un papel importante durante el proceso de unificación [Sessa, 2002]. Esta última línea de trabajo ha llevado a la implementación de dos lenguajes [Arcelli y Formato, 1999; Loia *et al.*, 2001], que desgraciadamente no están accesibles al público y de los que se dispone limitada documentación.

- En la segunda aproximación, los programas son un subconjunto de cláusulas con un grado de verdad asociado que está explícitamente anotado. El trabajo de computación y propagación de los grados de verdad se realiza a través de una semántica operacional que es una una extensión del principio de resolución clásico, mientras el mecanismo de unificación (sintáctica) permanece inalterado.

  Ejemplos de este tipo de lenguajes son el descrito en [Vojtáš y Paulík, 1996; Vojtáš, 2001] o el presentado en [Vaucheret *et al.*, 2002] que incorpora negación y la lógica subyacente usa intervalos en vez de números reales para representar los grados de verdad [Klir y Yuan, 1995].

  En nuestra opinión, el contexto más interesante lo representa la programación multi-adjunta considerada en [Medina *et al.*, 2001d,c,b,a; Medina y Ojeda-Aciego, 2002; Medina *et al.*, 2004; Medina y Ojeda-Aciego, 2004; Damásio *et al.*, 2007], en la que los grados de verdad se toman frecuentemente en un retículo y la sintaxis permite incorporar lógicas distintas en las reglas. Ha sido concebida como una generalización de la programación lógica monótona y residuada considerada en [Damásio y Moniz-Pereira, 2000; Dekhtyar y Subrahmanian, 2000; Damásio y Moniz-Pereira, 2001b, 2002, 2004] que, a su vez, captura (entre otros) los programas lógicos probabilísticos híbridos de Dekhtyar y Subrahmanian [2000], los programas de bases de datos deductivas probabilísticas de Lakshmanan y Sadri [2001], los programas lógicos probabilísticos ordinarios de Lukasiewicz [2001b] y los programas del marco de deducción cuantitativa de van Emden [1986] (que, a su vez, extienden a los programas de Dubois *et al.* [1991b]).

Además, como ya se ha referido, no hay unanimidad acerca de la lógica difusa usada en los distintos contextos. La mayoría de los sistemas usan la lógica min-max (para modelar la conjunción y la disyunción), pero algunos sistemas usan la lógica de Lukasiewicz [Klawonn y Kruse, 1994]. Otras aproximaciones permiten una interpretación más genérica de las conectivas [Vojtáš y Paulík, 1996], y la programación multi-adjunta admite además distintas lógicas para las conectivas de las reglas.

En cuanto al conjunto sobre el que se interpretan las fórmulas[12], nos encontramos con programas lógicos difusos que se interpretan en:

1. El intervalo $[0, 1]$, como es el caso de [Mukaidono *et al.*, 1989; van Emden, 1986; Shapiro, 1983; Vojtáš y Paulík, 1996; Vojtáš, 2001; Arcelli y Formato, 1999; Krajči *et al.*, 2004].

2. Un retículo, como por ejemplo [Medina *et al.*, 2001d,c,b; Medina y Ojeda-Aciego, 2002; Medina *et al.*, 2004; Medina y Ojeda-Aciego, 2004] (retículo multi-adjunto) y también [Damásio y Moniz-Pereira, 2000, 2001b, 2002; Damásio *et al.*, 2004b] (retículo residuado).

3. Un biretículo [Ginsberg, 1988; Fitting, 1991; Gerla, 2005; Loyer y Straccia, 2004; Straccia, 2005b], triretículo [Lakshmanan y Sadri, 2001] o, más generalmente, en un multiretículo [Medina *et al.*, 2005, 2000, 2006, 2007c,b,a].

4. Un conjunto de intervalos, como [Vaucheret *et al.*, 2002; Guadarrama *et al.*, 2004; Lakshmanan y Sadri, 2001; Lukasiewicz, 2001b; Atanassov y Georgiev, 1993].

5. Un dominio de cualificación, como en el caso [Caballero *et al.*, 2008; Rodríguez-Artalejo y Romero-Díaz, 2008, 2009][13].

Finalmente, observemos que la programación lógica difusa puede utilizarse también en el control borroso como se contempla en [Gerla, 2001, 2005; Hájek, 1998]. Gerla [2005] admite, también en este contexto, el tratamiento de información positiva y negativa si los grados de verdad se toman sobre un birretículo.

[12] En otros contextos se han usado estructuras más generales para interpretar fórmulas, como son los dominios algebraicos (véase [Rounds y Zhang, 2001; Scott, 1982]).

[13] En la actualidad, estamos interesados en contrastar este marco con el marco multi-adjunto que sirve de soporte a la mayoría de los desarrollos que se describen en esta tesis.

Capítulo 2

Algunos lenguajes lógicos difusos

En este capítulo consideramos los distintos lenguajes que hemos analizado para abordar sobre ellos la transformación de desplegado y adaptar, asimismo, los conceptos clásicos de evaluación parcial.

El primero es el lenguaje LIKELOG (acrónimo de "Likeness in Logic"), que no permite de hecho realizar eficazmente esta transformación salvo que modifiquemos –al menos– su sintaxis y su semántica operacional, tal como justificamos en lo que sigue. Pese a este inconveniente, seguimos interesados en dicho lenguaje porque contempla una unificación por similaridad, concepto éste que, aunque haya sido considerado en ámbitos parciales, está ya muy asentado en la programación lógica difusa (por ejemplo, resulta muy interesante para disponer de un sistema de *constestación de preguntas* (en inglés, *query answering*) flexible [Vojtáš, 2001]). Por otra parte, sabemos que los programas de LIKELOG podrían ser emulados en el lenguaje multiadjunto de Medina *et al.* [2004], para el que se puede formular la transformación de desplegado tal como describiremos en el Capítulo 4.

Posteriormente, hemos elegido el lenguaje descrito por Vojtáš y Paulík [1996], por ser una refencia clásica en la programación lógica difusa (generaliza, entre otros, el marco de la dedución cuantitativa de van Emden [1986]) y resultar adecuado para la formalización del concepto de desplegado de programas difusos.

Describimos a continuación este lenguaje que llamamos f-Prolog y realizamos sucesivas ampliaciones del mismo.

La primera ampliación la representa lf-Prolog, el lenguaje (etiquetado) intermedio imprescindible para el desplegado, ya que al desplegar un programa f-Prolog nos encontramos con el inconveniente (que en el contexto clásico no se presenta) de que el programa transformado resultante no pertenece al lenguaje f-Prolog del programa original. Esta ampliación lf-Prolog, que planteamos en [Julián *et al.*, 2004b], contempla marcas etiquetadas que son muy expresivas para describir el mecanismo operacional.

Las siguientes ampliaciones las representan los lenguajes ef-Prolog y lef-Prolog [Julián *et al.*, 2004a, 2005c] en los que permitimos distintas interpretaciones de las conectivas, para dar una mayor versatilidad a la hora de estimar la relevancia de cada cláusula del programa.

En el marco de la programación lógica difusa, el lenguaje multi-adjunto destaca, como precisaremos en la Sección 2.5, por su potencia expresiva. Y es apropiado para abordar el desplegado debido a la sintaxis sencilla y a que la semántica operacional original permite propagar los grados de verdad de las reglas originales y transformadas. Es el marco más general elegido para formalizar la transformación de desplegado, estudiar sus propiedades de corrección y de eficiencia; y para estudiar, asimismo, la evaluación parcial en el contexto difuso y su aplicación al cálculo de reductantes.

En este capítulo se describen las semánticas operacionales de todos los lenguajes considerados (f-Prolog, ef-Prolog –junto con sus versiones etiquetadas– y el lenguaje multi-adjunto) y aportamos los primeros resultados de esta tesis adaptando, en cada uno de los contextos, el concepto de independencia de la regla de computación, de manera semejante a como es formulada por Lloyd [1987] en la programación lógica pura.

Hemos logrado demostrar, en el Teorema 2.4.2 [Julián *et al.*, 2004b], en el Teorema 2.4.3 [Julián *et al.*, 2005c] y en el Teorema 3.1.11 [Julián *et al.*, 2005a] (éste resultado ya en el Capítulo 3), la independencia de la regla de computación para los distintos lenguajes analizados (f-Prolog, ef-Prolog y multi-adjunto).

Además, para el lenguaje multi-adjunto, reformulamos la fase interpretativa contemplándolo como un sistema de transición de estados, aspecto determinante para la definición del desplegado de estos programas y, también, para realizar un análisis del coste computacional de los mismos.

Por último, en el grupo DEC-TAU de nuestra universidad, hemos implementado una herramienta para traducir programas lógicos multi-adjuntos a programas Prolog,

que pretende ser una plataforma de gran utilidad para la optimización de programas difusos (entre otras ampliaciones y desarrollos que podamos incluir en el futuro). Con este sistema, libremente disponible en http://www.dsi.uclm.es/investigacion/dect/-FLOPERpage.html, ya es posible compilar, ejecutar y manipular programas lógicos difusos, al igual que generar y visualizar árboles de desplegado.

2.1. El lenguaje LIKELOG

En este apartado utilizamos el lenguaje LIKELOG tal cual se define en [Arcelli y Formato, 1999]. Sintácticamente, un programa LIKELOG $\mathcal{P}$ esta compuesto por dos componentes que llamaremos P y F, respectivamente:

- P es un conjunto de claúsulas Prolog y
- F es un conjunto de anotaciones (ecuaciones) del tipo: $eq(p1/a, p2/a) = n$, donde $p1/a$ y $p2/a$ son dos símbolos de predicado o de constante distintos (con aridad a) y n es un número que oscila entre 0 y 1 y que indica el grado de *similaridad* entre ambos.

La componente F aporta la noción de *borrosidad* sobre P. La semántica operacional se consigue al sustituir en la regla clásica de resolución, el algoritmo estándar de unificación sintáctica por otro de *unificación basada en similaridad*, que extiende al original al hacer uso de las anotaciones de F.

Dado un objetivo G, las respuestas esperadas tienen la forma (σ, n), donde σ es una *sustitución respuesta computada* (*computed answer substitution* o c.a.s) al estilo clásico de Prolog, y n un número que oscila entre 0 y 1 que viene a representar el grado de verdad de la respuesta.

Ejemplo 2.1.1. *Sea un $\mathcal{P}_1 = P_1 \cup F$ programa LIKELOG, donde el conjunto de claúsulas prolog es*

$$P_1 = \{\mathtt{C_1} \equiv \mathtt{p(X)} :- \mathtt{q(X)}.,\quad \mathtt{C_2} \equiv \mathtt{q(a)}.,\quad \mathtt{C_3} \equiv \mathtt{r(b)}.\}$$

mientras que la componente borrosa es

$$F = \{\mathtt{E_1} \equiv \mathtt{eq(q,r)} = 0.7\}$$

Para ejecutar el objetivo $\mathtt{p(X)}$ *en $\mathcal{P}_1$, damos pasos de resolución con unificación difusa (etiquetando cada paso con la cláusula usada, la c.a.s. acumulada y el grado de verdad propagado) obteniendo el siguiente par de derivaciones:*

- *La primera derivación es idéntica a otra clásica donde no se explota la componente difusa F a la hora de unificar:*

$$\mathtt{p(X)} \rightarrow_{(\mathtt{C}_1,\epsilon,1)} \mathtt{q(X)} \rightarrow_{(\mathtt{C}_2,\{\mathtt{X}\mapsto\mathtt{a}\},1)} \rightarrow \square,$$

  que se corresponde con la solución $\mathtt{X} \mapsto \mathtt{a}$ *con máximo grado de verdad.*

- *La segunda derivación explota en su segundo paso una unificación por similaridad entre* `q` *y* `r`*:*

$$\mathtt{p(X)} \rightarrow_{(\mathtt{C}_1,\epsilon,1)} \mathtt{q(X)} \rightarrow_{(\mathtt{E}_1,\epsilon,0.7)} \mathtt{r(X)} \rightarrow_{(\mathtt{C}_3,\{\mathtt{X}\mapsto\mathtt{b}\},0.7)} \square$$

  que se corresponde con la solución $\mathtt{X} \mapsto \mathtt{b}$ *con grado de verdad 0.7.*

Al intentar plantear una regla de desplegado en este contexto, encontramos que la transformación no debe afectar a F, ya que esta componente contiene simples anotaciones que no admiten cambios.

Por tanto, nuestro desplegado actuará sobre las claúsulas de P al estilo clásico, en los términos que se expresa en la siguiente definición.

Definición 2.1.2. *Sea $\mathcal{P} = P \cup F$ un programa LIKELOG y sea*

$$C \equiv (A \leftarrow B) \in P$$

una cláusula de programa (no unitaria). Entonces, el desplegado difuso del programa $\mathcal{P}$ con respecto a la cláusula C es el nuevo programa:

$$\mathcal{P}' = (P - \{C\}) \cup F \cup \{A\sigma \leftarrow B' | B \rightarrow_{C',\sigma,p} B'\}$$

El primer aspecto llamativo de la definición es que resulta imposible dar cuenta de la componente borrosa (es decir, del nuevo grado de verdad calculado) sobre las claúsulas desplegadas ya que éstas no admiten en su sintaxis la posibilidad de ser etiquetadas con algún grado de verdad (que sería necesario para las computaciones en el programa transformado).

Ejemplo 2.1.3. *Volviendo al Ejemplo 2.1.1, los segundos pasos de las dos derivaciones podrían darse sobre la cláusula* $\mathtt{C}_1$ *para obtener un desplegado de la misma y generar así el programa transformado $\mathcal{P}_2 = P_2 \cup F$ donde:*

$$P_2 = \{\mathtt{C}_4 \equiv \mathtt{p(a).}, \quad \mathtt{C}_5 \equiv \mathtt{p(b).}, \quad \mathtt{C}_2 \equiv \mathtt{q(a).}, \quad \mathtt{C}_3 \equiv \mathtt{r(b).}\}$$

Y ahora las derivaciones que tenemos para el mismo objetivo $\mathtt{p(X)}$ *en $\mathcal{P}_2$ son:*

- *La primera derivación da la misma solución que la primera que dimos en el programa original, aunque, como era de esperar, ahora lo hacemos de forma más eficiente ya que nos ahorramos un paso de resolución (el que dimos al desplegar):*

$$\mathtt{p(X)} \rightarrow_{(\mathtt{C_4},\{\mathtt{X}\mapsto\mathtt{a}\},1)} \rightarrow \square$$

- *La segunda derivación también es más eficiente que en el original, pero ahora ya no usamos unificación por similaridad (que de alguna manera debió quedar subsumida –aunque no lo hizo– en la regla desplegada) y se olvida de propagar el grado de verdad correcto (0.7):*

$$\mathtt{p(X)} \rightarrow_{(\mathtt{C_5},\{\mathtt{X}\mapsto\mathtt{b}\},1)} \square$$

  que se corresponde con la solución

$$\mathtt{X} \mapsto \mathtt{b}$$

  con (ERRÓNEAMENTE) total confianza o máximo grado de verdad.

El problema mostrado en el ejemplo anterior es que la información sobre la actualización de los grados de verdad que se calculan al hacer los pasos de desplegado, no queda "anotada" sobre la regla resultante, y eso impide tenerla en cuenta en los programas transformados.

Una forma de solucionarlo es permitir este tipo de anotaciones en las cláusulas del programa, pero entonces ya no estaríamos hablando de programas LIKELOG.

Es decir, sería necesario modificar/extender la sintaxis del lenguaje y, lo que es más relevante, modificar la semántica operacional para que el desplegado de programas LIKELOG fuera viable, por lo que abandonamos esta vía por el momento.

Afortunadamente, el lenguaje que veremos en la siguiente sección sí se hace eco de estas anotaciones (aunque al final de la sección evidenciaremos ciertos problemas colaterales) incluyéndolas de forma estándar en su sintaxis.

2.2. El lenguaje f-Prolog

Entre la variedad de lenguajes de programación lógica difusa existentes en la literatura, el descrito en [Vojtáš y Paulík, 1996] es una de las primeras referencias clásicas y resulta apropiado para definir el concepto de desplegado de programas lógicos difusos. Presentamos este lenguaje, que llamamos f-Prolog (Prolog *fuzzy*).

Sea $\mathcal{L}$ un lenguaje de primer orden conteniendo variables, constantes, símbolos de función, símbolos de predicado, cuantificadores $\forall$ y $\exists$, y conectivas $\neg$, seq, et_1 y et_2 (se entiende seq como una implicación $\rightarrow$ –la versión $\leftarrow$ flecha izquierda se escribe qes–, et_1 es la conjunción presente en el *modus ponens* con seq, y et_2 es la conjunción del cuerpo de las cláusulas.

Aunque et_1 y et_2 son conectivas binarias, permitiremos su generalización para contemplarlas como funciones de n argumentos.

Una *cláusula* (definida) es una fórmula

$$\forall(A \text{ qes } \mathrm{et}_2(B_1, \ldots, B_n))$$

y un *objetivo* (definido) es una fórmula

$$\forall(\text{qes } \mathrm{et}_2(B_1, \ldots, B_n))$$

donde A y cada B_i son fórmulas atómicas.

En general, les llamaremos $(\mathrm{seq}, \mathrm{et}_2)$-*fórmulas* o simplemente *fórmulas* si las conectivas usados no son relevantes o se deducen del contexto.

Usaremos también la notación

$$A \leftarrow B_1, \ldots, B_n$$

como azúcar sintáctico de $\forall(A \text{ qes } \mathrm{et}_2(B_1, \ldots, B_n))$. Igual que en programación lógica, A se dice la cabeza de la cláusula y $B_1, \ldots, B_n$ el cuerpo.

A las cláusulas con un cuerpo vacío les llamamos *hechos*, mientras que las cláusulas con cabeza y cuerpo serán *reglas*. Un tipo de cláusula degenerada es la *cláusula vacía*, simbolizada por '$\square$' y que representa una fórmula contradictoria.

Definición 2.2.1 ([Vojtáš y Paulík, 1996]). *Una* teoría difusa *es una aplicación $\mathcal{T}$ que interpreta fórmulas sobre números reales del intervalo* $(0, 1]$.

La aplicación $\mathcal{T}$ interpreta las restantes fórmulas del lenguaje que no están en $\mathrm{dom}(\mathcal{T})$ como 0.

Definición 2.2.2 ([Vojtáš y Paulík, 1996]). *Un* programa f-Prolog definido, *$\mathcal{P}$, es una teoría difusa tal que:*

1. $\mathrm{dom}(\mathcal{P})$ *es un conjunto de* $(\mathrm{seq}, \mathrm{et}_2)$*-cláusulas de programa definido o hechos,*
2. *para cada* $\mathcal{C}_1, \mathcal{C}_2 \in \mathrm{dom}(\mathcal{P})$, $\mathcal{C}_1 \approx \mathcal{C}_2$ *si, y sólo si,* $\mathcal{C}_1$ *es una variante de* $\mathcal{C}_2$ *y* $\mathcal{P}(\mathcal{C}_1) = \mathcal{P}(\mathcal{C}_2)$,

3. $\mathrm{dom}(\mathcal{P})/_{\approx}$ *es finito.*

Intuitivamente, un programa f-Prolog definido puede verse como un conjunto de pares $\langle \mathcal{C}; \alpha \rangle$, donde $\mathcal{C}$ es una cláusula definida y $\alpha = \mathcal{P}(\mathcal{C})$ es un *grado de verdad* que expresa en qué medida, o con qué grado, se satisface la cláusula $\mathcal{C}$, o bien el grado de confianza que el usuario del sistema tiene sobre $\mathcal{C}$.

A menudo, escribiremos "$\mathcal{C}$ with $\mathcal{P}(\mathcal{C})$" en vez de $\langle \mathcal{C}; \mathcal{P}(\mathcal{C}) \rangle$. Un grado de verdad $\alpha = 1$ significa que el usuario contempla la cláusula $\mathcal{C}$ con el más alto grado de verdad, expresando así la máxima confianza en dicha cláusula; por otra parte, un grado de verdad menor que 1 revela el grado de incertidumbre o de falta de confianza sobre $\mathcal{C}$; un grado de verdad próximo a 0 expresa el máximo grado de incertidumbre de la cláusula.

En [Vojtáš y Paulík, 1996], se enuncia la semántica declarativa de un programa f-Prolog en términos del *modelo mínimo de Herbrand difuso*. Se contempla igualmente su caracterización por la semántica de punto fijo.

En ocasiones, la función de verdad de las conectivas et_1 and et_2 se toma del modo:

$$\dot{\mathrm{et}}_1(r_1, \ldots, r_n) = \prod_{i=1}^{n} r_i$$

$$\dot{\mathrm{et}}_2(r_1, \ldots, r_n) = min\{r_1, \ldots, r_n\}$$

Sin embargo, preferimos no especificarlas y considerarlas como t-normas arbitrarias,

$$\dot{\mathrm{et}}_i : [0,1]^2 \to [0,1],$$

es decir, tales que satisfacen las propiedades conmutativa, asociativa, monotonía en ambos argumentos y $\dot{\mathrm{et}}_i(x, 1) = x$.

En consecuencia, generalizan el caso de la conjunción clásica $\wedge : \{0,1\}^2 \to \{0,1\}$ y no cumplen, necesariamente, la propiedad distributiva. Además, han de ser debidamente extendidas para que se puedan contemplar como funciones de n argumentos.

Semántica operacional

La semántica operacional de f-Prolog, debida a Vojtáš y Paulík [1996], especifica cómo obtener los enlaces de las variables y el grado de verdad de un objetivo $\mathcal{G}$ a través de una secuencia de pasos de resolución-SLD difusa. En lo que sigue, se presenta la formalización de los conceptos de resolución-SLD difusa, derivación-SLD difusa y respuesta computada difusa.

Si $\mathcal{P}$ es un programa y $\mathcal{G}$ un objetivo, puesto que dom($\mathcal{P}$) es un programa definido clásico, sobre él puede ejecutarse la resolución-SLD clásica. En consecuencia, el principal problema operacional consiste en describir cómo se lleva a cabo la evaluación de los grados de verdad.

Dada una regla de programa $\mathcal{C} : A \leftarrow B_1, \ldots, B_m$, con grado de verdad q, y un objetivo $\mathcal{G} \equiv \leftarrow A'$, tal que A' unifica con la cabeza A de $\mathcal{C}$ mediante un mgu θ, es posible dar un paso de resolución-SLD que conduce al resolvente $\mathcal{G}' \equiv \leftarrow (B_1, \ldots, B_m)\theta$.

Si deseamos evaluar el grado de verdad de $\mathcal{G}$, necesitamos computar los grados $r_1, \ldots, r_n$ de todos los subobjetivos $B_1, \ldots, B_m$ antes de que el grado q de la regla pueda ser aplicado, para obtener $\dot{\mathrm{e}}\mathrm{t}_1(q, \dot{\mathrm{e}}\mathrm{t}_2(r_1, \ldots, r_n))$, el grado del objetivo $\mathcal{G}$.

En consecuencia, necesitamos un mecanismo para recordar que una regla de programa ha sido aplicada en un paso anterior. En [Vojtáš y Paulík, 1996] se introduce un contexto gramatical para resolver este problema.

Esta gramática libre de contexto contempla marcas izquierda, $\boxed{\mathrm{L}_q}$, y derecha, $\boxed{\mathrm{R}_q}$, etiquetadas con números reales, para recordar el punto exacto en el que se ha aplicado una regla con grado q. Por tanto, el paso de resolución anterior puede ser anotado del modo:

$$\boxed{\mathrm{L}_q}\ B_1, \ldots, B_m\ \boxed{\mathrm{R}_q}$$

Llamaremos lf-Prolog al lenguaje extendido que se obtiene al añadir marcas etiquetadas y números reales al alfabeto f-Prolog. Una lf-expresión es un átomo, una secuencia de átomos, una secuencia de números reales, un número real encerrado entre marcas etiquetadas, y las combinaciones de estas lf-expresiones encerradas entre marcas etiquetadas.

La siguiente definición hace uso del lenguaje extendido lf-Prolog en la formalización del concepto de resolución-SLD difusa (escribimos $\overline{o}$ para la secuencia –tal vez vacía– de objetos sintácticos $o_1, \ldots, o_n$).

Definición 2.2.3. *Dado un objetivo $\mathcal{G} \equiv \leftarrow \mathcal{Q}$ y una sustitución ϑ, un* estado lf-Prolog *es un par $\langle \mathcal{Q}; \vartheta \rangle$. Denotaremos por $\mathcal{E}$ el conjunto de estados.*

Definición 2.2.4 ([Vojtáš y Paulík, 1996])**.** *Dado un programa* f-Prolog*, $\mathcal{P}$, definimos la* resolución-SLD difusa *como un sistema de transición de estados, cuya relación de transición $\rightarrow_{FR} \subset (\mathcal{E} \times \mathcal{E})$ es la relación más pequeña que satisface las siguientes reglas:*

Regla 1. **(Regla de Resolución Cláusula)**

$\langle(\overline{X}, A_m, \overline{Y}); \vartheta\rangle \rightarrow_{FR} \langle(\overline{X}, \boxed{\mathrm{L}_q} B_1, \ldots, B_l \boxed{\mathrm{R}_q}, \overline{Y})\theta; \vartheta\theta\rangle$ *si*

1. A_m *es el átomo seleccionado,*
2. θ *es un mgu de* A_m *y* A*,*
3. $\mathcal{P}(A \leftarrow B_1, \ldots, B_l) = q$ *y* $l \geq 1$*.*

Regla 2. **(Regla de Resolución Hecho)**

$\langle(\overline{X}, A_m, \overline{Y}); \vartheta\rangle \rightarrow_{FR} \langle(\overline{X}, r, \overline{Y})\theta; \vartheta\theta\rangle$ *si*

1. A_m *es el átomo seleccionado,*
2. θ *es un mgu de* A_m *y* A*, y*
3. $\mathcal{P}(A \leftarrow) = r$*.*

Regla 3. **(Regla de Resolución** $\dot{\mathrm{et}}_1$**)**

$\langle(\overline{X} \boxed{\mathrm{L}_q}\ r \boxed{\mathrm{R}_q} \overline{Y}); \vartheta\rangle \rightarrow_{FR} \langle\overline{X}, \dot{\mathrm{et}}_1(q, r), \overline{Y}; \vartheta\rangle$ *si*

1. r *es un número real.*

Regla 4. **(Regla de Resolución** $\dot{\mathrm{et}}_2$**)**

$\langle\overline{X}, r_1, \ldots, r_n, \overline{Y}; \vartheta\rangle \rightarrow_{FR} \langle\overline{X}, \dot{\mathrm{et}}_2(r_1, \ldots, r_n), \overline{Y}; \vartheta\rangle$ *si*

1. $r_1, \ldots, r_n$ *son números reales.*

De forma natural, todos los conceptos habituales de la programación lógica (paso de resolución-SLD, derivación-SLD, etc.) pueden ser extendidos para el caso difuso y, análogamente, se supone que las cláusulas involucradas en pasos de resolución-SLD difusa se renombran antes de ser usadas. Denotaremos con $\rightarrow_{FR1}, \rightarrow_{FR2}, \rightarrow_{FR3}$, $\rightarrow_{FR4}$ al paso de resolución-SLD difusa dado con la Regla 1, Regla 2, Regla 3 o Regla 4, respectivamente.

Cuando sea necesario, anotaremos, mediante un superíndice del símbolo $\rightarrow_{FR}$, la ef-expresión y/o la cláusula usada en el correspondiente paso de resolución.

En la siguiente definición se extiende la noción de respuesta computada a nuestro entorno difuso. En ella, denotamos por id la sustitución vacía, llamamos $\mathcal{V}ar(s)$ al conjunto de variables distintas que aparecen en el objeto sintáctico s, y $\theta[\mathcal{V}ar(s)]$ denota la sustitución obtenida de θ restringiendo su dominio, $Dom(\theta)$, a $\mathcal{V}ar(s)$.

Definición 2.2.5. *Sea $\mathcal{P}$ un programa* f-Prolog *y $\mathcal{G} \equiv \leftarrow \mathcal{Q}$ un objetivo. Un par $\langle r; \theta \rangle$ formado por un número real r y una sustitución θ es una* respuesta computada difusa *si hay una secuencia $\mathcal{E}_0, \ldots, \mathcal{E}_n$ (llamada* f-derivación de éxito*) tal que:*

1. $\mathcal{E}_0 = \langle \mathcal{Q}; id \rangle$;

2. *para cada $0 \leq i < n$, $\mathcal{G}_i \rightarrow_{FR} \mathcal{G}_{i+1}$ es un paso de resolución-SLD difusa;*

3. $\mathcal{E}_n = \langle r; \theta' \rangle$ *y* $\theta = \theta'[\mathcal{V}ar(\mathcal{Q})]$.

Ilustramos esta definición con el siguiente ejemplo. En el programa elegido, los predicados considerados podrían tener el siguiente significado:

$p(x)$: x es país industrializado $\quad$ $q(x, y)$: x es más desarrollado que y

$r(y)$: y es tecnológicamente avanzado $\quad$ $s(y)$: y es balcánico

Esta interpretación es coherente en el programa y si tomamos a =españa, b =rumanía, entonces podemos deducir:

"españa es país industrializado con grado de verdad 0.504".

Ejemplo 2.2.6. *Sea $\mathcal{P}$ el programa* f-Prolog*,*

$$\begin{array}{lll} \mathcal{C}_1: & p(X) \leftarrow q(X,Y), r(Y) & \textit{with } 0.8 \\ \mathcal{C}_2: & q(a,Y) \leftarrow s(Y) & \textit{with } 0.7 \\ \mathcal{C}_3: & q(Y,a) \leftarrow r(Y) & \textit{with } 0.8 \\ \mathcal{C}_4: & r(Y) \leftarrow & \textit{with } 0.7 \\ \mathcal{C}_5: & s(b) \leftarrow & \textit{with } 0.9 \end{array}$$

Por comodidad y analogía con Prolog, suponemos que existe una función de selección que elige de izquierda a derecha los átomos en un objetivo dado. Además, tomamos

$$\dot{\text{et}}_1(x, y) = x \cdot y \qquad \dot{\text{et}}_2(x, y) = \text{mín}\{x, y\}$$

En estas hipótesis, tenemos la siguiente f-derivación de éxito para el programa $\mathcal{P}$ y el objetivo $\leftarrow p(X)$ (subrayamos la lf-expresión seleccionada en cada paso de resolución) con respuesta computada difusa $\langle 0.504; \{X/a\} \rangle$:

$$\langle \underline{p(X)}, id\rangle \qquad \rightarrow_{FR1}{}^{\mathcal{C}_1}$$

$$\langle \boxed{\mathrm{L}_{0.8}}\ \underline{q(X_1,Y_1)}, r(Y_1)\ \boxed{\mathrm{R}_{0.8}}\ ;\{X/X_1\}\rangle \qquad \rightarrow_{FR1}{}^{\mathcal{C}_2}$$

$$\langle \boxed{\mathrm{L}_{0.8}}\ \boxed{\mathrm{L}_{0.7}}\ \underline{s(Y_2)}\ \boxed{\mathrm{R}_{0.7}}, r(Y_2)\ \boxed{\mathrm{R}_{0.8}}\ ;\{X/a, X_1/a, Y_1/Y_2\}\rangle \qquad \rightarrow_{FR2}{}^{\mathcal{C}_5}$$

$$\langle \boxed{\mathrm{L}_{0.8}}\ \boxed{\mathrm{L}_{0.7}}\ 0.9\ \boxed{\mathrm{R}_{0.7}}, \underline{r(b)}\ \boxed{\mathrm{R}_{0.8}}\ ;\{X/a, X_1/a, Y_1/b, Y_2/b\}\rangle \qquad \rightarrow_{FR2}{}^{\mathcal{C}_4}$$

$$\langle \boxed{\mathrm{L}_{0.8}}\ \underline{\boxed{\mathrm{L}_{0.7}}\ 0.9\ \boxed{\mathrm{R}_{0.7}}}, 0.7\ \boxed{\mathrm{R}_{0.8}}\ ;\{X/a, X_1/a, Y_1/b, Y_2/b, Y_3/b\}\rangle \qquad \rightarrow_{FR3}$$

$$\langle \boxed{\mathrm{L}_{0.8}}\ \underline{0.63, 0.7}\ \boxed{\mathrm{R}_{0.8}}\ ;\{X/a, X_1/a, Y_1/b, Y_2/b, Y_3/b\}\rangle \qquad \rightarrow_{FR4}$$

$$\langle \underline{\boxed{\mathrm{L}_{0.8}}\ 0.63\ \boxed{\mathrm{R}_{0.8}}}\ ;\{X/a, X_1/a, Y_1/b, Y_2/b, Y_3/b\}\rangle \qquad \rightarrow_{FR3}$$

$$\langle 0.504\ ;\{X/a, X_1/a, Y_1/b, Y_2/b, Y_3/b\}\rangle$$

En [Vojtáš y Paulík, 1996], los autores establecen los resultados de corrección para el mecanismo operacional de la resolución-SLD difusa (siguiendo una técnica similar a la propuesta por Lloyd [1987], para la programación lógica), pero extendiendo los resultados con el tratamiento de los grados de verdad.

Como ya dijimos, si permitimos el uso de marcas y números reales, no sólo en los estados intermedios de una derivación, sino también en los cuerpos de las cláusulas, tenemos un lenguaje etiquetado que llamamos lf-Prolog, para el que sigue siendo válida la semántica operacional anterior.

En consecuencia, f-Prolog y lf-Prolog, lenguaje que tendrá una fuerte repercusión en la definición de desplegado difuso, comparten el mismo principio operacional: la resolución-SLD difusa.

2.3. El lenguaje extendido ef-Prolog

Presentamos ahora una extensión del lenguaje f-Prolog (Prolog *fuzzy*), descrita por primera vez en [Julián *et al.*, 2004a], a fin de permitir una interpretación más flexible de las conectivas lógicas conforme al estilo considerado en [Vaucheret *et al.*, 2002].

Hemos dicho que, en el lenguaje f-Prolog, las conectivas et_1, et_2 son t-normas arbitrarias cuya función de verdad

$$\dot{\text{et}}_i : [0,1]^2 \to [0,1],$$

adecuadamente extendida a funciones de n argumentos, es fija en cada programa. Sin embargo, y tal como se justifica en [Vaucheret *et al.*, 2002], puede ser útil desde un punto de vista práctico asociar una interpretación concreta para cada operador et_1 o et_2 en cada cláusula de programa en vez de una única interpretación (para cada uno de las conectivas) en un programa determinado.

A tal efecto, observemos que la conjunción ha sido definida inicialmente por Zadeh a través del operador *min*, pero en la actualidad está ampliamente aceptado que no es el único que modela adecuadamente la conjunción, por la amplia variedad de expresiones que es necesario formalizar. Es lo que ilustramos en el siguiente ejemplo.

Ejemplo 2.3.1. *Dada la cláusula* $p(x) \leftarrow q(x), r(x)$ *si interpretamos*

$p(x)$ *como "x es deportista",*
$q(x)$ *como "x es joven",*
$r(x)$ *como "x es saludable",*

los predicados del cuerpo son de influencia positiva, es decir, se refuerzan mutuamente. En este caso a la conjunción et_2 *del cuerpo se le asocia habitualmente por el operador min como función de verdad. Lo mismo debemos considerar si* $q(x)$ *y* $r(x)$ *fuesen predicados independientes.*

Sin embargo, si $q(x)$ *se interpreta como "x es bajo" y* $r(x)$ *como "x es alto", es decir, si los predicados* $q(x)$ *y* $r(x)$ *son contradictorios, al conectivo* et_2 *del cuerpo de la cláusula es aconsejable asociarle el operador max como función de verdad (véase Aranda* et al. *[1993]).*

En general, la elección de la función de verdad de las conectivas depende del contexto, es decir, del problema real que se pretende modelar [Gupta y Qi, 1991].
Redefinimos el concepto de teoría difusa, ya visto para f-Prolog, para afrontar el problema de cómo se asocian distintas interpretaciones a las conectivas et_1 y et_2.

Definición 2.3.2. *Una* teoría difusa *es una aplicación $\mathcal{T}$ que asocia a cada fórmula un triple $\langle \alpha, le_1, le_2 \rangle \in (0,1] \times Sem \times Sem$, donde el* grado de verdad *$\alpha \in (0,1]$ y Sem es un conjunto de etiquetas semánticas indicando el significado asociado para et_1 y et_2 respectivamente.*

Entendemos que $\mathcal{T}$ es idénticamente nula sobre todas las fórmulas que no están en el $\mathrm{dom}(\mathcal{T})$.

Ahora podemos usar etiquetas indicando el significado asignado a los operadores et_1 o et_2 en la cláusula $\mathcal{C}$.

Por ejemplo, una etiqueta $le_i =$ `lukasiewicz` interpreta el operador et_i como una t-norma de Lukasiewicz, es decir, $\dot{\mathrm{et}}_i(x,y) = max\{0, x+y-1\}$. Otras posibles etiquetas podrían ser: `min`, si $\dot{\mathrm{et}}_i(x,y) = min\{x,y\}$; `product`, si $\dot{\mathrm{et}}_i(x,y) = x \cdot y$; etc.

Un valor `void` en Sem se emplea para expresar que no hay significado seleccionado para et_1 o et_2. Puesto que `void` debe estar asociado a una t-norma, se verificará $void(1,x) = void(x,1) = x$.

Definición 2.3.3. *Un* programa definido ef-Prolog, *$\mathcal{P}$, es una teoría difusa tal que:*

1. $\mathrm{dom}(\mathcal{P})$ *es un conjunto de* $(\mathrm{seq}, \mathrm{et}_2)$*-cláusulas de programa definido o hechos,*
2. *para $\mathcal{C}_1, \mathcal{C}_2 \in \mathrm{dom}(\mathcal{P})$, se tiene $\mathcal{C}_1 \approx \mathcal{C}_2$ si, y sólo si, $\mathcal{C}_1$ es una variante de $\mathcal{C}_2$ y $\mathcal{P}(\mathcal{C}_1) = \mathcal{P}(\mathcal{C}_2)$, y*
3. $\mathrm{dom}(\mathcal{P})/_{\approx}$ *es finito.*

Intuitivamente, un *programa definido* ef-Prolog puede verse como un conjunto de pares $\langle \mathcal{C}; \langle r, l1, l2 \rangle \rangle$, donde $\mathcal{C}$ es una cláusula, r es el grado de verdad de $\mathcal{C}$, y $l1$, $l2$ son etiquetas semánticas asociadas con los operadores $\mathrm{et}_1, \mathrm{et}_2$ de esta cláusula.

Sin embargo, nos inclinamos por escribir $\mathcal{C}$ `with` $\langle r, l1, l2 \rangle$, o de modo más descriptivo:

$$\mathcal{C} \texttt{ with } r \texttt{ and } le_1 = l1 \texttt{ and } le_2 = l2.$$

Si la cláusula $\mathcal{C}$ es un hecho, le_1 y le_2 pueden tomarse `void` y escribiremos solamente: $\mathcal{C}$ `with` r, omitiendo los valores de le_1 y le_2, y si el cuerpo sólo tiene un átomo asumimos $le_2 =$ `void`.

Análogamente, un objetivo $\mathcal{G}$ sólo tiene asociado una etiqueta semántica para et_2, pero no tiene grado de verdad inicial ni etiqueta semántica para et_1, así que escribiremos $\mathcal{G}$ `with` $le_2 = l2$.

Semántica operacional

Una vez caracterizada la extensión del lenguaje f-Prolog que hemos denominado ef-Prolog, debemos formalizar, de nuevo, los conceptos de resolución-SLD difusa, derivación-SLD difusa y de respuesta computada difusa para este lenguaje extendido.

Al objeto de tratar adecuadamente el proceso de resolución para este lenguaje que mejora la potencia expresiva de f-Prolog, es necesario también ampliar el formato de representación de las etiquetas para distinguir cuál el la función de verdad de $\dot{\text{et}}_1$ o $\dot{\text{et}}_2$ que debe ser aplicada.

A tal fin, introducimos las marcas que siguen, en las que se recoge el grado de verdad de una cláusula junto con la etiqueta que recuerda la interpretación de las correspondientes conectivas.

$$\boxed{\text{L}_{\langle q,\dot{\text{et}}_1,\dot{\text{et}}_2\rangle}} \qquad \boxed{\text{R}_{\langle q,\dot{\text{et}}_1,\dot{\text{et}}_2\rangle}}$$

Llamamos lenguaje lef-Prolog al obtenido añadiendo estas marcas etiquetadas y números reales al alfabeto f-Prolog. Ahora, una lef-expresión es un átomo, una secuencia de átomos, una secuencia de números reales, un número real encerrado entre las nuevas marcas etiquetadas que se consideran, y las combinaciones de las anteriores lef-expresiones también encerradas entras las nuevas marcas etiquetadas.

De forma similar a la sección anterior donde llamábamos lf-Prolog al lenguaje etiquetado obtenido de f-Prolog, ahora llamamos lef-Prolog a la extensión equivalente de ef-Prolog, que posee idéntica semántica operacional que ef- Prolog y tiene gran relevancia a la hora de definir transformaciones basadas en desplegado.

La siguiente definición hace uso de lef-Prolog en la formalización de la resolución-SLD difusa (recordemos que escribimos $\overline{o}$ para la secuencia –tal vez vacía– de objetos sintácticos $o_1, \ldots, o_n$).

Definición 2.3.4. *Dado un objetivo* $\mathcal{G} \equiv \leftarrow \mathcal{Q}$ *with* $le_2 = \dot{\text{et}}_2$ lef-Prolog *y* ϑ *una sustitución, un* estado ef-Prolog *es un par* $\langle \mathcal{Q}; \vartheta \rangle$. *Llamaremos,* $\mathcal{E}$ *el conjunto de estados* ef-Prolog.

Definición 2.3.5. *Dado un programa* ef-Prolog $\mathcal{P}$, *definimos la* resolución-SLD difusa *como un sistema de transición de estados, cuya relación de transición* $\rightarrow_{FR} \subset (\mathcal{E} \times \mathcal{E})$ *es la mínima relación que satisface las siguientes reglas:*

Regla 1. **(Regla de Resolución Cláusula)**

$\langle \overline{X}, A_m, \overline{Y}; \vartheta \rangle \rightarrow_{FR} \langle (\overline{X}, \boxed{\mathrm{L}_{\langle q, \dot{\mathrm{e}}\mathrm{t}_1, \dot{\mathrm{e}}\mathrm{t}_2 \rangle}} B_1, \ldots, B_l \boxed{\mathrm{R}_{\langle q, \dot{\mathrm{e}}\mathrm{t}_1, \dot{\mathrm{e}}\mathrm{t}_2 \rangle}}, \overline{Y})\theta; \vartheta\theta \rangle$ *si*

1. A_m *es el átomo seleccionado,*
2. θ *es un mgu de* A_m *y* A*,*
3. $\mathcal{C} : (A \leftarrow B_1, \ldots, B_l \; with \; \langle q, \dot{\mathrm{e}}\mathrm{t}_1, \dot{\mathrm{e}}\mathrm{t}_2 \rangle) \in \mathcal{P}$ *y* $l \geq 1$*.*

Regla 2. **(Regla de Resolución Hecho)**

$\langle \overline{X}, A_m, \overline{Y}; \vartheta \rangle \rightarrow_{FR} \langle (\overline{X}, r, \overline{Y})\theta; \vartheta\theta \rangle$ *si*

1. A_m *es el átomo seleccionado,*
2. θ *es un mgu de* A_m *y* A*, y*
3. $\mathcal{C} : (A \leftarrow \; con \; r) \in \mathcal{P}$*.*

Regla 3. **(Regla de Resolución** $\dot{\mathrm{e}}\mathrm{t}_1$**)**

$\langle \overline{X}, \boxed{\mathrm{L}_{\langle q, \dot{\mathrm{e}}\mathrm{t}_1, \dot{\mathrm{e}}\mathrm{t}_2 \rangle}} \; r \boxed{\mathrm{R}_{\langle q, \dot{\mathrm{e}}\mathrm{t}_1, \dot{\mathrm{e}}\mathrm{t}_2 \rangle}}, \overline{Y}; \vartheta \rangle \rightarrow_{FR} \langle \overline{X}, \dot{\mathrm{e}}\mathrm{t}_1(q, r), \overline{Y}; \vartheta \rangle$ *si*

1. r *es un número real.*

Regla 4. **(Regla de Resolución** $\dot{\mathrm{e}}\mathrm{t}_2$**)**

$\langle \overline{X}, \boxed{\mathrm{L}_{\langle q, \dot{\mathrm{e}}\mathrm{t}_1, \dot{\mathrm{e}}\mathrm{t}_2 \rangle}} \; \overline{r} \boxed{\mathrm{R}_{\langle q, \dot{\mathrm{e}}\mathrm{t}_1, \dot{\mathrm{e}}\mathrm{t}_2 \rangle}}, \overline{Y}; \vartheta \rangle \rightarrow_{FR}$

$\langle \overline{X}, \boxed{\mathrm{L}_{\langle q, \dot{\mathrm{e}}\mathrm{t}_1, \dot{\mathrm{e}}\mathrm{t}_2 \rangle}} \; \dot{\mathrm{e}}\mathrm{t}_2(\overline{r}) \boxed{\mathrm{R}_{\langle q, \dot{\mathrm{e}}\mathrm{t}_1, \dot{\mathrm{e}}\mathrm{t}_2 \rangle}}, \; \overline{Y}; \vartheta \rangle$ *si*

1. $\overline{r} \equiv r_1, \ldots, r_n$*, donde* $n > 1$*, son números reales.*

Los conceptos habituales de programación lógica (paso de resolución-SLD, derivación-SLD, etc.) pueden definirse también aquí de manera natural.

En lo que sigue, usaremos los símbolos $\rightarrow_{FR1}, \rightarrow_{FR2}, \rightarrow_{FR3}$ y $\rightarrow_{FR4}$ para denotar la aplicación de cada una de las cuatro reglas de resolución difusa anteriores.

Como ya se ha dicho, esta semántica operacional descrita para ef-Prolog sigue siendo válida para el lenguaje etiquetado lef-Prolog.

En la siguiente definición extendemos la noción de respuesta computada difusa a nuestro entorno.

Definición 2.3.6. *Sea $\mathcal{P}$ un programa* ef-Prolog *y $\mathcal{G} \equiv \leftarrow \mathcal{Q}$ with $le_2 = \dot{\text{et}}_2$ un objetivo. Un par $\langle r; \theta \rangle$ formado por un número real r y una sustitución θ es una* respuesta computada difusa *si hay una secuencia $\mathcal{E}_0, \ldots, \mathcal{E}_n$ (llamada* f-derivación de éxito*) tal que:*

1. $\mathcal{E}_0 = \langle \boxed{\text{L}_{\langle 1, void, \dot{\text{et}}_2 \rangle}}, \mathcal{Q}, \boxed{\text{R}_{\langle 1, void, \dot{\text{et}}_2 \rangle}}; id \rangle$,

2. *para cada $0 \leq i < n$, $\mathcal{G}_i \rightarrow_{FR} \mathcal{G}_{i+1}$ es un paso de resolución-SLD difusa (según se ha establecido en la Definición 2.3.5),*

3. $\mathcal{E}_n = \langle r; \theta' \rangle$ *y* $\theta = \theta'[\mathcal{V}ar(\mathcal{Q})]$.

Ilustramos esta definición en el ejemplo posterior, que puede verse como una adaptación del Ejemplo 2.2.6 al nuevo marco más rico.

Ejemplo 2.3.7. *Sea $\mathcal{P}$ un programa* ef-Prolog,

$$
\begin{array}{lll}
\mathcal{C}_1: & p(X) \leftarrow q(X,Y), r(Y) & \textit{with } \langle 0.8, \texttt{prod}, \texttt{min} \rangle \\
\mathcal{C}_2: & q(a,Y) \leftarrow s(Y) & \textit{with } \langle 0.7, \texttt{prod}, \texttt{void} \rangle \\
\mathcal{C}_3: & q(Y,a) \leftarrow r(Y) & \textit{with } \langle 0.8, \texttt{luka}, \texttt{void} \rangle \\
\mathcal{C}_4: & r(Y) \leftarrow & \textit{with } 0.7 \\
\mathcal{C}_5: & s(b) \leftarrow & \textit{with } 0.9
\end{array}
$$

La que sigue es una f-derivación de éxito para el programa $\mathcal{P}$ y el objetivo

$$\leftarrow p(X), r(a) \textit{ with } \texttt{min}$$

con respuesta computada $\langle 0.504; \{X/a\} \rangle$:

$\langle \boxed{\mathrm{L}_{\Psi_0}}\ \underline{p(X)}, r(a)\ \boxed{\mathrm{R}_{\Psi_0}}\ ; \sigma_0 \rangle \quad \rightarrow_{FR1}{}^{\mathcal{C}_1}$

$\langle \boxed{\mathrm{L}_{\Psi_0}}\ \boxed{\mathrm{L}_{\Psi_1}}\ \underline{q(X_1, Y_1)}, r(Y_1)\ \boxed{\mathrm{R}_{\Psi_1}}\ r(a)\ \boxed{\mathrm{R}_{\Psi_0}}\ ; \sigma_1 \rangle \quad \rightarrow_{FR1}{}^{\mathcal{C}_2}$

$\langle \boxed{\mathrm{L}_{\Psi_0}}\ \boxed{\mathrm{L}_{\Psi_1}}\ \boxed{\mathrm{L}_{\Psi_2}}\ \underline{s(Y_2)}\ \boxed{\mathrm{R}_{\Psi_2}}\ r(Y_2)\ \boxed{\mathrm{R}_{\Psi_1}}\ r(a)\ \boxed{\mathrm{R}_{\Psi_0}}\ ; \sigma_2 \rangle \quad \rightarrow_{FR2}{}^{\mathcal{C}_5}$

$\langle \boxed{\mathrm{L}_{\Psi_0}}\ \boxed{\mathrm{L}_{\Psi_1}}\ \boxed{\mathrm{L}_{\Psi_2}}\ 0.9\ \boxed{\mathrm{R}_{\Psi_2}}\ \underline{r(b)}\ \boxed{\mathrm{R}_{\Psi_1}}\ r(a)\ \boxed{\mathrm{R}_{\Psi_0}}\ ; \sigma_3 \rangle \quad \rightarrow_{FR2}{}^{\mathcal{C}_4}$

$\langle \boxed{\mathrm{L}_{\Psi_0}}\ \boxed{\mathrm{L}_{\Psi_1}}\ \boxed{\mathrm{L}_{\Psi_2}}\ 0.9\ \boxed{\mathrm{R}_{\Psi_2}}\ 0.7\ \boxed{\mathrm{R}_{\Psi_1}}\ \underline{r(a)}\ \boxed{\mathrm{R}_{\Psi_0}}\ ; \sigma_4 \rangle \quad \rightarrow_{FR2}{}^{\mathcal{C}_4}$

$\langle \boxed{\mathrm{L}_{\Psi_0}}\ \boxed{\mathrm{L}_{\Psi_1}}\ \underline{\boxed{\mathrm{L}_{\Psi_2}}\ 0.9\ \boxed{\mathrm{R}_{\Psi_2}}}\ 0.7\ \boxed{\mathrm{R}_{\Psi_1}}\ 0.7\ \boxed{\mathrm{R}_{\Psi_0}}\ ; \sigma_5 \rangle \quad \rightarrow_{FR3}$

```
//dado que product(0.9,0.7)=0.63
```

$\langle \boxed{\mathrm{L}_{\Psi_0}}\ \underline{\boxed{\mathrm{L}_{\Psi_1}}\ 0.63, 0.7\ \boxed{\mathrm{R}_{\Psi_1}}}\ 0.7\ \boxed{\mathrm{R}_{\Psi_0}}\ ; \sigma_5 \rangle \quad \rightarrow_{FR4}$

```
//dado que min(0.63,0.7)=0.63
```

$\langle \boxed{\mathrm{L}_{\Psi_0}}\ \underline{\boxed{\mathrm{L}_{\Psi_1}}\ 0.63\ \boxed{\mathrm{R}_{\Psi_1}}}\ 0.7\ \boxed{\mathrm{R}_{\Psi_0}}\ ; \sigma_5 \rangle \quad \rightarrow_{FR3}$

```
//dado que product(0.63,0.8)=0.504
```

$\langle \underline{\boxed{\mathrm{L}_{\Psi_0}}\ 0.504, 0.7\ \boxed{\mathrm{R}_{\Psi_0}}}\ ; \sigma_5 \rangle \quad \rightarrow_{FR4}$

```
//dado que min(0.504,0.7)=0.504
```

$\langle \underline{\boxed{\mathrm{L}_{\Psi_0}}\ 0.504\ \boxed{\mathrm{R}_{\Psi_0}}}\ ; \sigma_5 \rangle \quad \rightarrow_{FR3}$

```
// dado que toda t-norm ėt1(1,x) = x, entonces void(1,0.504)=0.504
```

$\langle\ 0.504\ ; \sigma_5 \rangle$

donde

$$\Psi_0 \equiv \langle 1, \texttt{void}, \texttt{min} \rangle,$$

$$\Psi_1 \equiv \langle 0.8, \texttt{prod}, \texttt{min} \rangle,$$

$$\Psi_2 \equiv \langle 0.7, \texttt{prod}, \texttt{void} \rangle,$$

son los triples asociados al objetivo original y las cláusulas $\mathcal{C}_1$ *y* $\mathcal{C}_2$ *respectivamente. Además, resultan las sustituciones*

$$\sigma_0 = id,$$

$$\sigma_1 = \{X/X_1\},$$

$$\sigma_2 = \{X/a, X_1/a,\ Y_1/Y_2\},$$

$$\sigma_3 = \{X/a, X_1/a, Y_1/b, Y_2/b\},$$

$$\sigma_4 = \{X/a, X_1/a,\ Y_1/b, Y_2/b, Y_3/b\},$$
$$\sigma_5 = \{X/a, X_1/a,\ Y_1/b, Y_2/b, Y_3/b, Y_4/a\}.$$

2.4. Independencia de la regla de computación

Al igual que en el cálculo de la resolución-SLD clásica, suponemos la existencia de una función de selección fija, también llamada regla de computación, que decide, sobre un objetivo dado, cuál es la expresión seleccionada para explotar en el siguiente paso de computación.

En particular, cuando construimos la f-derivación mostrada en el Ejemplo 2.2.6, hemos usado una regla de computación similar a la de Prolog, seleccionando los átomos de izquierda a derecha, pero retrasando la aplicación de las reglas de resolución $\dot{\text{et}}_3$ y $\dot{\text{et}}_4$ hasta que se han explotado todos los átomos.

Antes de acometer los resultados generales con los que extendemos a un contexto borroso el teorema de independencia de la regla de computación clásico, recogido en [Lloyd, 1987], introducimos previamente la notación, los conceptos y resultados instrumentales necesarios.

En lo sucesivo, usaremos $\mathcal{C} \ll \mathcal{P}$ para denotar una versión renombrada o *estandarizada aparte* o nueva variante[1] de una cláusula del programa $\mathcal{P}$ tal que $\mathcal{C}$ no contiene variables que hayan sido usadas previamente en la computación.

La representación equacional de una sustitución $\theta = \{x_1/t_1, \ldots, x_n/t_n\}$ es el conjunto de equaciones $\widehat{\theta} = \{x_1 = t_1, \ldots, x_n = t_n\}$. Denotamos por $mgu(E)$ el *unificador más general*[2] de un conjunto de ecuaciones E.

La *composición paralela* de sustituciones [Palamidessi, 1990] corresponde a la noción de unificación generalizada de sustituciones. Dadas las sustituciones idempotentes θ_1 y θ_2, la composición paralela $\theta_1 \Uparrow \theta_2 = mgu(\widehat{\theta_1} \cup \widehat{\theta_2})$[3]. Haremos uso de la siguiente propiedad más adelante.

Lema 2.4.1 ([Palamidessi, 1990]). *Las sustituciones idempotentes θ_1 y θ_2 satisfacen,*

$$\theta_1 \Uparrow \theta_2 = \theta_1 mgu(\widehat{\theta_2}\theta_1) = \theta_2 mgu(\widehat{\theta_1}\theta_2).$$

[1] Formalmente, decimos que dos expresiones E_1 y E_2 son variantes una de la otra si, y sólo si, existen sustituciones de renombramiento ρ y ρ' tal que $E_1 = E_2\rho$ y $E_2 = E_1\rho'$.

[2] Véase la Sección 1.4 para una definición formal de este concepto.

[3] Véase de nuevo la Sección 1.4 para una definición del conjunto unión de sustituciones.

2.4.1. Independencia de la regla de computación para lf-Prolog

El siguiente teorema, enunciado por primera vez en [Julián *et al.*, 2004b] y demostrado en [Julián *et al.*, 2004c], extiende al entorno difuso la independencia de la regla de computación demostrado en [Lloyd, 1987] para la programación lógica.

Teorema 2.4.2 (Independencia de la Regla de Computación)**.** *Sea $\mathcal{P}$ un programa* lf-Prolog *y $\mathcal{G} \equiv \leftarrow\mathcal{Q}$ un objetivo* lf-Prolog*. Si*

$$\langle \mathcal{G}; id\rangle \rightarrow_{FR}{}^{n}_{\mathcal{R}} \langle r; \theta\rangle$$

es una derivación-SLD difusa de n pasos que usa $\mathcal{R}$ como regla de computación, entonces

$$\langle \mathcal{G}; id\rangle \rightarrow_{FR}{}^{n}_{\mathcal{R}'} \langle r; \theta\rangle$$

es una derivación-SLD difusa de n pasos que usa cualquier otra regla de computación $\mathcal{R}'$.

No incluimos la demostración por ser este resultado un caso particular del enunciado a continuación para el lenguaje extendido.

2.4.2. Independencia de la regla de computación para ef-Prolog

También para el lenguaje extendido ef-Prolog hemos logrado este resultado esencial –enunciado por primera vez en [Julián *et al.*, 2004a]– que generaliza al Teorema 2.4.2 y cuya prueba –recogida en [Julián *et al.*, 2005c]– es más compleja.

Teorema 2.4.3 (Independencia de la Regla de Computación)**.** *Sea $\mathcal{P}$ un programa* lef-Prolog*, y $\mathcal{G} \equiv \leftarrow\mathcal{Q}$ with $le_2 = \dot{\text{et}}_2$ un objetivo* lef-Prolog*. Entonces, para cualquier par de reglas de computación $\mathcal{R}$ y $\mathcal{R}'$, tenemos que:*

$$\langle \boxed{\text{L}_\Psi}\ \mathcal{Q}\ \boxed{\text{R}_\Psi}\ ; id\rangle \rightarrow_{FR}{}^{n}_{\mathcal{R}} \langle r; \theta\rangle \quad \textit{si, y sólo si,} \quad \langle \boxed{\text{L}_\Psi}\ \mathcal{Q}\ \boxed{\text{R}_\Psi}\ ; id\rangle \rightarrow_{FR}{}^{n}_{\mathcal{R}'} \langle r; \theta\rangle$$

donde $\Psi \equiv \langle 1, void, \dot{\text{et}}_2\rangle$ y n es el (mismo) número de pasos de resolución-SLD difusa en ambas derivaciones.

Para demostrar la independencia de la regla de computación para el lenguaje lef-Prolog, necesitamos los lemas auxiliares que siguen. El primero se refiere a la conservación de las sustituciones en las respuestas computadas difusas, obtenidas al realizar dos pasos de derivación-SLD difusa explotando dos átomos diferentes de un objetivo dado, con independencia del orden en que se exploten.

Lema 2.4.4. *Sea $\mathcal{P}$ un programa* lef-Prolog *y sea $\mathcal{G} \equiv \leftarrow \mathcal{Q}_0$ `with le2 =` $\dot{\text{et}}_2^0$ un objetivo* ef-Prolog*, tal que A y A' son átomos de $\mathcal{Q}_0$ y $\Psi_0 \equiv \langle 1, void, \dot{et}_2^0 \rangle$. Entonces,*

$$\langle \boxed{\mathrm{L}_{\Psi_0}} \mathcal{Q}_0 \boxed{\mathrm{R}_{\Psi_0}} \; ; \theta_0 \rangle \rightarrow_{FR}{}^{A} \langle \boxed{\mathrm{L}_{\Psi_0}} \mathcal{Q}_1 \boxed{\mathrm{R}_{\Psi_0}} \; ; \theta_0\theta_1 \rangle \rightarrow_{FR}{}^{A'\theta_1} \langle \boxed{\mathrm{L}_{\Psi_0}} \mathcal{Q}_2 \boxed{\mathrm{R}_{\Psi_0}} \; ; \theta_0\theta_1\theta_2 \rangle$$

si, y sólo si,

$$\langle \boxed{\mathrm{L}_{\Psi_0}} \mathcal{Q}_0 \boxed{\mathrm{R}_{\Psi_0}} \; ; \theta_0 \rangle \rightarrow_{FR}{}^{A'} \langle \boxed{\mathrm{L}_{\Psi_0}} \mathcal{Q}'_1 \boxed{\mathrm{R}_{\Psi_0}} \; ; \theta_0\theta'_1 \rangle \rightarrow_{FR}{}^{A\theta'_1} \langle \boxed{\mathrm{L}_{\Psi_0}} \mathcal{Q}'_2 \boxed{\mathrm{R}_{\Psi_0}} \; ; \theta_0\theta'_1\theta'_2 \rangle$$

donde $\theta_0\theta_1\theta_2 = \theta_0\theta'_1\theta'_2$.

Demostración.
($\Rightarrow$) Supongamos que en la cabeza de las cláusulas $\mathcal{C}_1, \mathcal{C}_2 \ll \mathcal{P}$ tenemos los átomos H_1 y H_2 que usamos para explotar (instancias de) átomos A y A', respectivamente, en la f-derivación considerada:

$$\langle \boxed{\mathrm{L}_{\Psi_0}} \mathcal{Q}_0 \boxed{\mathrm{R}_{\Psi_0}} \; ; \theta_0 \rangle \rightarrow_{FR}{}^{A} \langle \boxed{\mathrm{L}_{\Psi_0}} \mathcal{Q}_1 \boxed{\mathrm{R}_{\Psi_0}} \; ; \theta_0\theta_1 \rangle \rightarrow_{FR}{}^{A'\theta_1} \langle \boxed{\mathrm{L}_{\Psi_0}} \mathcal{Q}_2 \boxed{\mathrm{R}_{\Psi_0}} \; ; \theta_0\theta_1\theta_2 \rangle$$

donde $\theta_1 = mgu(\{A = H_1\})$ y $\theta_2 = mgu(\{A'\theta_1 = H_2\})$.

Además, puesto que $\theta_0\theta_1\theta_2 \neq fallo$, en particular $\theta_2 \neq fallo$, y en consecuencia $\theta'_1 = mgu(\{A' = H_2\}) \neq fallo$.

Entonces, se tienen las siguientes igualdades:

$$\begin{array}{lcl}
\theta_1\theta_2 & = & \\
\theta_1 mgu(\{A'\theta_1 = H_2\}) & = & \text{(puesto que } Dom(\theta_1) \cap \mathcal{V}ar(\mathcal{C}_2) = \varnothing) \\
\theta_1 mgu(\widehat{mgu}(\{A' = H_2\})\theta_1) & = & \\
\theta_1 mgu(\widehat{\theta'_1}\theta_1) & = & \text{(por el Lema 2.4.1)} \\
\theta_1 \Uparrow \theta'_1 & = & \text{(por el Lema 2.4.1)} \\
\theta'_1 mgu(\widehat{\theta_1}\theta'_1) & = & \\
\theta'_1 mgu(\widehat{mgu}(\{A = H_1\})\theta'_1) & = & \text{(puesto que } Dom(\theta'_1) \cap \mathcal{V}ar(\mathcal{C}_1) = \varnothing) \\
\theta'_1 mgu(\{A\theta'_1 = H_1\}) & &
\end{array}$$

Además, como $\theta_1\theta_2 \neq fallo$ entonces $\theta_1' mgu(\{A\theta_1' = H_1\}) \neq fallo$ y, en particular, $\theta_2' = mgu(\{A\theta_1' = H_1\}) \neq fallo$.

Por tanto, $\theta_1\theta_2 = \theta_1'\theta_2'$, de donde se deduce la igualdad $\theta_0\theta_1\theta_2 = \theta_0\theta_1'\theta_2'$, como deseábamos probar.
($\Leftarrow$) Este recíproco puede probarse de manera similar a la implicación directa, explotando también la equivalencia entre $\theta_1\theta_2$ y $\theta_1'\theta_2'$. □

El siguiente lema generaliza el anterior en dos aspectos diferentes:

- por garantizar la preservación de los dos elementos significativos del estado, esto es, no sólo la sustitución, sino también, los grados de verdad, y
- por considerar todo el conjunto de reglas la Definición 2.3.5 (en vez de solamente las dos primeras) cuando aplicamos los dos pasos de resolución en la derivación considerada.

Lema 2.4.5 (Lema de Conmutación). *Sea $\mathcal{P}$ un programa* lef-Prolog *y sea $\mathcal{G} \equiv \leftarrow \mathcal{Q}_0$* `with e2 =` $\dot{et}_2^0$ *un objetivo* lef-Prolog*, tal que E y E' son lef-expresiones diferentes en $\mathcal{Q}_0$ y $\Psi_0 \equiv \langle 1, void, \dot{et}_2^0 \rangle$.*
Entonces,

$$\left\langle \boxed{\mathrm{L}_{\Psi_0}} \mathcal{Q}_0 \boxed{\mathrm{R}_{\Psi_0}} \,;\theta_0 \right\rangle \rightarrow_{FR}{}^{E} \left\langle \boxed{\mathrm{L}_{\Psi_0}} \mathcal{Q}_1 \boxed{\mathrm{R}_{\Psi_0}} \,;\theta_1 \right\rangle \rightarrow_{FR}{}^{E'\theta_1} \left\langle \boxed{\mathrm{L}_{\Psi_0}} \mathcal{Q}_2 \boxed{\mathrm{R}_{\Psi_0}} \,;\theta_2 \right\rangle$$

si, y sólo si,

$$\left\langle \boxed{\mathrm{L}_{\Psi_0}} \mathcal{Q}_0 \boxed{\mathrm{R}_{\Psi_0}} \,;\theta_0 \right\rangle \rightarrow_{FR}{}^{E'} \left\langle \boxed{\mathrm{L}_{\Psi_0}} \mathcal{Q}_1' \boxed{\mathrm{R}_{\Psi_0}} \,;\theta_1' \right\rangle \rightarrow_{FR}{}^{E\theta_1'} \left\langle \boxed{\mathrm{L}_{\Psi_0}} \mathcal{Q}_2' \boxed{\mathrm{R}_{\Psi_0}} \,;\theta_2' \right\rangle$$

donde $\mathcal{Q}_2 = \mathcal{Q}_2'$ y $\theta_2 = \theta_2'$.

Demostración. En lo que sigue, supondremos –sin pérdida de generalidad– que el cuerpo $\mathcal{Q}_0$ del objetivo tiene la forma

$$\mathcal{Q}_0 \equiv \overline{X}, E_1, \overline{Y}, E_2, \overline{Z}$$

donde $\overline{X}, \overline{Y}$ y $\overline{Z}$ son secuencias arbitrarias de lef-expresiones válidas (tal que $\overline{X}$ está encabezado por $\boxed{\mathrm{L}_{\Psi_0}}$ y $\overline{Z}$ finaliza por $\boxed{\mathrm{R}_{\Psi_0}}$) y E_1, E_2, es decir, las lef-expresiones selecionadas en el primer y segundo paso de la ef-derivación, serán átomos (denotados por A o A'), secuencias de números reales (que designamos por $\overline{r}$ o $\overline{n}$) o números reales encerrados entre marcas etiquetadas (por ejemplo, $\boxed{\mathrm{L}_{\Psi}}\, r \,\boxed{\mathrm{R}_{\Psi}}$ o $\boxed{\mathrm{L}_{\Psi}}\, n \,\boxed{\mathrm{R}_{\Psi}}$,

con $\Psi \equiv \langle p, \dot{\mathrm{et}}_1, \dot{\mathrm{et}}_2 \rangle$, $\Psi' \equiv \langle q, \dot{\mathrm{et}}'_1, \dot{\mathrm{et}}'_2 \rangle$), dependiendo de la regla de resolución de la Definición 2.3.5 usada en cada paso.

Para hacer más legible la demostración, subrayamos la lf-expresión explotada en cada paso de derivación.

Procederemos exaustivamente con cada uno de los casos posibles. Por fortuna, observemos que no es relevante si E_1 está a la izquierda o a la derecha de E_2 y, además, el caso en que el primer paso es dado con la regla $\mathcal{R}_i$ y el segundo con la regla $\mathcal{R}_j$ es análogo al caso en que el primer paso se da con la regla $\mathcal{R}_j$ y el segundo con la regla $\mathcal{R}_i$, lo que reduce drásticamente el número de opciones a considerar.

1. Primer paso con la Regla 1 y segundo paso con la Regla 1.
   Suponemos que: $\mathcal{C}_1 : (H_1 \leftarrow \overline{B_1}; \Psi) \ll \mathcal{P}$ y $\mathcal{C}_2 : (H_2 \leftarrow \overline{B_2}; \Psi') \ll \mathcal{P}$, with $\Psi \equiv \langle p, \dot{\mathrm{et}}_1, \dot{\mathrm{et}}_2 \rangle$ y $\Psi' \equiv \langle q, \dot{\mathrm{et}}'_1, \dot{\mathrm{et}}'_2 \rangle$. Entonces,

$$\begin{aligned}
&\langle \overline{X}, \underline{A}, \overline{Y}, A', \overline{Z}; \theta_0 \rangle && \rightarrow_{FR1}{}^{\mathcal{C}_1} \\
&\langle (\overline{X}, \boxed{\mathrm{L}_\Psi}\ \overline{B_1} \boxed{\mathrm{R}_\Psi}\ , \overline{Y}, \underline{A'}, \overline{Z})\theta_1; \theta_1 \rangle && \rightarrow_{FR1}{}^{\mathcal{C}_2} \\
&\langle (\overline{X}, \boxed{\mathrm{L}_\Psi}\ \overline{B_1} \boxed{\mathrm{R}_\Psi}\ , \overline{Y}, \boxed{\mathrm{L}_{\Psi'}}\ \overline{B_2} \boxed{\mathrm{R}_{\Psi'}}\ , \overline{Z})\theta_2; \theta_2 \rangle &&
\end{aligned}$$

   si, y sólo si,

$$\begin{aligned}
&\langle \overline{X}, A, \overline{Y}, \underline{A'}, \overline{Z}; \theta_0 \rangle && \rightarrow_{FR1}{}^{\mathcal{C}_2} \\
&\langle (\overline{X}, \underline{A}, \overline{Y}, \boxed{\mathrm{L}_{\Psi'}}\ \overline{B_2} \boxed{\mathrm{R}_{\Psi'}}\ , \overline{Z})\theta'_1; \theta'_1 \rangle && \rightarrow_{FR1}{}^{\mathcal{C}_1} \\
&\langle (\overline{X}, \boxed{\mathrm{L}_\Psi}\ \overline{B_1} \boxed{\mathrm{R}_\Psi}\ , \overline{Y}, \boxed{\mathrm{L}_{\Psi'}}\ \overline{B_2} \boxed{\mathrm{R}_{\Psi'}}\ , \overline{Z})\theta'_2; \theta'_2 \rangle &&
\end{aligned}$$

   donde, por el Lema 2.4.4 tenemos que $\theta_2 = \theta'_2$, lo que garantiza, además, que las primeras componentes del estado final en ambas derivaciones son sintácticamente iguales, como deseábamos probar.

2. Primer paso con la Regla 1 y segundo paso con la Regla 2.
   Suponemos que: $\mathcal{C}_1 : (H_1 \leftarrow \overline{B_1}; \Psi) \ll \mathcal{P}$ y $\mathcal{C}_2 : (H_2 \leftarrow; q) \ll \mathcal{P}$, with $\Psi \equiv \langle p, \dot{\mathrm{et}}_1, \dot{\mathrm{et}}_2 \rangle$.

Entonces,

$$\langle \overline{X}, \underline{A}, \overline{Y}, A', \overline{Z}; \theta_0 \rangle \qquad \rightarrow_{FR1}{}^{\mathcal{C}_1}$$

$$\langle (\overline{X}, \boxed{\mathrm{L}_\Psi}\ \overline{B_1}\ \boxed{\mathrm{R}_\Psi}, \overline{Y}, \underline{A'}, \overline{Z})\theta_1; \theta_1 \rangle \qquad \rightarrow_{FR2}{}^{\mathcal{C}_2}$$

$$\langle (\overline{X}, \boxed{\mathrm{L}_\Psi}\ \overline{B_1}\ \boxed{\mathrm{R}_\Psi}, \overline{Y}, q, \overline{Z})\theta_2; \theta_2 \rangle$$

si, y sólo si,

$$\langle \overline{X}, A, \overline{Y}, \underline{A'}, \overline{Z}; \theta_0 \rangle \qquad \rightarrow_{FR2}{}^{\mathcal{C}_2}$$

$$\langle (\overline{X}, \underline{A}, \overline{Y}, q, \overline{Z})\theta_1'; \theta_1' \rangle \qquad \rightarrow_{FR1}{}^{\mathcal{C}_1}$$

$$\langle (\overline{X}, \boxed{\mathrm{L}_\Psi}\ \overline{B_1}\ \boxed{\mathrm{R}_\Psi}, \overline{Y}, q, \overline{Z})\theta_2'; \theta_2' \rangle$$

donde, por el Lema 2.4.4 tenemos $\theta_2 = \theta_2'$ y, además, las primeras componentes del estado final en ambas derivaciones son sintácticamente iguales, como queríamos probar.

3. Primer paso con la Regla 1 y segundo paso con la Regla 3.
   Suponemos que: $\mathcal{C}_1 : (H_1 \leftarrow \overline{B_1}; \Psi) \ll \mathcal{P}$ with $\Psi \equiv \langle p, \dot{\mathrm{e}}\mathrm{t}_1, \dot{\mathrm{e}}\mathrm{t}_2 \rangle$, y $\Psi' \equiv \langle q, \dot{\mathrm{e}}\mathrm{t}_1', \dot{\mathrm{e}}\mathrm{t}_2' \rangle$.
   Entonces,

$$\langle \overline{X}, \underline{A}, \overline{Y}, \boxed{\mathrm{L}_{\Psi'}}\ n\ \boxed{\mathrm{R}_{\Psi'}}, \overline{Z}; \theta_0 \rangle \qquad \rightarrow_{FR1}{}^{\mathcal{C}_1}$$

$$\langle (\overline{X}, \boxed{\mathrm{L}_\Psi}\ \overline{B_1}\ \boxed{\mathrm{R}_\Psi}, \overline{Y}, \underline{\boxed{\mathrm{L}_{\Psi'}}\ n\ \boxed{\mathrm{R}_{\Psi'}}}, \overline{Z})\theta_1; \theta_1 \rangle \qquad \rightarrow_{FR3}$$

$$\langle (\overline{X}, \boxed{\mathrm{L}_\Psi}\ \overline{B_1}\ \boxed{\mathrm{R}_\Psi}, \overline{Y}, \dot{\mathrm{e}}\mathrm{t}_1'(q, n), \overline{Z})\theta_1; \theta_1 \rangle$$

si, y sólo si,

$$\langle \overline{X}, A, \overline{Y}, \underline{\boxed{\mathrm{L}_{\Psi'}}\ n\ \boxed{\mathrm{R}_{\Psi'}}}, \overline{Z}; \theta_0 \rangle \qquad \rightarrow_{FR3}$$

$$\langle \overline{X}, \underline{A}, \overline{Y}, \dot{\mathrm{e}}\mathrm{t}_1'(q, n), \overline{Z}; \theta_0 \rangle \qquad \rightarrow_{FR1}{}^{\mathcal{C}_1}$$

$$\langle (\overline{X}, \boxed{\mathrm{L}_\Psi}\ \overline{B_1}\ \boxed{\mathrm{R}_\Psi}, \overline{Y}, \dot{\mathrm{e}}\mathrm{t}_1'(q, n), \overline{Z})\theta_1; \theta_1 \rangle$$

como deseábamos probar.

4. Primer paso con la Regla 1 y segundo paso con la Regla 4.
   Suponemos que: $\mathcal{C}_1 : (H_1 \leftarrow \overline{B_1}; \Psi) \ll \mathcal{P}$ with $\Psi \equiv \langle p, \dot{\text{et}}_1, \dot{\text{et}}_2 \rangle$, y $\Psi' \equiv \langle q, \dot{\text{et}}_1', \dot{\text{et}}_2' \rangle$.
   Entonces,

$$\langle \overline{X}, \underline{A}, \overline{Y}, \boxed{\text{L}_{\Psi'}}\ \overline{r}\ \boxed{\text{R}_{\Psi'}}, \overline{Z}; \theta_0 \rangle \qquad \rightarrow_{FR1}{}^{\mathcal{C}_1}$$

$$\langle (\overline{X}, \boxed{\text{L}_{\Psi}}\ \overline{B_1} \boxed{\text{R}_{\Psi}}, \overline{Y}, \underline{\boxed{\text{L}_{\Psi'}}\ \overline{r}\ \boxed{\text{R}_{\Psi'}}}, \overline{Z}) \theta_1; \theta_1 \rangle \quad \rightarrow_{FR4}$$

$$\langle (\overline{X}, \boxed{\text{L}_{\Psi}}\ \overline{B_1} \boxed{\text{R}_{\Psi}}, \overline{Y}, \dot{\text{et}}_2'(\overline{r}), \overline{Z}) \theta_1; \theta_1 \rangle$$

   si, y sólo si,

$$\langle \overline{X}, A, \overline{Y}, \underline{\boxed{\text{L}_{\Psi'}}\ \overline{r}\ \boxed{\text{R}_{\Psi'}}}, \overline{Z}; \theta_0 \rangle \qquad \rightarrow_{FR4}$$

$$\langle \overline{X}, \underline{A}, \overline{Y}, \dot{\text{et}}_2'(\overline{r}), \overline{Z}; \theta_0 \rangle \qquad \rightarrow_{FR1}{}^{\mathcal{C}_1}$$

$$\langle (\overline{X}, \boxed{\text{L}_{\Psi}}\ \overline{B_1} \boxed{\text{R}_{\Psi}}, \overline{Y}, \dot{\text{et}}_2'(\overline{r}), \overline{Z}) \theta_1; \theta_1 \rangle$$

   que da el resultado.

5. Primer paso con la Regla 2 y segundo con Regla 2.
   Suponemos que: $\mathcal{C}_1 : (H_1 \leftarrow; p) \ll \mathcal{P}$ y $\mathcal{C}_2 \equiv (H_2 \leftarrow; q) \ll \mathcal{P}$.
   Entonces,

$$\langle \overline{X}, \underline{A}, \overline{Y}, A', \overline{Z}; \theta_0 \rangle \qquad \rightarrow_{FR2}{}^{\mathcal{C}_1}$$

$$\langle (\overline{X}, p, \overline{Y}, \underline{A'}, \overline{Z}) \theta_1; \theta_1 \rangle \quad \rightarrow_{FR2}{}^{\mathcal{C}_2}$$

$$\langle (\overline{X}, p, \overline{Y}, q, \overline{Z}) \theta_2; \theta_2 \rangle$$

   si, y sólo si,

$$\langle \overline{X}, A, \overline{Y}, \underline{A'}, \overline{Z}; \theta_0 \rangle \qquad \rightarrow_{FR2}{}^{\mathcal{C}_2}$$

$$\langle (\overline{X}, \underline{A}, \overline{Y}, q, \overline{Z}) \theta_1'; \theta_1' \rangle \quad \rightarrow_{FR2}{}^{\mathcal{C}_1}$$

$$\langle (\overline{X}, p, \overline{Y}, q, \overline{Z}) \theta_2'; \theta_2' \rangle$$

   y por el Lema 2.4.4 resulta $\theta_2 = \theta_2'$, lo que supone también la igualdad sintáctica de las primeras componentes del estado final en ambas derivaciones, como necesitábamos.

6. Primer paso con la Regla 2 y segundo paso con la Regla 3.
   Suponemos que: $\mathcal{C}_1 : (H_1 \leftarrow ; q) \ll \mathcal{P}$ y $\Psi \equiv \langle p, \dot{\mathrm{e}}\mathrm{t}_1, \dot{\mathrm{e}}\mathrm{t}_2 \rangle$.
   Entonces,

$$\langle \overline{X}, \underline{A}, \overline{Y}, \boxed{\mathrm{L}_\Psi}\ n\ \boxed{\mathrm{R}_\Psi}\ , \overline{Z}; \theta_0 \rangle \quad \rightarrow_{FR2}{}^{\mathcal{C}_1}$$

$$\langle (\overline{X}, q, \overline{Y}, \underline{\boxed{\mathrm{L}_\Psi}\ n\ \boxed{\mathrm{R}_\Psi}\ , \overline{Z}}) \theta_1; \theta_1 \rangle \quad \rightarrow_{FR3}$$

$$\langle (\overline{X}, q, \overline{Y}, \dot{\mathrm{e}}\mathrm{t}_1(p, n), \overline{Z}) \theta_1; \theta_1 \rangle$$

   si, y sólo si,

$$\langle \overline{X}, A, \overline{Y}, \underline{\boxed{\mathrm{L}_\Psi}\ n\ \boxed{\mathrm{R}_\Psi}\ , \overline{Z}}; \theta_0 \rangle \quad \rightarrow_{FR3}$$

$$\langle \overline{X}, \underline{A}, \overline{Y}, \dot{\mathrm{e}}\mathrm{t}_1(p, n), \overline{Z}; \theta_0 \rangle \quad \rightarrow_{FR2}{}^{\mathcal{C}_1}$$

$$\langle (\overline{X}, q, \overline{Y}, \dot{\mathrm{e}}\mathrm{t}_1(p, n), \overline{Z}) \theta_1; \theta_1 \rangle$$

   como deseábamos probar.

7. Primer paso con la Regla 2 y segundo paso con la Regla 4.
   Suponemos que: $\mathcal{C}_1 : (H_1 \leftarrow \overline{B_1}; q) \ll \mathcal{P}$ y $\Psi \equiv \langle p, \dot{\mathrm{e}}\mathrm{t}_1, \dot{\mathrm{e}}\mathrm{t}_2 \rangle$.
   Entonces,

$$\langle \overline{X}, \underline{A}, \overline{Y}, \boxed{\mathrm{L}_\Psi}\ \overline{r}\ \boxed{\mathrm{R}_\Psi}\ , \overline{Z}; \theta_0 \rangle \quad \rightarrow_{FR2}{}^{\mathcal{C}_1}$$

$$\langle (\overline{X}, q, \overline{Y}, \underline{\boxed{\mathrm{L}_\Psi}\ \overline{r}\ \boxed{\mathrm{R}_\Psi}\ , \overline{Z}}) \theta_1; \theta_1 \rangle \quad \rightarrow_{FR4}$$

$$\langle (\overline{X}, q, \overline{Y}, \dot{\mathrm{e}}\mathrm{t}_2(\overline{r}), \overline{Z}) \theta_1; \theta_1 \rangle$$

   si, y sólo si,

$$\langle \overline{X}, A, \overline{Y}, \underline{\boxed{\mathrm{L}_\Psi}\ \overline{r}\ \boxed{\mathrm{R}_\Psi}\ , \overline{Z}}; \theta_0 \rangle \quad \rightarrow_{FR4}$$

$$\langle \overline{X}, \underline{A}, \overline{Y}, \dot{\mathrm{e}}\mathrm{t}_2(\overline{r}), \overline{Z}; \theta_0 \rangle \quad \rightarrow_{FR2}{}^{\mathcal{C}_1}$$

$$\langle (\overline{X}, q, \overline{Y}, \dot{\mathrm{e}}\mathrm{t}_2(\overline{r}), \overline{Z}) \theta_1; \theta_1 \rangle$$

   que nos da el resultado.

8. Primer paso con la Regla 3 y segundo paso con la Regla 3.

Suponemos que $\Psi \equiv \langle p, \dot{\mathrm{e}}\mathrm{t}_1, \dot{\mathrm{e}}\mathrm{t}_2 \rangle$ y $\Psi' \equiv \langle q, \dot{\mathrm{e}}\mathrm{t}_1', \dot{\mathrm{e}}\mathrm{t}_2' \rangle$. Entonces,

$$\langle \overline{X}, \underline{\boxed{\mathrm{L}_\Psi}\ r\ \boxed{\mathrm{R}_\Psi}}, \overline{Y}, \boxed{\mathrm{L}_{\Psi'}}\ n\ \boxed{\mathrm{R}_{\Psi'}}, \overline{Z}; \theta_0 \rangle \quad \rightarrow_{FR3}$$

$$\langle \overline{X}, \dot{\mathrm{e}}\mathrm{t}_1(p, r), \overline{Y}, \underline{\boxed{\mathrm{L}_{\Psi'}}\ n\ \boxed{\mathrm{R}_{\Psi'}}}, \overline{Z}; \theta_0 \rangle \quad \rightarrow_{FR3}$$

$$\langle \overline{X}, \dot{\mathrm{e}}\mathrm{t}_1(p, r), \overline{Y}, \dot{\mathrm{e}}\mathrm{t}_1'(q, n), \overline{Z}; \theta_0 \rangle$$

si, y sólo si,

$$\langle \overline{X}, \boxed{\mathrm{L}_\Psi}\ r\ \boxed{\mathrm{R}_{\Psi'}}, \overline{Y}, \underline{\boxed{\mathrm{L}_{\Psi'}}\ n\ \boxed{\mathrm{R}_{\Psi'}}}, \overline{Z}; \theta_0 \rangle \quad \rightarrow_{FR3}$$

$$\langle \overline{X}, \underline{\boxed{\mathrm{L}_\Psi}\ r\ \boxed{\mathrm{R}_{\Psi'}}}, \overline{Y}, \dot{\mathrm{e}}\mathrm{t}_1'(q, n), \overline{Z}; \theta_0 \rangle \quad \rightarrow_{FR3}$$

$$\langle \overline{X}, \dot{\mathrm{e}}\mathrm{t}_1(p, r), \overline{Y}, \dot{\mathrm{e}}\mathrm{t}_1'(q, n), \overline{Z}; \theta_0 \rangle$$

como queríamos probar.

9. Primer paso con la Regla 3 y segundo paso con la Regla 4.
Suponemos que $\Psi \equiv \langle p, \dot{\mathrm{e}}\mathrm{t}_1, \dot{\mathrm{e}}\mathrm{t}_2 \rangle$ y $\Psi' \equiv \langle q, \dot{\mathrm{e}}\mathrm{t}_1', \dot{\mathrm{e}}\mathrm{t}_2' \rangle$. Entonces,

$$\langle \overline{X}, \underline{\boxed{\mathrm{L}_\Psi}\ r\ \boxed{\mathrm{R}_\Psi}}, \overline{Y}, \boxed{\mathrm{L}_{\Psi'}}\ \overline{n}\ \boxed{\mathrm{R}_{\Psi'}}, \overline{Z}; \theta_0 \rangle \quad \rightarrow_{FR3}$$

$$\langle \overline{X}, \dot{\mathrm{e}}\mathrm{t}_1(p, r), \overline{Y}, \underline{\boxed{\mathrm{L}_{\Psi'}}\ \overline{n}\ \boxed{\mathrm{R}_{\Psi'}}}, \overline{Z}; \theta_0 \rangle \quad \rightarrow_{FR4}$$

$$\langle \overline{X}, \dot{\mathrm{e}}\mathrm{t}_1(p, r), \overline{Y}, \dot{\mathrm{e}}\mathrm{t}_2'(\overline{n}), \overline{Z}; \theta_0 \rangle$$

si, y sólo si,

$$\langle \overline{X}, \boxed{\mathrm{L}_\Psi}\ r\ \boxed{\mathrm{R}_\Psi}, \overline{Y}, \underline{\boxed{\mathrm{L}_{\Psi'}}\ \overline{n}\ \boxed{\mathrm{R}_{\Psi'}}}, \overline{Z}; \theta_0 \rangle \quad \rightarrow_{FR4}$$

$$\langle \overline{X}, \underline{\boxed{\mathrm{L}_\Psi}\ r\ \boxed{\mathrm{R}_\Psi}}, \overline{Y}, \dot{\mathrm{e}}\mathrm{t}_2'(\overline{n}), \overline{Z}; \theta_0 \rangle \quad \rightarrow_{FR3}$$

$$\langle \overline{X}, \dot{\mathrm{e}}\mathrm{t}_1(p, r), \overline{Y}, \dot{\mathrm{e}}\mathrm{t}_2'(\overline{n}), \overline{Z}; \theta_0 \rangle$$

como queríamos.

10. Primer paso con la Regla 4 y segundo paso con la Regla 4.

Suponemos que $\Psi \equiv \langle p, \dot{\mathrm{et}}_1, \dot{\mathrm{et}}_2 \rangle$ y $\Psi' \equiv \langle q, \dot{\mathrm{et}}_1', \dot{\mathrm{et}}_2' \rangle$. Entonces,

$$\langle \overline{X}, \underline{\boxed{\mathrm{L}_\Psi}\ \overline{r}\ \boxed{\mathrm{R}_\Psi}}, \overline{Y}, \boxed{\mathrm{L}_{\Psi'}}\ \overline{n}\ \boxed{\mathrm{R}_{\Psi'}}, \overline{Z}; \theta_0 \rangle \quad \rightarrow_{FR4}$$

$$\langle \overline{X}, \dot{\mathrm{et}}_2(\overline{r}), \overline{Y}, \underline{\boxed{\mathrm{L}_{\Psi'}}\ \overline{n}\ \boxed{\mathrm{R}_{\Psi'}}}, \overline{Z}; \theta_0 \rangle \quad \rightarrow_{FR4}$$

$$\langle \overline{X}, \dot{\mathrm{et}}_2(\overline{r}), \overline{Y}, \dot{\mathrm{et}}_2'(\overline{n}), \overline{Z}; \theta_0 \rangle$$

si, y sólo si,

$$\langle \overline{X}, \boxed{\mathrm{L}_\Psi}\ \overline{r}\ \boxed{\mathrm{R}_\Psi}, \overline{Y}, \underline{\boxed{\mathrm{L}_{\Psi'}}\ \overline{n}\ \boxed{\mathrm{R}_{\Psi'}}}, \overline{Z}; \theta_0 \rangle \quad \rightarrow_{FR4}$$

$$\langle \overline{X}, \underline{\boxed{\mathrm{L}_\Psi}\ \overline{r}\ \boxed{\mathrm{R}_\Psi}}, \overline{Y}, \dot{\mathrm{et}}_2'(\overline{n}), \overline{Z}; \theta_0 \rangle \quad \rightarrow_{FR4}$$

$$\langle \overline{X}, \dot{\mathrm{et}}_2(\overline{r}), \overline{Y}, \dot{\mathrm{et}}_2'(\overline{n}), \overline{Z}; \theta_0 \rangle$$

que nos da el resultado.

□

Por último, la prueba de la independencia de la regla de computación expresada en el Teorema 2.4.3 –y su caso particular expresado en el Teorema 2.4.2– es ahora inmediata, puesto que se obtiene sin más que aplicar reiteradamente el Lema de Conmutación 2.4.5.

2.5. El lenguaje multi-adjunto

La programación lógica multi-adjunta ha sido introducida en [Medina *et al.*, 2001d,c] como una generalización de la programación lógica monótona y residuada considerada en [Damásio y Moniz-Pereira, 2000; Dekhtyar y Subrahmanian, 2000; Damásio y Moniz-Pereira, 2001b, 2002, 2004] que, a su vez, extiende (entre otros) los programas lógicos probabilísticos híbridos de Dekhtyar y Subrahmanian [2000], los programas de bases de datos deductivas probabilísticas de Lakshmanan y Sadri [2001], los programas lógicos probabilísticos ordinarios de Lukasiewicz [2001b] y los programas del marco de deducción cuantitativa de van Emden [1986] (que, por su parte, extienden a los programas de Dubois *et al.* [1991b]).

En consecuencia, la programación lógica multi-adjunta constituye un marco muy general en el que se pueden ubicar otros lenguajes lógicos difusos, lo que lo hace muy

atractivo para formular reglas de transformación de programas que, una vez caracterizadas en este marco, podrían ser aplicadas a estos lenguajes (instancias del lenguaje multi-adjunto). En concreto, en la primera parte de esta tesis definiremos una regla de desplegado para programas multi-adjuntos. Además, en la segunda parte (véase el Capítulo 7), adaptaremos a este marco los conceptos clásicos de evaluación parcial a fin de disponer de técnicas de especialización de programas multi-adjuntos y poder abordar asimismo el cálculo eficiente de reductantes (necesarios para garantizar la completitud).

La selección de este lenguaje está justificada también por el alto nivel de expresividad, así como por disponer de una semántica procedural clara. El primer punto es útil para acentuar la relevancia y generalidad de nuestros resultados, facilitando amplia difusión de los mismos, en tanto que la segunda parte es crucial para definir de manera efectiva nuestras transformaciones (en particular, para una definición formal de reglas de desplegado y de técnicas de evaluación parcial, es preceptivo que la semántica procedural esté formalizada en términos de un sistema de transición de estados).

Observando la expresividad, la aproximación lógica multi-adjunta representa un marco difuso extremadamente flexible basado en reglas con un grado de verdad asociado, que mejora ampliamente antiguas aproximaciones introducidas previamente en este campo (véase, por ejemplo, el sistema Prolog–Elf de Ishizuka y Kanai [1985], el sistema Fril de Baldwin *et al.* [1995] y variantes difusas de Prolog propuestas en [Li y Liu, 1990; Vojtáš y Paulík, 1996; Guadarrama *et al.*, 2004]).

En [Damásio *et al.*, 2007][4], los autores justifican la necesidad de considerar un marco muy abstracto a fin de poder establecer y demostrar resultados generales de terminación, semántica de punto fijo, procedimientos de *contestación de preguntas* (o *query answering*), etc., que puedan aplicarse a varios escenarios, aparentemente distintos, como la dedución cuantitativa de van Emden, la programación lógica posibilista, la resolución-SLD no-clásica, los programas lógicos probabilísticos ordinarios y las bases de datos deductivas probabilísticas.

Por el interés creciente de los modelos de razonamiento con información "imperfecta", hemos observado que en los últimos años ha proliferado un alto número de propuestas para la integración del razonamiento aproximado en el contexto de la programación lógica (clásica). Y existen, naturalmente, otros sistemas que no están totalmente cubiertos por la aproximación lógica multi-adjunta. Es el caso, por

[4] En este trabajo, se incorporan tipos al lenguaje lógico multi-adjunto.

ejemplo, de la programación lógica basada en similaridad [Sessa, 2002] y la programación lógica anotada [Kifer y Subrahmanian, 1992]. Sin embargo, para el primer caso, podemos encontrar en [Medina *et al.*, 2004] algunos análisis que establecen ciertas correspondencias entre ambos lenguajes que sólo son útiles a nivel teórico. En particular, puede probarse que el efecto de los métodos de unificación/resolución basados en similaridad de Sessa [2002] pueden replicarse de alguna manera (a nivel teórico) aplicando el mecanismo procedural de los programas lógicos multi-adjuntos ampliado con reglas con pesos especiales que simulan ecuaciones de similaridad.

Por otra parte, para los programas lógicos anotados de Kifer y Subrahmanian [1992] y especialmente para la más reciente versión de Straccia [2005a]; Loyer y Straccia [2005], que contempla programas de sintaxis muy sencilla, pasamos a valorar las ventajas e inconvenientes que comporta para nosotros, a efectos de formular la transformación de desplegado y de adaptar lás técnicas de evaluación parcial.

La mayoría de los marcos difusos (incluyendo la programación lógica multi-adjunta) presentan una limitación importante: no admitir ninguna forma de negación no-monótona[5]. No es el caso del que se contempla en [Straccia, 2005a; Loyer y Straccia, 2005], que ha sido objeto de nuestra atención por este hecho relevante y porque es también un marco muy general: según se recoge en [Loyer y Straccia, 2005], los programas lógicos normales contemplados permiten capturar, entre otros, el marco de la deducción cuantitativa de van Emden [1986], el marco posibilista de Dubois *et al.* [1991b], el marco de las bases deductivas de datos de Lakshmanan y Shiri [2001] y la programación lógica difusa de Vojtáš [2001].

Comparando esta aproximación (desde nuestro punto de vista, es decir, a efectos de abordar las transformaciones de desplegado, las técnicas de evaluación parcial y el cálculo de reductantes) con la programación multi-adjunta podemos observar:

- **Expresividad**. Aparte del hecho de que en la aproximación lógica multi-adjunta la discusión se centra en el caso monótono, ignorando la negación por defecto, la noción subyacente de retículo es diferente de la de birretículo [Fitting, 1991], que es un estructura ligeramente más general que la de retículo.

  Estos programas son capaces de afrontar la negación no-monótona y proporcionar una vía elegante y potente para combinar grados de certeza y de duda en la reglas de programa. Aunque esta laguna del marco multi-adjunto ha sido aliviada en [Damásio *et al.*, 2007] considerando múltiples tipos a través de los

[5] In [Damásio *et al.*, 2007], estos autores afirman que "ya hemos comenzado la investigación en esta dirección" (es decir, se plantean incorporar la negación en los programas multi-adjuntos).

llamados multi-retículos, el tratamiento propuesto semeja en cualquier caso más débil que el desarrollado en [Straccia, 2005a; Loyer y Straccia, 2005].

- **Sintaxis**. En [Straccia, 2005a; Loyer y Straccia, 2005], las reglas de programa tienen la forma $A \leftarrow f(B_1, \ldots, B_n)$ donde f es un operator interpretado por una función de verdad que es combinación de otras funciones de verdad de las conectivas del cuerpo de la regla, en tanto que A y B_i son átomos. La sintaxis es muy próxima a la usada en la lógica multi-adjunta, como explícitamente se dice en [Damásio *et al.*, 2007]: "a veces, representaremos los cuerpos de las fórmulas como $@[B_1, \ldots, B_n]$, donde los B_i son variables proposicionales del cuerpo y $@$ es el agregador monótono obtenido como una composición"[6]. Como vimos hasta aquí, salvando las diferencias en cuanto al retículo que se asocia a cada programa, ambas aproximaciones son bastante similares desde un punto de vista sintáctico. Sin embargo, en [Straccia, 2005a] no se permiten símbolos de función y los autores afirman que, en aras de una presentación más sencilla, se limita la atención al caso proposicional.

- **Semántica Procedural**. La mayor diferencia entre ambas aproximaciones, a los efectos que nos proponemos (la transformación y optimización de programas), surge a nivel procedural. Veremos, en la Sección 3.1, que la formalización de la semántica procedural del lenguaje multi-adjunto como un sistema de transición de estados resulta crucial para definir el desplegado, para construir árboles de desplegado y formalizar el concepto de evaluación parcial en el Capítulo 7. En [Straccia, 2005a; Loyer y Straccia, 2005] se aporta un procedimiento general de contestación de preguntas, de arriba-abajo, para programas lógicos normales (que son abordados también en las referencias [Loyer y Straccia, 2002b, 2005]) sobre retículos y birretículos. El método puede concebirse como una semántica procedural para este tipo de programas, y tiene la ventaja de que puede instanciarse para adaptarse a diferentes semánticas (como la semántica de Kripke-Kleene y la semántica Bien-Fundada –véase también [Loyer y Straccia, 2003, 2004]–) cuando evaluamos objetivos. Desafortunadamente, el procedimiento usado de contestación de preguntas se presenta en forma algorítmica que está lejos de la formulación como sistema de transición de estados sobre la que tradicionalmente se contemplan las reglas de transformación de programas que abordamos.

[6] Véase también [Guerrero y Moreno, 2008], donde se aporta una operación de transformación llamada "agregación" que adapta automáticamente reglas de programa a este formato más simple.

En consecuecia, después de todos los pormenores observados, podemos extraer las siguientes conclusiones: *i*) el marco de [Straccia, 2005a; Loyer y Straccia, 2005] es una aproximación atractiva en la que (una vez superado el caso proposicional y/o permitiendo la presencia de símbolos de función en su sintaxis) estamos interesados, para adaptar nuestras herramientas y poder capturar recursos expresivos importantes como la negación no-monótona; *ii*) a tal fin, es preceptivo investigar previamente nuevas vías de formulación de su semántica procedural y mecanismos de contestación de preguntas; y *iii*) pensamos que la experiencia acumulada en el desarrollo de técnicas para la transformación de programas lógicos multi-adjuntos, nos puede servir para extender nuestros resultados a un marco diferente, y sugestivo, como el que se contempla en las referencias [Straccia, 2005a,b; Loyer y Straccia, 2002b, 2003, 2004, 2005].

En definitiva, entre los distintos lenguajes de programación lógica difusa que podemos encontrar en la literatura (véase la Sección 1.5 para una mayor documentación del área), el lenguaje multi-adjunto es un lenguaje muy potente y expresivo, dado que permitirá distintas implicaciones en la reglas y conectivas muy generales en los cuerpos de éstas. Además, resultará apropiado para abordar el desplegado de programas lógicos difusos debido, principalmente, a que posee una sintaxis sencilla y una semántica operacional adecuada para dar cuenta de los grados de verdad. Es, por tanto, el marco (más general) elegido para formular la transformación de desplegado, estudiar sus propiedades de corrección y de eficiencia; para formular, asimismo, la evaluación parcial en el contexto difuso y su aplicación al cálculo de reductantes.

Resumimos en lo que sigue los principales rasgos de la programación lógica multi-adjunta descrita, entre otras, en las referencias [Medina *et al.*, 2001d,c,b,a; Medina y Ojeda-Aciego, 2002; Medina *et al.*, 2004; Medina y Ojeda-Aciego, 2004; Damásio *et al.*, 2007].

2.5.1. Nociones básicas

La programación lógica multi-adjunta utiliza un lenguaje de primer orden $\mathcal{L}$ conteniendo variables, constantes, símbolos de función, símbolos de predicado, los cuantificadores $\forall$ y $\exists$, y varias conectivas (arbitrarias) para capturar diferentes enlaces entre predicados:

$$\begin{array}{llll} \wedge_1, & \wedge_2, & \ldots, & \wedge_k \quad \texttt{(conjunciones)} \\ \vee_1, & \vee_2, & \ldots, & \vee_l \quad \texttt{(disyunciones)} \\ \leftarrow_1, & \leftarrow_2, & \ldots, & \leftarrow_m \quad \texttt{(implicaciones)} \\ @_1, & @_2, & \ldots, & @_n \quad \texttt{(agregadores)} \end{array}$$

Admitimos, por tanto, distintas implicaciones ($\leftarrow_1, \leftarrow_2, \ldots, \leftarrow_m$) y otras conectivas que también podrían agruparse todas ellas bajo el término de *agregador* y que se usarán para combinar/propagar el grado de verdad en las reglas. Estos agregadores (denotados por $@_1, @_2, \ldots, @_n$) subsumen a las conjunciones (denotadas por $\wedge_1, \wedge_2, \ldots, \wedge_k$), las disyunciones ($\vee_1, \vee_2, \ldots, \vee_l$), operadores media y otros operadores híbridos, como ya hemos considerado en el capítulo anterior –para el caso en que el retículo L era el intervalo $[0,1]$–. Por definición, la función de verdad, en un retículo $(L, \leq)$, de un agregador n-ario $\dot{@} : L^n \to L$ es monótona y satisface las condiciones de frontera $\dot{@}(\top, \ldots, \top) = \top$, $\dot{@}(\bot, \ldots, \bot) = \bot$.

Aunque las conectivas $\wedge_i$, $\vee_i$ y $@_i$ son operadores binarios, habitualmente los generalizamos para contemplarlos como funciones con un número arbitrario de argumentos. De este modo, escribiremos[7], $@_i(x_1, \ldots, x_n)$ en lugar de $@_i(x_1, @_i\,(x_2, \ldots, @_i(x_{n-1}, x_n)\ldots))$.

Asumimos también que en el lenguaje $\mathcal{L}$ se admiten elementos $\alpha \in L$ y conjunciones $\&_i$ de un retículo multi-adjunto, $(L, \leq, \leftarrow_1, \&_1, \ldots, \leftarrow_n, \&_n)$, equipado con una colección de pares adjuntos $\langle \leftarrow_i, \&_i \rangle$, donde cada $\&_i$ es una conjunción[8] ligada a la evaluación del *modus ponens*. En general, el conjunto de grados de verdad L es cualquier retículo (acotado y) completo aunque en los ejemplos, a efectos de una mejor legibilidad, consideraremos el intervalo $[0,1]$ con el orden usual (que es un retículo totalmente ordenado), salvo en aquéllos en que el carácter parcial del orden sea relevante.

La programación lógica multi-adjunta está especialmente interesada en un subconjunto de fórmulas del lenguaje de primer orden $\mathcal{L}$ descrito anteriormente. Una *regla* es una fórmula $A \leftarrow \mathcal{B}$, donde A es una fórmula atómica (habitualmente llamada *cabeza*) y $\mathcal{B}$ es una fórmula construida a partir de fórmulas atómicas $B_1, \ldots, B_n$

[7] Puesto que para algunas conectivas no puede garantizarse la asociatividad (ni la conmutatividad) los paréntesis son necesarios [Vojtáš, 2001] y, de este modo, convenimos el uso de las conectivas de cualquier número de argumentos.

[8] Procuramos reservar el símbolo $\&_j$ para estas conjunciones adjuntas a fin de distinguirlas de otras conjunciones del lenguaje.

– $n \geq 0$ – y conjunciones, disyunciones, agregadores, grados de verdad y conjunciones adjuntas (que llamaremos *cuerpo*). Las reglas con cuerpo vacío se llaman *hechos*. Un *objetivo* es un cuerpo planteado como una pregunta al sistema. Las variables de las reglas se suponen universalmente cuantificadas.

Las fórmulas se interpretan en el retículo multi-adjunto mencionado anteriormente. Para precisar este concepto, recogemos las definiciones de par adjunto y retículo multi-adjunto contempladas en [Medina *et al.*, 2001d] y en [Medina *et al.*, 2001c], agrupándolas del siguiente modo:

Definición 2.5.1. *Sea* $(L, \leq)$ *un retículo. Un* retículo multi-adjunto *es una* $n-$*upla* $(L, \leq, \leftarrow_1, \&_1, \ldots, \leftarrow_n, \&_n)$ *que cumple las siguientes condiciones:*

1. $(L, \leq)$ *es acotado (es decir, existe el ínfimo,* $\bot$*, y el supremo,* $\top$*, de* $(L, \leq)$*).*
2. $(L, \leq)$ *es completo*[9]*, es decir, para todo subconjunto* $X \subset L$ *existen* $inf(X)$*,* $sup(X)$[10]*.*
3. $\top \,\&_i\, v = v \,\&_i\, \top = v$*,* $\forall v \in L$*,* $i = 1, \ldots, n$*.*
4. *cada par* $(\leftarrow_i, \&_i)$*, con* $i = 1, \ldots, n$*, es un* par adjunto*, es decir:*
   a) *la operación* $\&_i$ *es creciente en ambos argumentos*[11]*.*
   b) *la operación* $\leftarrow_i$ *es creciente en el primer argumento y decreciente en el segundo argumento.*
   c) propiedad adjunta*:* $\forall x, y, z \in L$*,* $x \leq (y \leftarrow_i z) \Leftrightarrow (x \&_i\, z) \leq y$*.*

En cuanto al enunciado de la propiedad adjunta, x representará en lo sucesivo el grado de verdad de la regla $y \leftarrow_i z$, de modo que se satisface dicha regla (la correspondiente interpretación satisface dicha regla) si, y sólo si, el grado de verdad y de la cabeza es mayor o igual que la conjunción adjunta del grado de verdad z de la hipótesis con el grado de verdad x de la regla.

La noción de par adjunto ha sido introducida por primera vez en el contexto lógico por Pavelka [1979] concibiendo el conjunto de grados de verdad como una categoría

[9] En verdad, si $(L, \leq)$ es completo, entonces es acotado y podríamos reformular la definición evitando exigir explícitamente el carácter acotado.

[10] Téngase en cuenta que la existencia de $inf(X)$, $sup(X)$ está garantizada cuando X es finito, dado que L es un retículo.

[11] En la definición no se exige conmutatividad (véase [Krajci *et al.*, 2002], por ejemplo, donde la ausencia de conmutatividad es un rasgo singular de uno de los contextos de aplicación de la programación lógica multi-adjunta) ni asociatividad para esta conectiva.

y la relación entre cada conjunción y su implicación asociada como un funtor de dicha categoría. Esta noción es un nuevo ejemplo del concepto de adjunción definido en 1950 por Kan, en la teoría general de categorías.

Tal como se observa en [Medina *et al.*, 2004], los retículos residuados (véase [Dilworth, 1939; Gerla, 2004; Novak *et al.*, 1999]) son ejemplos de retículos multi-adjuntos en los que hay un sólo par adjunto y la conjunción del par determina estructa de monoide (con elemento neutro $\top$) en el retículo subyacente.

Además, en [Damasio *et al.*, 2004; Damásio *et al.*, 2004a, 2007] se contempla un marco de programación multi-adjunta (el de los programas lógicos multi-adjuntos tipificados) en el que se combina de manera muy sugerente el uso de varios retículos multi-adjuntos. Esta flexibilidad resulta apropiada para asociar a cada regla de programa dos (varios, en general) grados de verdad, pudiendo representar uno de ellos el factor de certeza y otro el factor de duda de dicha regla.

Conviene recordar finalmente que los símbolos $\&_i$ (de conjunción adjunta), y $\alpha \in L$ (grado de verdad de L) deben pertenecer al lenguaje $\mathcal{L}$, para que sea viable definir el mecanismo operacional del lenguaje lógico multi-adjunto (véase la Sección 3.1.1) y, posteriormente, formular el desplegado para este tipo de programas (véase la Sección 4.4). Por este motivo, damos en los siguientes términos la definición de programa lógico multi-adjunto (de manera análoga a como se recoge en [Medina *et al.*, 2004]), entendido como un conjunto de reglas lógicas con una grado de verdad asociado que es un elemento del retículo multi-adjunto prefijado.

Definición 2.5.2. *Un programa lógico multi-adjunto es un conjunto $\mathcal{P}$ de reglas de la forma $\langle A \leftarrow_i \mathcal{B}; v\rangle$ verificando:*

i) A es una fórmula atómica (llamada cabeza).

ii) $\mathcal{B}$ es una fórmula arbitraria, llamada cuerpo, construida con fórmulas atómicas $B_1, \ldots, B_n, n \geq 0$ y cualesquiera conjunciones, disyunciones, agregadores, grados de verdad $\alpha \in L$ y conjunciones adjuntas $\&_i$.

iii) $v \in L$ es el grado de verdad de la fórmula lógica $A \leftarrow_i \mathcal{B}$, donde $(L, \leq)$ es el retículo multi-adjunto asociado al programa $\mathcal{P}$.

Como cabe esperar, llamaremos hechos a las reglas con cuerpo $\top$ (es decir, con cuerpo vacío).

Por otra parte, entenderemos una *interpretación de Herbrand difusa*[12], $\mathcal{I}$, como una aplicación de la base de Herbrand, $B_{\mathcal{L}}$, en el retículo multi-adjunto de grados de

[12]Véase la Sección 3.2 posterior para más detalle.

verdad L, de modo que el grado de verdad de un átomo básico $A \in B_{\mathcal{L}}$ es un elemento $\mathcal{I}(A) \in L$. Habitualmente, fijada una asignación ϑ que asocia elementos del universo de Herbrand $U_{\mathcal{L}}$ a términos, la valoración de una fórmula bajo una interpretación se obtiene por inducción estructural sobre la complejidad de esta fórmula:

$$\begin{aligned}
\mathcal{I}(p(t_1,\ldots,t_n))[\vartheta] &= \mathcal{I}(p(t_1\vartheta,\ldots,t_n\vartheta)),\\
\mathcal{I}(c(A_1,\ldots,A_n))[\vartheta] &= \dot{c}(\mathcal{I}(A_1)[\vartheta],\ldots,\mathcal{I}(A_n)[\vartheta]),\\
\mathcal{I}(A \leftarrow \mathcal{B})[\vartheta] &= \mathcal{I}(A)[\vartheta] \dot{\leftarrow} \mathcal{I}(\mathcal{B})[\vartheta],\\
\mathcal{I}((\forall x)\mathcal{A})[\vartheta] &= inf\{\mathcal{I}(\mathcal{A})[\vartheta'] \mid \vartheta' \ \ x\text{–equivalente a } \ \vartheta\},
\end{aligned}$$

donde p es un símbolo de predicado, c una conectiva arbitraria, A y A_i fórmulas atómicas, $\mathcal{B}$ cualquier cuerpo, $\mathcal{A}$ cualquier fórmula, y denotamos la función significado (función grado de verdad) de una conectiva c por $\dot{c}$. Cuando la valoración ϑ no sea relevante, la omitiremos en la interpretación de una fórmula.

Podemos describir los problemas reales en los que tengamos conocimiento vago por medio de teorías difusas que constituyen los programas lógicos difusos.

Definición 2.5.3. *Una* teoría difusa *es una función parcial T que asocia a cada fórmula un elemento (grado de verdad) del retículo L.*

Un programa multi-adjunto*, $\mathcal{P}$, es una teoría difusa tal que el dominio,* $\mathrm{dom}(\mathcal{P})$*, es un conjunto finito de reglas y L es un retículo multi-adjunto equipado con varios pares adjuntos.*

Informalmente hablando, un programa lógico multi-adjunto puede verse como un conjunto de pares $\langle \mathcal{R}; \alpha \rangle$, donde $\mathcal{R}$ es una regla y $\alpha = \mathcal{P}(\mathcal{R})$ es un *grado de verdad* que expresa la confianza que el usuario del sistema tiene en la regla $\mathcal{R}$. A menudo, escribiremos "$\mathcal{R}$ with $\mathcal{P}(\mathcal{R})$" en lugar de $\langle \mathcal{C}; \mathcal{P}(\mathcal{R}) \rangle$. Los grados de verdad deben asignarse axiomáticamente por un experto.

Una interpretación $\mathcal{I}$ es un *modelo* de una teoría difusa si para cada regla $\mathcal{R} \in \mathrm{dom}(\mathcal{P})$, $\mathcal{I}(\mathcal{R}) \geq \mathcal{P}(\mathcal{R})$.

El significado asociado a un par adjunto es el siguiente: $\leftarrow_i$ es una implicación para la que $\&_i$ es una conjunción adjunta, de modo que se satisface la regla de inferencia *modus ponens (generalizado)*:

$$\frac{\langle (A \leftarrow_i B), x \rangle \quad \langle B, y \rangle}{\langle A, \&_i(x,y) \rangle}$$

Esto es, si una interpretación $\mathcal{I}$ es un modelo de $A \leftarrow_i B$ (es decir, $\mathcal{I}(A \leftarrow_i B) \succeq x$) y de B (es decir, $\mathcal{I}(B) \succeq y$) entonces $\mathcal{I}$ es un modelo de A (esto es, $\mathcal{I}(A) \succeq x \&_i y$). La corrección del mecanismo operacional puede establecerse del siguiente modo: $x \leq \mathcal{I}(A \leftarrow_i B) = \mathcal{I}(A) \dot{\leftarrow}_i \mathcal{I}(B) \leq \mathcal{I}(A) \dot{\leftarrow}_i y$, ya que "$\dot{\leftarrow}_i$" es decreciente en su segundo argumento y $\mathcal{I}(B) \succeq y$; entonces, por la propiedad adjunta, $\mathcal{I}(A) \succeq x \&_i y$.

En [Medina *et al.*, 2004], se define la semántica declarativa de un programa lógico multi-adjunto en términos del operador de punto fijo. Por nuestra parte, en el capítulo siguiente, abordamos detalladamente las semánticas conocidas para el lenguaje multi-adjunto, así como las relaciones entre elllas.

2.6. Un entorno de programación lógica difusa para la investigación

Uno de los desafíos más importantes de las tecnologías *software* para la sociedad de la información, consiste en la propuesta de nuevos métodos, técnicas, plataformas y herramientas de nueva generación que tengan un comportamiento correcto en entornos cambiantes e imprecisos. En este escenario se necesitan prioritariamente nuevos lenguajes, formalismos y teorías, pero también entornos de programación y herramientas automáticas asociadas, que den soporte sistemático y racional al desarrollo del *software*. Teniendo en mente estos fines, los lenguajes declarativos disfrutan de una serie de ventajas extras, que sin duda alguna se basan en su fuerte fundamentación en algún tipo de lógica subyacente, lo que se relaciona con la potencia expresiva de las notaciones formales, con correspondencia directa entre la sintaxis (los programas) y la semántica (lo que significan), y con los medios que permiten analizar (manual o automáticamente) el texto en un lenguaje formal para extraer conclusiones. La lógica difusa está cobrando cada día más interés como soporte de este tipo de lenguajes declarativos, donde además se buscan integraciones y combinaciones de paradigmas cada vez más potentes. Por este motivo, estamos interesados en el diseño e implementación de entornos y lenguajes de programación basados en la lógica difusa, así como la aplicación de estos lenguajes a la solución de problemas concretos mediante la construcción de programas correctos y eficientes.

En concreto, la implementación de un prototipo de intérprete/compilador para el lenguaje lógico multi-adjunto es, en la actualidad, una de las tareas prioritarias en el grupo de investigación DEC-TAU de la UCLM.

2.6.1. Características generales de FLOPER

En [Julián *et al.*, 2009] describimos un método potente para traducir programas difusos a código Prolog estándar directamente ejecutable. Mostramos las capacidades básicas de esta herramienta que aspira a ser una plataforma experimental determinante en la tarea de optimización, transformación, especialización, depuración, etc., de programas difusos, que el grupo está abordando en los últimos años.

El objetivo final es que el código compilado sea ejecutado en cualquier intérprete Prolog de un modo totalmente transparente para el usuario final, es decir, nuestra intención es que, después de introducir programas difusos y objetivos difusos al sistema, éste nos permita devolver respuestas computadas difusas (esto es, pares incluyendo grados de verdad y sustituciones) aún cuando todos los cómputos intermedios hayan sido ejecutados en un entorno de programación lógica pura (no difusa).

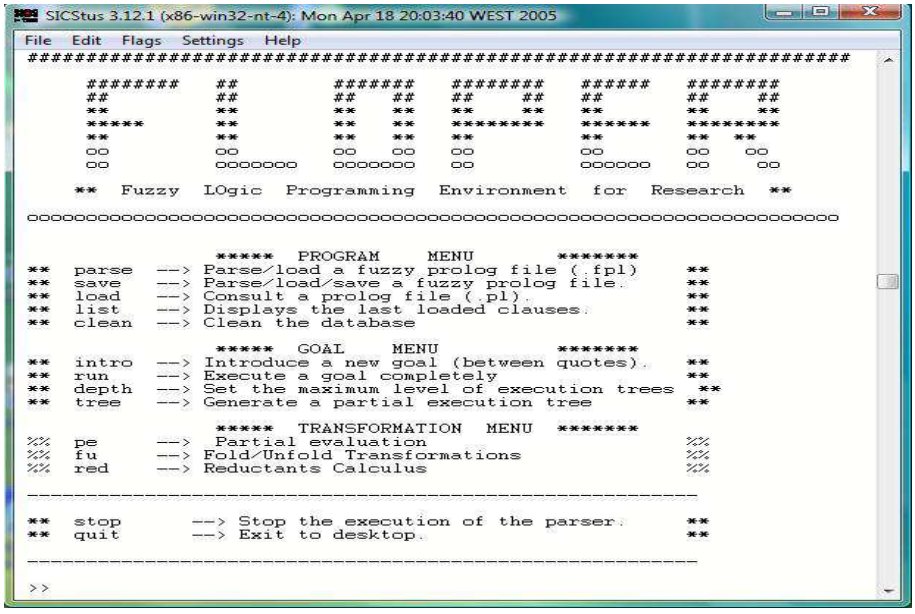
Nuestra aproximación está inspirada, en parte, en Guadarrama *et al.* [2004], donde se concibe en un lenguaje lógico difuso cercano al nuestro un intérprete usando programación lógica con restricciones sobre números reales ($CLP(\mathcal{R})$) para ser implementado de manera eficiente. Por nuestra parte, haremos uso del lenguaje de programación Prolog sin considerar una extensión $CLP(R)$, puesto que es suficiente para implementar nuestros conceptos, al mismo tiempo que la simplicidad tanto de la técnica como del lenguaje objeto resulte accesible a una audiencia más amplia.

Por el momento, nuestro sistema trata solamente con programas cuyo retículo asociado es el intervalo cerrado $[0, 1]$ y las conectivas pertenecen a alguna lógica difusa convencional (como la lógica producto, lógica de Łukasiewicz y lógica intuicionista de Gödel). Aunque la potencia expresiva de esta implementación preliminar es bastante limitada, una tarea de máxima prioridad para desarrollos futuros será lograr que el sistema acepte programas difusos con retículos multi-adjuntos de forma paramétrica, lo cual implica el diseño de protocolos, interfaces, etc., apropiados en los que ya estamos trabajando.

Hemos usado Sicstus Prolog v.3.12.5 para la ejecución de programas difusos, una vez que han sido traducidos a código Prolog, así como para la implementación de la propia herramienta.

Nuestro analizador sintáctico ha sido desarrollado usando *Gramáticas de Cláusulas Definidas* (Definite Clause Grammars, DCG's), un recurso del lenguaje Prolog que aporta una notación adecuada para enunciar las reglas de producción de una gramática de forma ágil y precisa. La aplicación contiene alrededor de 300 cláusulas

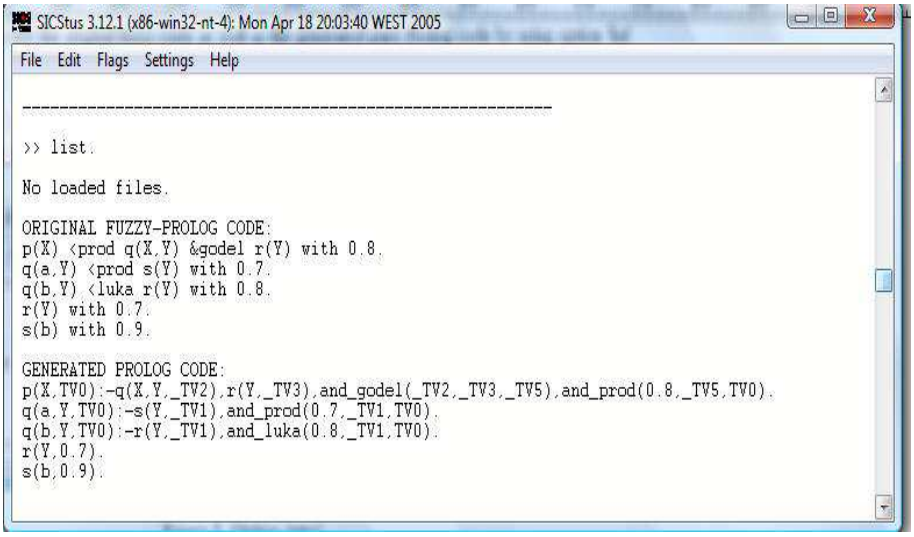
y, una vez que se carga en un intérprete Prolog (en nuestro caso, Sicstus Prolog), muestra un menú que incluye (entre otras) opciones para:



• `Cargar` un archivo prolog con extensión `'.pl'`. Esta acción es útil para leer un archivo conteniendo un conjunto de cláusulas que implementen agregadores, predicados de usuario, etc. Además, las conectivas originales de la *lógica producto*, *lógica de Gödel* y *lógica de Łukasiewicz*, están definidas en el archivo `prelude.pl`, que se carga automáticamente por el sistema al principio de cada sesión de trabajo.

• `Analizar` un programa difuso incluido en un archivo con extensión `'.fpl'`. Es la opción principal encargada de analizar sintácticamente un fichero difuso y cargar el código Prolog resultante en la base de datos del sistema. A fin de llevar a cabo simultáneamente el proceso de traducción con la generación de código, cada predicado *de análisis sintáctico* (*parsing*) descrito por reglas DCG se ha ampliado con una variable como argumento extra que está concebida para incluir el código Prolog generado después de analizar el correspondiente fragmento de código difuso.

• `Listar` el conjunto de cláusulas Prolog cargadas desde un archivo `'.pl'` así como las que se han obtenido después de compilar un archivo `'.fpl'`. Naturalmente, se muestra también el programa difuso original contenido en este último archivo, como se observa en la siguiente figura.



• `Guardar` el código Prolog resultante en un archivo, y finalmente

• `Ejecutar` un objetivo difuso después de ser introducido desde el teclado. Así por ejemplo, si el objetivo proporcionado por el usuario es

`p(X) &G r(a)`,

entonces el sistema lo traduce al objetivo Prolog estándar:

`p(X,_TV1),r(a,_TV2), and_G(_TV1,_TV2,_TV3).`

Sin embargo, esta expresión requiere una manipulación final antes de ser ejecutada, que consiste en renombrar su última variable grado de verdad (en este caso `_TV3`) por `Truth_Degree`. Ahora, adviértase que el conjunto de variables no anónimas del objetivo Prolog resultante, son simplemente aquéllas que aparecían en el objetivo difuso original (es decir, `X`) más la que contiene su grado de verdad asociado (esto es, `Truth_Degree`).

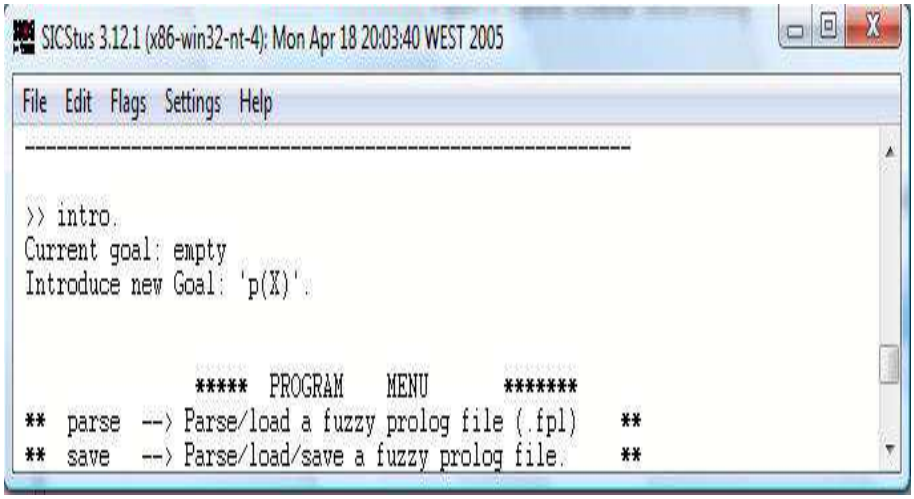
Entonces, después de la reevaluación del objetivo Prolog

`?- p(X,_TV1),r(a,_TV2), and_G(_TV1,_TV2, With_Truth_Degree)`,

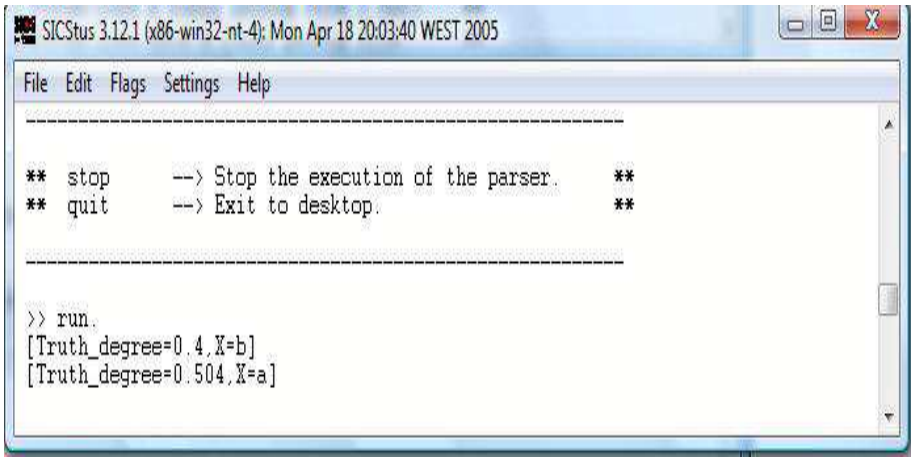
el intérprete Prolog devuelve el siguiente par de respuestas computadas difusas deseadas:

```
X=a, Truth_Degree=0.504;
X=b, Truth_Degree=0.4;
no
```

La siguiente ilustración muestra el menú de introducción de un objetivo:



y en la posterior el resultado de su ejecución.



En consecuencia, nuestra implementación traduce un programa lógico multi-adjunto y un objetivo difuso a código Prolog estándar, permitiendo la ejecución de programas difusos en un entorno de programación lógica clásica. Una explicación pormenorizada de las técnicas de implementación usadas puede encontrarse en el enlace: http://www.dsi.uclm.es/investigacion/dect/FLOPERpage.html.

Es importante tener en cuenta que, en esta implementación, todos los cálculos

internos (incluyendo compilación y ejecución) son derivaciones Prolog puras mientras que las entradas (programas y objetivos difusos) y salidas (respuestas computadas difusas) tienen siempre apariencia difusa, lo que genera la ilusión en el usuario de estar trabajando con una herramienta totalmente difusa.

2.6.2. Traduciendo programas difusos a código Prolog

Esta sección está dedicada a detallar un método sencillo pero potente, para la traducción de programas difusos a código Prolog estándar, en los términos que hemos recogido en [Julián *et al.*, 2006c]. El objetivo final es que el código compilado sea ejecutado en cualquier intérprete Prolog de modo totalmente transparente para el usuario final. Nos proponemos que, una vez introducidos los programas y objetivos difusos al sistema, éste devuelva las respuestas computadas difusas (pares formados por un grado de verdad y una sustitución) incluso en el caso en que los cálculos intermedios se hayan realizado en un contexto lógico puro (no difuso).

En lo que sigue, además de concretar los convenios sintácticos que admite nuestro sistema para traducir a Prolog los programas lógicos multi-adjuntos descubrimos cómo durante el proceso de análisis sintáctico, el sistema genera código Prolog siguiendo estas directrices:

- Cada **átomo** que aparece en una regla difusa se traduce a un átomo Prolog ampliado con un argumento extra, llamado "variable grado de verdad", de la forma `_TVi`. El objetivo pretendido de esta variable anónima es que almacene el grado de verdad resultante después de la subsiguiente evaluación de cada átomo.

- El papel de los **operadores de agregación** puede ser desempeñado adecuadamente por cláusulas Prolog estándar definiendo las funciones de verdad de los agregadores mediante lo que llamamos "predicados de agregación". Por ejemplo:

  $\texttt{and\_prod}(\texttt{X},\texttt{Y},\texttt{Z}) : -\texttt{Z is X} * \texttt{Y}.$

  $\texttt{and\_godel}(\texttt{X},\texttt{Y},\texttt{Z}) : -(\texttt{X} =< \texttt{Y}, \texttt{Z} = \texttt{X}; \texttt{X} > \texttt{Y}, \texttt{Z} = \texttt{Y}).$

  $\texttt{and\_luka}(\texttt{X},\texttt{Y},\texttt{Z}) : -\texttt{H is X} + \texttt{Y} - 1,\ (\texttt{H} =< 0, \texttt{Z} = 0; \texttt{H} > 0, \texttt{Z} = \texttt{H}).$

- Los **hechos del programa difuso** (es decir, reglas sin cuerpo) se expanden en tiempo de compilación a hechos Prolog, donde el argumento adicional del

átomo (de la cabeza) no es una variable grado de verdad sino directamente el grado de verdad de la correspondiente regla. Por ejemplo, considerando de nuevo el programa (que tiene por retículo asociado el intervalo cerrado $([0,1],\leq)$ donde $\leq$ es el orden usual),

$\texttt{p(X)} < \texttt{prod q(X,Y)}\ \ \&\texttt{godel r(Y)}\ \ \texttt{with}\ \ 0.8$

$\texttt{q(a,Y)} < \texttt{prod s(Y)}\ \ \texttt{with}\ \ 0.7$

$\texttt{q(b,Y)} < \texttt{luka r(Y)}\ \ \texttt{with}\ \ 0.8$

$\texttt{r(\_)}\ \ \texttt{with}\ \ 0.7$

$\texttt{s(b)}\ \ \texttt{with}\ \ 0.9$

vemos que las reglas cuatro y cinco pueden representarse por los hechos Prolog $\texttt{r(\_,0.7)}$ y $\texttt{s(b,0.9)}$, respectivamente.

- Las **reglas de programa difuso** se traducen a cláusulas Prolog efectuando llamadas apropiadas a los átomos que forman el cuerpo de éstas. Respecto a las llamadas a los predicados de agregación, deben ser pospuestas al final del cuerpo, a fin de garantizar que las variables de grado de verdad usadas como argumentos se instancien debidamente en el momento apropiado. En este sentido, es también importante respetar el orden adecuado en la ejecución de las llamadas. En particular, la última llamada debe ser efectuada necesariamente al predicado de agregación que modela la conjunción adjunta al operador implicación de la regla, haciendo uso también del grado de verdad de ésta. Por ejemplo, las tres primeras reglas de nuestro ejemplo pueden representarse por las cláusulas Prolog:

  $\texttt{p(X, \_TV0)} :- \texttt{q(X,Y, \_TV1)}, \texttt{r(Y, \_TV2)},\ \texttt{and\_godel(\_TV1, \_TV2, \_TV3)},$

  $\texttt{and\_prod(0.8, \_TV3, \_TV0)}.$

  $\texttt{q(a,Y, \_TV0)} :- \texttt{s(Y, \_TV1)},\ \texttt{and\_prod(0.7, \_TV1, \_TV0)}.$

  $\texttt{q(b,Y, \_TV0)} :- \texttt{r(Y, \_TV1)},\ \texttt{and\_luka(0.8, \_TV1, \_TV0)}.$

- Un **objetivo difuso** se traduce a un objetivo Prolog donde la correspondiente llamada a los átomos aparece en su orden textual antes de las correspondientes a los predicados de agregación. Dado que los agregadores no son necesariamente asociativos, deben aparecer en una secuencia conveniente, al igual que ocurre

con la traducción de los cuerpos de cláusulas descrita anteriormente. Por ejemplo, el objetivo `p(X) &godel r(a)` puede representarse por el siguiente objetivo Prolog:

$?-$ `p(X, _TV1), r(a, _TV2), and_godel(_TV1, _TV2, _TV3).`

Siguiendo este procedimiento, la herramienta FLOPER traduce a código Prolog estándar el programa lógico multi-adjunto y el objetivo considerado anteriormente.

2.6.3. Nuevas utilidades del entorno FLOPER

El método que acabamos de describir es útil para la resolución de objetivos y la ejecución de programas, pero resulta insuficiente cuando se intentan realizar otro tipo de acciones más sofisticadas. En particular, obsérvese que este desarrollo inicial únicamente permite simular derivaciones difusas completas (ejecutando las correspondientes derivaciones Prolog basadas en resolución-SLD), pero no permite generar derivaciones parciales ni siquiera aplicar un sólo paso admisible/interpretativo sobre una expresión difusa determinada.

Y este tipo de manipulaciones de bajo nivel son preceptivas cuando intentamos incorporar a la herramienta mecanismos para generar derivaciones de un número fijo de pasos, reconfigurar el cuerpo de una regla de programa, aplicar sustituciones en su cabeza, etc., a fin de poder implementar algunas técnicas de transformación de programas como las que se contemplan en esta tesis. Por ejemplo (y tal como veremos después), nuestra transformación de desplegado difuso se define como el reemplazamiento de una regla de programa $\mathcal{R} : \langle A \leftarrow_i \mathcal{B}; \alpha \rangle$ por el conjunto de reglas $\{\langle A\sigma \leftarrow_i \mathcal{B}'; \alpha \rangle \mid \langle \mathcal{B}; id \rangle \rightarrow_{AS/IS} \langle \mathcal{B}'; \sigma \rangle\}$, que exige, efectivamente, la implementación de recursos útiles para la generación de derivaciones de un sólo paso, la reconfiguración del cuerpo de una regla, la aplicación de sustituciones en su cabeza, entre otras.

Para alcanzar este objetivo, hemos ideado una nueva representación de bajo nivel para código difuso: cada predicado *de análisis sintáctico* usado en reglas DCG's (que ya contiene un parámetro asignando el código Prolog obtenido después del proceso de compilación) ha sido ampliado con un segundo argumento extra para almacenar ahora la nueva representación asociada al correspondiente fragmento de código difuso analizado. Para no entrar en detalles técnicos acerca de tal representación de bajo nivel, baste el siguiente ejemplo ilustrativo. Recordemos que después de analizar la

siguiente regla:

$$\texttt{p(X)} < \texttt{prod q(X,Y)} \;\; \&\texttt{godel r(Y)} \;\; \texttt{with} \;\; 0.8$$

se generaba la cláusula Prolog:

$$\texttt{p(X, \_TV0)} : -\texttt{q(X,Y, \_TV1)}, \texttt{r(Y, \_TV2)}, \; \texttt{and\_godel(\_TV1, \_TV2, \_TV3)}, \; \texttt{and\_prod(0.8, \_TV3, \_TV0)}$$

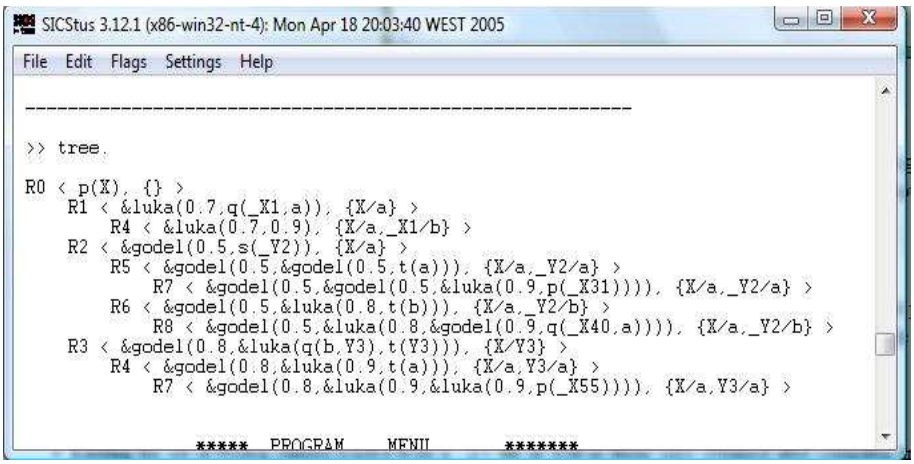
Mientras que ahora además, se genera la siguiente expresión (que es almacenada como un hecho prolog):

```
rule(number(1),
  head(atom(pred(p, 1), var('X')])),
  impl(prod),
  body(and(godel, 2,
          [atom(pred(q, 2),
            [var('X'), var('Y')]),
          atom(pred(r, 1),
            [var('Y')])])),
  td(0.8))
```

Como puede verse, la idea de esta especificación es ir encapsulando cada elemento de una regla difusa en una representación jerarquizada y estructurada en formato de hecho Prolog.

Sobre estos hechos es más sencillo trabajar a bajo nivel, mediante la definición de predicados que realizan desde las operaciones más específicas como son la aplicación de una sustitución, aplicación de un paso admisible, etc., hasta las de mayor calibre, como es el caso de la generación/visualización de un árbol de desplegado para un programa y objetivo dados, permitiendo al usuario que se modifique la profundidad del mismo en cada ejecución, tal y como se observa en la siguiente figura.

Este nuevo recurso de FLOPER nos será de mucha utilidad en el Capítulo 7 ya que la obtención de árboles de desplegado será el punto de partida para especializar/optimizar programas multi-adjuntos y calcular resultantes/reductantes para los mismos.



Como trabajo futuro, en el grupo DEC-TAU, ya estamos abordando las siguientes mejoras de esta herramienta.

- Modelado de la fase interpretativa del lenguaje multi-adjunto, de manera análoga a como hicimos con la fase operacional, abordando la posibilidad de implementar funciones de selección más generales.
- Incorporación de retículos multi-adjuntos arbitrarios y de otras nociones difusas como las ya clásicas relaciones de similaridad.
- Plantear la generación de código para un máquina virtual basada en alguna extensión del diseño de la WAM, que es un estándar para la implementación eficiente de lenguajes de programación lógica.
- Realización de un interfaz gráfico que permite la interacción con el usuario de forma más sencilla y operativa, y que pueda facilitar la implantación de este tipo de tecnología en diferentes contextos (académicos, industriales, etc.).

Para terminar, volvemos a remarcar la intención última de que FLOPER se constituya en la principal plataforma sobre la que implantar nuestros resultados fundamentales, permitiendo (además de programar y desarrollar aplicaciones escritas en lenguajes lógicos difusos) la incorporación de potentes técnicas declarativas de manipulación de programas. En este sentido, insistimos que haciendo uso de la capacidad de FLOPER para generar árboles de desplegado, podremos abordar próximamente buena parte de las técnicas desarrolladas en esta tesis, dando soporte al cálculo de reductantes, la especialización por EP y la optimización de programas difusos usando transformaciones de plegado/desplegado.

2.7. Conclusiones

Las contribuciones referidas en este capítulo pueden resumirse del siguiente modo:

- Hemos realizado una extensión del lenguaje f-Prolog (y también del lf-Prolog) de Vojtáš y Paulík [1996] a fin de permitir una interpretación más flexible de las conectivas, y de forma que se conservan las semánticas operacional y declarativa del lenguaje original.
- Hemos clarificado la fase interpretativa del lenguaje multi-adjunto formulándola en términos de un sistema de transición de estados, lo que será esencial para la definición del desplegado en este contexto y, asimismo, para abordar un análisis del coste computacional de los programas multi-adjuntos.
- Hemos formulado y demostrado la independencia de la regla de computación para los lenguajes f-Prolog, ef-Prolog y para el lenguaje multi-adjunto.
- Hemos implementado, en el grupo DEC-TAU, un prototipo de intérprete/ compilador para traducir programas lógicos multi-adjuntos en código Prolog, que pretende ser una plataforma de gran utilidad para la optimización de programas difusos. Esta herramienta permite, ya en la actualidad, compilar, ejecutar y manipular programas lógicos difusos, así como generar y visualizar árboles de desplegado.

Capítulo 3

Semánticas del lenguaje multi-adjunto

Presentamos en este capítulo una revisión muy completa en la que se abordan y refinan todas las semánticas conocidas al día de hoy para la programación multi-adjunta, y se aportan los resultados que justifican las (mejores) relaciones entre ellas.

Describimos, en primer término, la semántica procedural (operacional e interpretativa), y aportamos, tal como se recoge en [Julián *et al.*, 2006c], una nueva concepción de la fase interpretativa que será preceptiva para la formalización del desplegado interpretativo en el marco multi-adjunto.

Demostramos la independencia de la regla de computación (de la fase operacional), extendiendo a nuestro contexto el resultado bien conocido de la programación lógica clásica por el que, tal como se justifica en [Julián *et al.*, 2005a], a la hora de ejecutar pasos admisibles de computación, no importa la función se selección de subexpresiones que se emplee.

Posteriormente obtenemos, por primera vez, la noción semántica de modelo mínimo difuso concebido éste como el ínfimo de un conjunto de interpretaciones. Veremos que este concepto reproduce la concepción clásica de modelo mínimo de la programación lógica pura, es equivalente a la semántica de punto fijo y también a la semántica procedural entendida como el conjunto de respuestas computadas difusas. Además, veremos que es posible identificar las respuestas correctas mediante el modelo mínimo difuso, de forma más útil que en la programación lógica ordinaria.

Además, incluimos una demostración original de la corrección de la semántica procedural de la programación multi-adjunta. Y si el resultado es una réplica del caso clásico (de la programación lógica pura), la prueba sí contempla novedades muy significativas con respecto a la primitiva.

Por último, nos ocupamos también de la existencia de las distintas semánticas en situaciones más generales que las naturales del marco multi-adjunto, poniendo de manifiesto que hay casos en los que existe la semántica declarativa de modelo mínimo difuso aunque no existan las correspondientes procedural y de punto fijo.

Por otra parte, la extensión más atractiva que cabe plantearse para completar las equivalencias presentadas en este trabajo es la obtención de semánticas por desplegado para el lenguaje lógico multi-adjunto, concretando la equivalencia con las anteriores.

En [Julián *et al.*, 2009] se recogen buena parte de los resultados esenciales de este capítulo.

3.1. Semántica procedural del lenguaje lógico multi-adjunto

La semántica procedural del lenguaje lógico multi-adjunto puede ser concebida como una fase operacional seguida de una interpretativa. Aunque este punto de vista está implícito en [Medina *et al.*, 2001d, 2004], en esta sección establecemos una separación diáfana entre ambas fases. Además, aportamos una nueva definición de fase interpretativa, de corte procedural, que no sólo es útil para clarificar todo el mecanismo computacional, sino que resulta crucial para formalizar el concepto de desplegado interpretativo que estudiaremos en la Sección 4.5.

3.1.1. Fase operacional

Formalizamos aquí los conceptos de paso de computación admisible difuso, derivación admisible difusa y respuesta computada difusa, con ligeras diferencias respecto a las definiciones que aparecen en [Medina *et al.*, 2004].

A tal efecto, recordamos que necesitamos disponer de un lenguaje que sea el resultado de añadir al alfabeto básico (conteniendo variables, constantes, funciones, símbolos de predicado, conectivas y cuantificadores) descrito en la Sección 2.5: *i*) elementos del retículo (grados de verdad) y *ii*) una conjunción adjunta $\&_i$ para cada

implicación $\leftarrow_i$ que figure en el programa lógico multi-adjunto. Denotaremos este lenguaje (como ya se dijo en la Sección 2.5) por $\mathcal{L}$, y las fórmulas de $\mathcal{L}$ las denominaremos $\mathcal{L}$-fórmulas. Además, un $\mathcal{L}$-programa (que también llamaremos programa lógico multi-adjunto o, simplemente, programa) se entenderá un conjunto de pares $\langle \mathcal{L}$-regla, grado de verdad$\rangle$.

El mecanismo operacional que vamos a definir es un proceso de razonamiento que, usando una generalización del *modus ponens*, proporciona una cota inferior del grado de verdad de un objetivo bajo cualquier modelo de un programa.

De manera más precisa, partiendo de un objetivo extendido y aplicando el *modus ponens* sobre un átomo seleccionado A de dicho objetivo y una regla $\langle A' \leftarrow_i \mathcal{B}, v\rangle$ del programa, si existe una sustitución $\theta = mgu(\{A = A'\})$[1], reemplazamos el átomo A por la expresión extendida $(v \&_i \mathcal{B})\theta$. El proceso se repite hasta obtener una fórmula extendida que no contiene átomos. Entonces, el grado de verdad de ésta se obtiene sin más que interpretarla en el retículo asociado. La propiedad adjunta del par $(\leftarrow_i, \&_i)$ garantiza que el grado de verdad resultante es, en efecto, una cota inferior del grado de verdad del objetivo.

La semántica procedural que describiremos aquí no está basada en refutación: está orientada a obtener una cota de la respuesta computada difusa óptima para la pregunta planteada [Medina y Ojeda-Aciego, 2002].

En esencia, en este contexto, una computación puede verse como un proceso con dos fases procedurales: una primera de tipo operacional y la segunda de carácter interpretativo. En otros lenguajes precedentes, como por ejemplo que se contemplan en los trabajos de Vojtáš y Paulík [1996]; Vojtáš [2001], se combinan ambas fases.

En la formalización del concepto de computación admisible difusa, escribiremos $\mathcal{C}[A]$, o más generalmente $\mathcal{C}[A_1, \ldots, A_n]$, para denotar una $\mathcal{L}$-fórmula donde A, o $A_1, \ldots, A_n$ son respectivamente, subexpresiones (por lo general, átomos) que aparecen arbitrariamente en el contexto –posiblemente vacío– $\mathcal{C}[\,]$. Además, la expresión $\mathcal{C}[A/A']$ (y su extensión obvia) significa el reemplazamiento de A por A' en el contexto $\mathcal{C}[\,]$.

Definición 3.1.1. *Sea $\mathcal{Q}$ un objetivo y σ una sustitución, un* estado *es un par $\langle \mathcal{Q}; \sigma\rangle$. Sea $\mathcal{E}$ el conjunto de estados. Dado ahora un programa multi-adjunto $\mathcal{P}$, definimos una* computación admisible difusa *como un sistema de transición de estados, cuya relación de transición $\rightarrow_{AS} \subset (\mathcal{E} \times \mathcal{E})$ es la menor relación que satisface las*

[1] Por $mgu(E)$ denotamos el *unificador más general* de un conjunto de expresiones E (véase la Sección 1.4, donde introducimos este concepto).

siguientes reglas admisibles*:*

Regla 1.

$$\langle \mathcal{Q}[A]; \sigma \rangle \rightarrow_{AS} \langle (\mathcal{Q}[A/v \&_i \mathcal{B}])\theta; \sigma\theta \rangle \ si \begin{cases} (1) & A \text{ es el átomo seleccionado en } \mathcal{Q} \\ (2) & \theta = mgu(\{A' = A\}) \\ (3) & \langle A' \leftarrow_i \mathcal{B}; v \rangle \text{ en } \mathcal{P} \text{ y } \mathcal{B} \text{ no es vacío.} \end{cases}$$

Regla 2.

$$\langle \mathcal{Q}[A]; \sigma \rangle \rightarrow_{AS} \langle (\mathcal{Q}[A/v])\theta; \sigma\theta \rangle \ si \begin{cases} (1) & A \text{ es el átomo seleccionado en } \mathcal{Q} \\ (2) & \theta = mgu(\{A' = A\}) \\ (3) & \langle A' \leftarrow; v \rangle \text{ en } \mathcal{P}. \end{cases}$$

Regla 3.

$$\langle \mathcal{Q}[A]; \sigma \rangle \rightarrow_{AS} \langle (\mathcal{Q}[A/\bot]); \sigma \rangle \ si \begin{cases} (1) & A \text{ es el átomo seleccionado en } \mathcal{Q} \\ (2) & \text{no existe regla en } \mathcal{P} \text{ cuya cabeza unifique con } A. \end{cases}$$

Todos los conceptos usuales de la programación lógica (derivación, paso de derivación, etc.) pueden extenderse a este contexto, al igual que se supone que las fórmulas involucradas en pasos de computación admisibles difusos son renombradas antes de ser usadas.

Obsérvese que la Regla 3 se introduce para contemplar las (posibles) derivaciones admisibles de fallo, lo que supone una novedad importante con respecto a la programación lógica pura en la que no cabe contrapartida al cese de una derivación.

En lo que sigue, usaremos los símbolos $\rightarrow_{AS1}$, $\rightarrow_{AS2}$ y $\rightarrow_{AS3}$ para referirnos explícitamente a la aplicación de cada una de las reglas admisibles. Cuando sea necesario, anotaremos la regla que se use en el paso correspondiente como un superíndice del símbolo $\rightarrow_{AS}$. Usaremos también $\rightarrow_{AS}^{n}$ para denotar una secuencia de n pasos de una computación admisible y $\rightarrow_{AS}^{*}$ para una secuencia arbitraria de cero o más pasos.

Definición 3.1.2. *Sea $\mathcal{P}$ un programa y $\mathcal{Q}$ un objetivo. Una secuencia $\mathcal{E}_0 \rightarrow_{AS} \mathcal{E}_1 \rightarrow^*_{AS} \mathcal{E}_n$ es una derivación admisible de* éxito *si:*

1. *$\mathcal{E}_0 = \langle \mathcal{Q}; id \rangle$, donde id es la sustitución vacía;*
2. *para cada $0 \leq i < n$, $\mathcal{E}_i \rightarrow_{AS} \mathcal{E}_{i+1}$ es un paso de derivación admisible;*
3. *$\mathcal{E}_n = \langle \mathcal{Q}'; \theta' \rangle$ y $\mathcal{Q}'$ es una $\mathcal{L}$-formula que no contiene átomos.*

Tengamos en cuenta que, si después de una secuencia de pasos admisibles, un objetivo se transforma en $\mathcal{L}$-fórmula que no contiene átomos, puede ser interpretada directamente en el retículo multi-adjunto L. Esto justifica la siguiente extensión de la noción de respuesta computada a este contexto. En ella usamos $\mathcal{V}ar(s)$ para referirnos al conjunto de variables distintas que aparecen en el objeto sintáctico s, y $\theta[\mathcal{V}ar(s)]$ denota la sustitución obtenida a partir de θ al restringir su dominio, $Dom(\theta)$, a $\mathcal{V}ar(s)$.

Definición 3.1.3. *Sea $\mathcal{P}$ un programa lógico multi-adjunto y $\mathcal{Q}$ un objetivo. Una* derivación admisible *es una secuencia $\langle \mathcal{Q}; id \rangle \rightarrow^*_{AS} \langle \mathcal{Q}'; \theta \rangle$. Cuando $\mathcal{Q}'$ es una fórmula que no contiene átomos, el par $\langle \mathcal{Q}'; \sigma \rangle$, donde $\sigma = \theta[\mathcal{V}ar(\mathcal{Q})]$, se denomina una* respuesta computada admisible *(a.c.a.,* admissible computed answer*) para esta derivación.*

Definición 3.1.4. *Sea $\mathcal{P}$ un programa multi-adjunto y $\mathcal{Q}$ un objetivo. Dada una derivación admisible $\langle \mathcal{Q}; id \rangle \rightarrow^*_{AS} \langle @(r_1, \ldots, r_n); \theta \rangle$, con $r_i \in L$ para todo $i \in \{1, \ldots, n\}$, el par $\langle \dot{@}(r_1, \ldots, r_n); \sigma \rangle$, donde $\sigma = \theta[\mathcal{V}ar(\mathcal{Q})]$, es una* respuesta computada difusa *(f.c.a.,* fuzzy computed answer*) para esta derivación.*

Ilustramos los conceptos previos y la última definición por medio de un ejemplo.

Ejemplo 3.1.5. *Sea $\mathcal{P}$ el programa lógico multi-adjunto que sigue,*

$$
\begin{array}{lll}
\mathcal{R}_1: & p(X) \leftarrow_{\texttt{prod}} q(X,Y) \wedge_{\texttt{G}} r(Y) & \textit{with } 0.8 \\
\mathcal{R}_2: & q(a,Y) \leftarrow_{\texttt{prod}} s(Y) & \textit{with } 0.7 \\
\mathcal{R}_3: & q(Y,a) \leftarrow_{\texttt{luka}} r(Y) & \textit{with } 0.8 \\
\mathcal{R}_4: & r(Y) \leftarrow & \textit{with } 0.7 \\
\mathcal{R}_5: & s(b) \leftarrow & \textit{with } 0.9
\end{array}
$$

para el que las etiquetas `prod`, `G` *y* `luka` *significan* producto lógico, lógica intuicionista de Gödel *y* lógica de Łukasiewicz *respectivamente. Esto es,* $\dot{\&}_{\texttt{prod}}(x,y) = x \cdot y$, $\dot{\wedge}_{\texttt{G}}(x,y) = min\{x,y\}$, *y* $\dot{\&}_{\texttt{luka}}(x,y) = max\{0, x+y-1\}$.

En la siguiente derivación admisible para el programa $\mathcal{P}$ *y el objetivo* $\leftarrow p(X) \wedge_{\texttt{G}} r(a)$, *subrayamos la expresión seleccionada en cada paso admisible:*

$$\begin{aligned}
\langle \underline{p(X)} \wedge_{\texttt{G}} r(a); id\rangle \rightarrow_{AS1}{}^{\mathcal{R}_1} & \quad \langle (0.8 \&_{\texttt{prod}} (\underline{q(X_1, Y_1)} \wedge_{\texttt{G}} r(Y_1))) \wedge_{\texttt{G}} r(a); \sigma_1 \rangle \\
\rightarrow_{AS1}{}^{\mathcal{R}_2} & \quad \langle (0.8 \&_{\texttt{prod}} ((0.7 \&_{\texttt{prod}} \underline{s(Y_2)}) \wedge_{\texttt{G}} r(Y_2))) \wedge_{\texttt{G}} r(a); \sigma_2 \rangle \\
\rightarrow_{AS2}{}^{\mathcal{R}_5} & \quad \langle (0.8 \&_{\texttt{prod}} ((0.7 \&_{\texttt{prod}} 0.9) \wedge_{\texttt{G}} \underline{r(b)})) \wedge_{\texttt{G}} r(a); \sigma_3 \rangle \\
\rightarrow_{AS2}{}^{\mathcal{R}_4} & \quad \langle (0.8 \&_{\texttt{prod}} ((0.7 \&_{\texttt{prod}} 0.9) \wedge_{\texttt{G}} 0.7)) \wedge_{\texttt{G}} \underline{r(a)}; \sigma_4 \rangle \\
\rightarrow_{AS2}{}^{\mathcal{R}_4} & \quad \langle (0.8 \&_{\texttt{prod}} ((0.7 \&_{\texttt{prod}} 0.9) \wedge_{\texttt{G}} 0.7)) \wedge_{\texttt{G}} 0.7; \sigma_5 \rangle
\end{aligned}$$

donde

$$\begin{aligned}
\sigma_1 &= \{X/X_1\} \\
\sigma_2 &= \{X/a, X_1/a, Y_1/Y_2\} \\
\sigma_3 &= \{X/a, X_1/a, Y_1/b, Y_2/b\} \\
\sigma_4 &= \{X/a, X_1/a, Y_1/b, Y_2/b, Y_3/b\} \\
\sigma_5 &= \{X/a, X_1/a, Y_1/b, Y_2/b, Y_3/b, Y_4/a\}
\end{aligned}$$

Entonces, la respuesta computada admisible para esta derivación admisible es el par $\langle (0.8 \&_{\texttt{prod}} ((0.7 \&_{\texttt{prod}} 0.9) \wedge_{\texttt{G}} 0.7)) \wedge_{\texttt{G}} 0.7; \sigma_5 \rangle$. *Además, la respuesta computada difusa es el par* $\langle 0.504, \{X/a\} \rangle$, *ya que:*

$$\begin{gathered}
\mathcal{I}((0.8 \&_{\texttt{prod}} ((0.7 \&_{\texttt{prod}} 0.9) \wedge_{\texttt{G}} 0.7)) \wedge_{\texttt{G}} 0.7) = \\
\vdots \\
(0.8 \dot{\&}_{\texttt{prod}} ((0.7 \dot{\&}_{\texttt{prod}} 0.9) \dot{\wedge}_{\texttt{G}} 0.7)) \dot{\wedge}_{\texttt{G}} 0.7 = \\
\vdots \\
0.504 \dot{\wedge}_{\texttt{G}} 0.7 = \\
0.504
\end{gathered}$$

y $\sigma_5[\mathcal{V}ar(\mathcal{Q})] = \{X/a\}$.

3.1.2. Fase interpretativa

Si explotamos todos los átomos de un objetivo, aplicando tantos pasos admisibles como sea necesario en la fase operacional, llegamos a una fórmula que no contiene átomos que puede ser directamente interpretada en el correspondiente retículo multi-adjunto L.

En lo que sigue diseñamos, tal como se recoge en [Julián *et al.*, 2006c], esta fase interpretativa en términos de un sistema de transición de estados. Clarificamos de este modo la mencionada fase interpretativa y sentamos las bases para la posterior definición del desplegado interpretativo en términos de pasos interpretativos. Además, hemos logrado desprendernos de elementos "ruidosos" como la función de selección (para esta fase interpretativa) y los correspondientes resultados de independencia.

Veamos, entonces, la noción de paso de computación interpretativo, derivación interpretativa y (recordemos la de) respuesta computada difusa.

Definición 3.1.6 (Paso Interpretativo)**.** *Sea $\mathcal{P}$ un programa, $\mathcal{Q}$ un objetivo y σ una sustitución. Formalizamos la noción de* computación interpretativa *como un sistema de transición de estados, cuya relación de transición $\rightarrow_{IS} \subset (\mathcal{E} \times \mathcal{E})$ se define como*

$$\langle Q[@(r_1, r_2)]; \sigma\rangle \rightarrow_{IS} \langle Q[@(r_1, r_2)/\dot{@}(r_1, r_2)]; \sigma\rangle$$

donde $\dot{@}$ es la función de verdad de la connectiva $@$ en el retículo $(L, \leq)$ asociado al programa $\mathcal{P}$.

Definición 3.1.7. *Sea $\mathcal{P}$ un programa y $\langle Q; \sigma\rangle$ una respuesta computada admisible (a.c.a.), es decir, $\mathcal{Q}$ es un objetivo que no contiene átomos. Una* derivación interpretativa *es una secuencia $\langle Q; \sigma\rangle \rightarrow^*_{IS} \langle Q'; \sigma\rangle$.*

Recordemos que si $Q' = r \in L$, siendo $(L, \leq)$ el retículo asociado a $\mathcal{P}$, el estado $\langle r; \sigma\rangle$ se dice[2] una *respuesta computada difusa* o f.c.a.

Habitualmente, nos referiremos a una *derivación completa* como la secuencia de pasos admisibles/interpretativos de la forma $\langle Q; id\rangle \rightarrow^*_{AS} \langle \mathcal{Q}'; \sigma\rangle \rightarrow^*_{IS} \langle r; \sigma\rangle$, donde $\langle \mathcal{Q}'; \sigma[\mathcal{V}ar(\mathcal{Q})]\rangle$ y $\langle r; \sigma[\mathcal{V}ar(\mathcal{Q})]\rangle$ son, respectivamente, la a.c.a. y la f.c.a. para la derivación. En ocasiones, la denotaremos por $\langle Q; id\rangle \rightarrow^*_{AS/IS} \langle r; \sigma\rangle$ (y diremos que $\langle r; \sigma\rangle$ es la respuesta computada difusa de la correspondiente derivación en la que se agrupan los pasos admisibles e interpretativos).

[2] Véase la Definición 3.1.4.

Ejemplo 3.1.8. *Completamos ahora la derivación previa del Ejemplo 3.1.5, ejecutando los pasos interpretativos necesarios para obtener la respuesta computada difusa (f.c.a.) con respecto al retículo* $([0,1], \leq)$.

$$\langle((0.8\&_{\mathtt{prod}}(\underline{(0.7\&_{\mathtt{prod}}0.9)}\&_{\mathtt{G}}0.7))\&_{\mathtt{G}}0.7; \{X/a\}\rangle \rightarrow_{IS}$$

$$\langle((0.8\&_{\mathtt{prod}}(\underline{0.63\&_{\mathtt{G}}0.7}))\&_{\mathtt{G}}0.7; \{X/a\}\rangle \rightarrow_{IS}$$

$$\langle(\underline{(0.8\&_{\mathtt{prod}}0.63)}\&_{\mathtt{G}}0.7; \{X/a\}\rangle \rightarrow_{IS}$$

$$\langle\underline{0.504\&_{\mathtt{G}}0.7}; \{X/a\}\rangle \rightarrow_{IS}$$

$$\langle 0.504; \{X/a\}\rangle$$

Entonces la f.c.a. para esta derivación completa es el par $\langle 0.504; \{X/a\}\rangle$.

3.1.3. Independencia de la regla de computación para el lenguaje multi-adjunto

En [Julián *et al.*, 2005a] se enuncia y demuestra por primera vez la independencia de la regla de computación para la programación lógica multi-adjunta, resultado que en un contexto mucho más general corresponde a los Teoremas 2.4.3 y 2.4.2 del capítulo anterior.

Como para la resolución-SLD, suponemos la existencia de una función de selección fija, también llamada *regla de computación*, que decide, sobre un objetivo dado, cuál es la expresión seleccionada para explotar en el siguiente paso admisible difuso. Dada un regla de computación difusa $\mathcal{S}$, decimos que una derivación admisible difusa es *vía* $\mathcal{S}$ si la expresión seleccionada en todo paso admisible se obtiene por la imagen de la aplicación $\mathcal{S}$ sobre el correspondiente objetivo en cada paso.

Establecemos, para el lenguaje multi-adjunto, la independencia de la regla de computación de manera análoga a la obtenida en [Lloyd, 1987] para el caso de la programación lógica pura. Para lograr este resultado necesitamos los siguientes lemas auxiliares. El primero garantiza que se preservan las sustituciones en respuestas computadas difusas obtenidas por la aplicación de dos pasos admisibles dados con la primera y/o la segunda regla de la Definición 3.1.1, con independencia del orden en que se explotan los átomos.

Lema 3.1.9. *Sea $\mathcal{P}$ un programa multi-adjunto y $\mathcal{Q}_0[A, A']$ un objetivo. Si en las siguientes derivaciones consideramos sólo pasos del tipo $AS1$ y $AS2$, entonces,*

$$\langle \mathcal{Q}_0; \theta_0 \rangle \rightarrow_{AS}{}^{A} \langle \mathcal{Q}_1; \theta_0\theta_1 \rangle \rightarrow_{AS}{}^{A'\theta_1} \langle \mathcal{Q}_2; \theta_0\theta_1\theta_2 \rangle$$

si, y sólo si,

$$\langle \mathcal{Q}_0; \theta_0 \rangle \rightarrow_{AS}{}^{A'} \langle \mathcal{Q}'_1; \theta_0\theta'_1 \rangle \rightarrow_{AS}{}^{A\theta'_1} \langle \mathcal{Q}'_2; \theta_0\theta'_1\theta'_2 \rangle$$

donde $\theta_0\theta_1\theta_2 = \theta_0\theta'_1\theta'_2$.

Demostración. ($\Rightarrow$) Sean H_1 y H_2 los átomos cabeza de las reglas $\mathcal{R}_1, \mathcal{R}_2 \ll \mathcal{P}$ usadas para explotar (instancias de) los átomos A y A', respectivamente, en la derivación considerada: $\langle \mathcal{Q}_0; \theta_0 \rangle \rightarrow_{AS}{}^{A} \langle \mathcal{Q}_1; \theta_0\theta_1 \rangle \rightarrow_{AS}{}^{A'\theta_1} \langle \mathcal{Q}_2; \theta_0\theta_1\theta_2 \rangle$, donde $\theta_1 = mgu(\{A = H_1\})$ y $\theta_2 = mgu(\{A'\theta_1 = H_2\})$. Además, puesto que no consideramos pasos del tipo $AS3$, entonces $\theta_0\theta_1\theta_2 \neq fallo$, y en particular $\theta_2 \neq fallo$, por lo que también se tiene $\theta'_1 = mgu(\{A' = H_2\}) \neq fallo$. Por tanto, se satisfacen las siguientes igualdades:

$$\begin{array}{ll}
\theta_1\theta_2 & = \\
\theta_1 mgu(\{A'\theta_1 = H_2\}) & = (\text{ya que } Dom(\theta_1) \cap \mathcal{V}ar(\mathcal{R}_2) = \varnothing) \\
\theta_1 mgu(\widehat{mgu}(\{A' = H_2\})\theta_1) & = \\
\theta_1 mgu(\widehat{\theta'_1}\theta_1) & = (\text{por el Lema 2.4.1}) \\
\theta_1 \Uparrow \theta'_1 & = (\text{por el Lema 2.4.1}) \\
\theta'_1 mgu(\widehat{\theta_1}\theta'_1) & = \\
\theta'_1 mgu(\widehat{mgu}(\{A = H_1\})\theta'_1) & = (\text{ya que } Dom(\theta'_1) \cap \mathcal{V}ar(\mathcal{R}_1) = \varnothing) \\
\theta'_1 mgu(\{A\theta'_1 = H_1\}) &
\end{array}$$

Además, como $\theta_1\theta_2 \neq fallo$ se tiene $\theta'_1 mgu(\{A\theta'_1 = H_1\}) \neq fallo$ y, en particular, $\theta'_2 = mgu(\{A\theta'_1 = H_1\}) \neq fallo$. Por tanto, $\theta_1\theta_2 = \theta'_1\theta'_2$ de donde $\theta_0\theta_1\theta_2 = \theta_0\theta'_1\theta'_2$, como queríamos.
($\Leftarrow$) El recíproco de la implicación anterior puede probarse de manera análoga, explotando la equivalencia entre $\theta_1\theta_2$ y $\theta'_1\theta'_2$. □

El siguiente resultado generaliza el Lema 3.1.9 en dos aspectos diferentes:

- preserva las dos componentes del estado de derivación, esto es, no sólo la sustitución sino también los grados de verdad parcialmente evaluados; y
- considera el conjunto completo de reglas admisibles de la Definición 3.1.1 (en lugar de sólo las dos primeras) cuando aplicamos los dos pasos admisibles en las derivaciones consideradas.

Lema 3.1.10 (Lema de Conmutación)**.** *Sea $\mathcal{P}$ un programa y $\mathcal{Q}_0[A, A']$ un objetivo. Entonces,*

$$\langle \mathcal{Q}_0; \theta_0 \rangle \rightarrow_{AS}{}^{A} \langle \mathcal{Q}_1; \theta_1 \rangle \rightarrow_{AS}{}^{A'\theta_1} \langle \mathcal{Q}_2; \theta_2 \rangle$$

si, y sólo si,

$$\langle \mathcal{Q}_0; \theta_0 \rangle \rightarrow_{AS}{}^{A'} \langle \mathcal{Q}'_1; \theta'_1 \rangle \rightarrow_{AS}{}^{A\theta'_1} \langle \mathcal{Q}'_2; \theta'_2 \rangle$$

donde $\mathcal{Q}_2 = \mathcal{Q}'_2$ y $\theta_2 = \theta'_2$.

Demostración. Para una mejor legibilidad, subrayamos los átomos seleccionados que son explotados en cada paso de derivación. En la demostración, puede procederse exaustivamente con cada uno de los posibles casos que se presentan. Afortunadamente, el caso donde el primer paso se da con la regla $\mathcal{R}_i$ y el segundo con la regla $\mathcal{R}_j$ es análogo al caso en el que el primer paso se da con la regla $\mathcal{R}_j$ y el segundo con la regla $\mathcal{R}_i$, lo que reduce significativamente el número de opciones a detallar.

1. Primer paso con la Regla 1 y segundo paso con la Regla 1.
   Suponemos que $\mathcal{R}_1 : \langle H_1 \leftarrow_1 \mathcal{B}_1; v_1 \rangle \ll \mathcal{P}$, $\mathcal{R}_2 : \langle H_2 \leftarrow_2 \mathcal{B}_2; v_2 \rangle \ll \mathcal{P}$. Entonces,

$$\begin{aligned}
&\langle \mathcal{Q}_0[\underline{A}, A']; \theta_0 \rangle && \rightarrow_{AS1}{}^{\mathcal{R}_1} \\
&\langle \mathcal{Q}_0[A/(v_1 \&_1 \mathcal{B}_1), \underline{A'}]\theta_1; \theta_1 \rangle && \rightarrow_{AS1}{}^{\mathcal{R}_2} \\
&\langle \mathcal{Q}_0[A/(v_1 \&_1 \mathcal{B}_1), A'/(v_2 \&_2 \mathcal{B}_2)]\theta_2; \theta_2 \rangle &&
\end{aligned}$$

   si, y sólo si,

$$\begin{aligned}
&\langle \mathcal{Q}_0[A, \underline{A'}]; \theta_0 \rangle && \rightarrow_{AS1}{}^{\mathcal{R}_2} \\
&\langle \mathcal{Q}_0[\underline{A}, A'/(v_2 \&_2 \mathcal{B}_2)]\theta'_1; \theta'_1 \rangle && \rightarrow_{AS1}{}^{\mathcal{R}_1} \\
&\langle \mathcal{Q}_0[A/(v_1 \&_1 \mathcal{B}_1), A'/(v_2 \&_2 \mathcal{B}_2)]\theta'_2; \theta'_2 \rangle &&
\end{aligned}$$

Por el Lema 3.1.9, $\theta_2 = \theta_2'$, lo que garantiza que las primeras componentes del estado final en ambas derivaciones son sintácticamente idénticas, como queríamos demostrar.

2. Primer paso con la Regla 1 y segundo paso con la Regla 2.
Suponemos ahora que $\mathcal{R}_1 : \langle H_1 \leftarrow_1 \mathcal{B}_1; v_1\rangle \ll \mathcal{P}$, $\mathcal{R}_2 : \langle H_2 \leftarrow; v_2\rangle \ll \mathcal{P}$. El lector puede comprobar que este caso es totalmente análogo al anterior, usando ahora ${\rightarrow_{AS2}}^{\mathcal{R}_2}$ en lugar de ${\rightarrow_{AS1}}^{\mathcal{R}_2}$ a fin de que, en vez de la expresión $A'/(v_2 \&_2 \mathcal{B}_2)$, tengamos ahora A'/v_2.

3. Primer paso con la Regla 1 y segundo paso con la Regla 3.
En este caso, suponemos que el átomo A unifica con la cabeza de una regla $\mathcal{R}_1 : \langle H_1 \leftarrow_1 \mathcal{B}_1; v_1\rangle \ll \mathcal{P}$, mientras que no existe ninguna regla en el programa cuya cabeza unifique con (una instancia de) el átomo A'. Es importante observar que, a diferencia de los casos previos, en las derivaciones consideradas se computará sólo un unificador (en lugar de dos). Entonces,

$$\begin{aligned}
&\langle \mathcal{Q}_0[\underline{A}, A']; \theta_0\rangle && {\rightarrow_{AS1}}^{\mathcal{R}_1}\\
&\langle \mathcal{Q}_0[A/(v_1 \&_1 \mathcal{B}_1), \underline{A'}]\theta_1; \theta_1\rangle && \rightarrow_{AS3}\\
&\langle \mathcal{Q}_0[A/(v_1 \&_1 \mathcal{B}_1), A'/\bot]\theta_1; \theta_1\rangle
\end{aligned}$$

si, y sólo si,

$$\begin{aligned}
&\langle \mathcal{Q}_0[A, \underline{A'}]; \theta_0\rangle && \rightarrow_{AS3}\\
&\langle \mathcal{Q}_0[\underline{A}, A'/\bot]; \theta_0\rangle && {\rightarrow_{AS1}}^{\mathcal{R}_1}\\
&\langle \mathcal{Q}_0[A/(v_1 \&_1 \mathcal{B}_1), A'/\bot]\theta_1; \theta_1\rangle
\end{aligned}$$

como queríamos demostrar.

4. Primer paso con la Regla 2 y segundo paso con la Regla 2.
Este caso es semejante al primero, teniendo en cuenta que ahora todos los pasos admisibles son del tipo $\rightarrow_{AS2}$ en vez de $\rightarrow_{AS1}$, y en lugar de las expresiones $A/(v_1 \&_1 \mathcal{B}_1)$ y $A'/(v_2 \&_2 \mathcal{B}_2)$, de los objetivos considerados, tendríamos ahora las expresiones A/v_1 y A'/v_2, respectivamente.

5. Primer paso con la Regla 2 y segundo paso con la Regla 3.
   Omitimos también la prueba de esta caso, dadas las semejanzas con la del tercero.

6. Primer paso con la Regla 3 y segundo paso con la Regla 3.
   En este caso final, no se computan unificadores (lo que contrasta fuertemente con cualquier otro caso), resultando:

$$\langle \mathcal{Q}_0[\underline{A}, A']; \theta_0 \rangle \quad \rightarrow_{AS3}$$

$$\langle \mathcal{Q}_0[A/\bot, \underline{A'}]; \theta_0 \rangle \quad \rightarrow_{AS3}$$

$$\langle \mathcal{Q}_0[A/\bot, A'/\bot]; \theta_0 \rangle$$

   si, y sólo si,

$$\langle \mathcal{Q}_0[A, \underline{A'}]; \theta_0 \rangle \quad \rightarrow_{AS3}$$

$$\langle \mathcal{Q}_0[\underline{A}, A'/\bot]; \theta_0 \rangle \quad \rightarrow_{AS3}$$

$$\langle \mathcal{Q}_0[A/\bot, A'/\bot]; \theta_0 \rangle$$

   como queríamos demostrar.

□

Entonces, podemos formalizar y demostrar el principal resultado de esta sección.

Teorema 3.1.11 (Independencia de la Regla de Computación)**.** *Sea $\mathcal{P}$ un programa y $\mathcal{Q}$ un objetivo. Para cualquier par de reglas de computación $\mathcal{S}$ y $\mathcal{S}'$, se tiene:*

$$\langle \mathcal{Q}; id \rangle \rightarrow_{AS}{}^{k}_{\mathcal{S}} \langle @(r_1, \ldots, r_n); \theta \rangle$$

si, y sólo si,

$$\langle \mathcal{Q}; id \rangle \rightarrow_{AS}{}^{k}_{\mathcal{S}'} \langle @(r_1, \ldots, r_n); \theta \rangle$$

donde $r_i \in L$ para todo $i \in \{1, \ldots, n\}$ y k es el (mismo) número de pasos admisibles difusos en ambas derivaciones.

Demostración. Se obtiene sin dificultad al aplicar repetidamente el Lema de Conmutación 3.1.10. □

3.2. Semántica de punto fijo

En esta sección, introducimos formalmente las nociones semánticas de interpretación, de modelo y de operador $T_{\mathcal{P}}$ de punto fijo para un programa $\mathcal{P}$ lógico multi-adjunto, de manera semejante a como han sido contemplados en [Medina *et al.*, 2004]. Además, sobre algunos ejemplos concretamos los puntos fijos de este operador.

Definición 3.2.1. *Una interpretación*[3] *es una aplicación* $\mathcal{I} : B_{\mathcal{P}} \longrightarrow L$, *donde* $B_{\mathcal{P}}$ *es la base de Herbrand del programa* $\mathcal{P}$ *y* $(L, \leq)$ *es el retículo multi-adjunto asociado a dicho programa.*

$\mathcal{I}$ se extiende de manera natural al conjunto de fórmulas básicas del lenguaje. Para interpretar una fórmula A no básica (cerrada, y universalmente cuantificada en el caso del lenguaje multi-adjunto), basta tomar

$$\mathcal{I}(A) = inf\{\mathcal{I}(A\xi) \mid A\xi \text{ es una instancia básica de A}\}$$

Sea $\mathcal{H}$ el conjunto de interpretaciones dotado del orden inducido por el orden del conjunto L:

$$\mathcal{I}_j \leq \mathcal{I}_k \Longleftrightarrow \mathcal{I}_j(F) \leq \mathcal{I}_k(F), \forall F \in B_{\mathcal{P}}$$

Es trivial comprobar que $(\mathcal{H}, \leq)$ hereda la estructura de retículo completo del retículo $(L, \leq)$.

Definición 3.2.2. *Una interpretación* $\mathcal{I}$ *satisface una regla* $\langle A \leftarrow_i \mathcal{B}; v\rangle$ *si, y sólo si,* $v \leq \mathcal{I}(A \leftarrow_i \mathcal{B})$. *Una interpretación* $\mathcal{I}$ *es un modelo*[4] *de* $\mathcal{P}$ *si, y sólo si,* $\mathcal{I}$ *satisface todas las reglas de* $\mathcal{P}$.

Por otra parte, en un retículo completo $(L, \leq)$, una función $f : L \longrightarrow L$ es monótona si, y sólo si, $\forall x, y \in L, x \leq y \Longrightarrow f(x) \leq f(y)$. Un punto fijo de f es un elemento $x \in L$ tal que $f(x) = x$. El resultado básico[5] para el estudio de puntos fijos de funciones sobre retículos es el siguiente teorema de Knaster-Tarski (véase [Tarski, 1955]).

Teorema 3.2.3. *Sea* f *una función monótona sobre un retículo completo* $(L, \leq)$. *Entonces,* f *tiene un punto fijo.*

[3] Nosotros diremos también interpretación de Herbrand difusa.
[4] Nosotros diremos también modelo de Herbrand difuso.
[5] Otros teoremas de punto fijo pueden verse en [Khamsi y Misane, 1997; Stouti, 2004].

Además, el conjunto de puntos fijos de f es un retículo completo, por lo que tiene sentido tomar el menor punto fijo de f. Este menor punto fijo puede obtenerse iterando f sobre el elemento $\bot \in L$, es decir, es el supremo de la sucesión no decreciente $x_0, \ldots, x_i, x_{i+1}, \ldots, x_\lambda, \ldots$ verificando que para cualquier $i \geq 0, x_0 = \bot$, $x_{i+1} = f(x_i)$, mientras que para un cierto índice λ, $y_\lambda = sup\{y_i : f(y_i) = y_i, i > \lambda\}$.

En [Medina *et al.*, 2001d, 2004] se extiende el operador $T_\mathcal{P}$, definido por van Emden y Kowalski [1976], para el lenguaje multi-adjunto, en los términos que recoge la siguiente definición. Los autores aportan con este operador[6] la semántica de un programa multi-adjunto $\mathcal{P}$ que toman como el menor punto fijo de $T_\mathcal{P}$. Por nuestra parte, justificaremos en el Teorema 3.4.1 la equivalencia entre esta construcción y nuestra noción de modelo mínimo de Herbrand difuso.

Definición 3.2.4. *Sea $\mathcal{P}$ un programa lógico multi-adjunto, $\mathcal{I}$ una interpretación y A una fórmula básica. Definimos el operador $T_\mathcal{P}$ como una aplicación en el conjunto de interpretaciones tal que para cada átomo básico A*

$$T_\mathcal{P}(\mathcal{I})(A) = sup\{v \dot{\&}_i \mathcal{I}(\mathcal{B}\theta) \mid \langle H \leftarrow_i \mathcal{B}; v\rangle \in \mathcal{P}, A = H\theta\}$$

También en [Medina *et al.*, 2004] se demuestra el siguiente resultado por el que $T_\mathcal{P}$ permite caracterizar las interpretaciones que son un modelo de $\mathcal{P}$.

Teorema 3.2.5. *Una interpretación $\mathcal{I}$ es un modelo de un programa multi-adjunto $\mathcal{P}$ si, y sólo si, $T_\mathcal{P}(\mathcal{I}) \leq \mathcal{I}$.*

Por tanto, para todo modelo $\mathcal{I}$ de $\mathcal{P}$ tenemos

$$T_\mathcal{P}(\mathcal{I})(A) = sup\{v \dot{\&}_i \mathcal{I}(\mathcal{B}\theta) \mid \langle H \leftarrow_i \mathcal{B}; v\rangle \in \mathcal{P}, A = H\theta\} \leq \mathcal{I}(A)$$

Además, en general, la igualdad $T_\mathcal{P}(\mathcal{I})(A) = \mathcal{I}(A)$ no se alcanza. Este hecho es análogo al que se obtiene en el caso (discreto) de la programación lógica pura: en líneas generales, se puede asegurar que "cuanto más se evalúe/interprete (itere el operador $T_\mathcal{P}$ sobre) un átomo, tanto más puede decrecer su grado de verdad".

Por otra parte, tanto en la programación lógica pura como en la programación lógica multi-adjunta puede existir más de un punto fijo. El siguiente ejemplo ilustra este hecho para el caso clásico.

[6] Según esta definición de $T_\mathcal{P}$, es sencillo comprobar que la interpretación de un átomo básico A, por un modelo $\mathcal{I}$ de $\mathcal{P}$, es mayor o igual que la interpretación del cuerpo del PE^1-reductante, que introducimos en el Capítulo 7, correspondiente a este átomo.

Ejemplo 3.2.6. *Sea $\mathcal{P}$ el programa lógico formado por la cláusula $p(a)\leftarrow p(a)$. La base de Herbrand de $\mathcal{P}$ es el conjunto $\mathcal{B}_H = \{p(a)\}$ y los subconjuntos de la base de Herbrand $\mathcal{I}_1 = \varnothing, \mathcal{I}_2 = \{p(a)\}$ son los únicos modelos de $\mathcal{P}$. Ambos son puntos fijos del operador (de punto fijo) definido en [Lloyd, 1987].*

En el contexto multi-adjunto podríamos tomar el mismo ejemplo, toda vez que todo programa lógico definido puede ser expresado como un programa multi-adjunto. Con la intención de considerar un caso más específico aportamos también el siguiente ejemplo.

Ejemplo 3.2.7. *Considérese $\mathcal{P}$ el programa lógico multi-adjunto formado por la única regla $\langle p(a)\leftarrow p(a); 0.5\rangle$ y con retículo asociado el intervalo $([0,1],\leq)$. Sea $(\&_{\mathtt{G}}, \leftarrow_{\mathtt{G}})$ el par adjunto en $([0,1],\leq)$ de la lógica de* Gödel*, con funciones de verdad de $\&_{\mathtt{G}}$ y $\leftarrow_{\mathtt{G}}$ dadas por:*

$$\dot{\&}_{\mathtt{G}}(x,y) = inf\{x,y\} \qquad y \qquad x \dot{\rightarrow}_{\mathtt{G}} y = \begin{cases} \top, & si\ x \leq y \\ y, & en\ otro\ caso \end{cases}$$

Entonces, las interpretaciones $\mathcal{I}_1, \mathcal{I}_2$ definidas por $\mathcal{I}_1(p(a)) = 0, \mathcal{I}_2(p(a)) = 0.5$ verifican:

$$T_{\mathcal{P}}(\mathcal{I}_1)(p(a)) = sup\{0.5\dot{\&}\mathcal{I}_1(p(a))\} = sup\{0.5\dot{\&}0\} = 0 = \mathcal{I}_1(p(a))$$

$$T_{\mathcal{P}}(\mathcal{I}_2)(p(a)) = sup\{0.5\dot{\&}\mathcal{I}_2(p(a))\} = sup\{0.5\dot{\&}0.5\} = 0.5 = \mathcal{I}_2(p(a))$$

y, por tanto, ambos son puntos fijos del operador $T_{\mathcal{P}}$, resultando $\mathcal{I}_1$ el menor punto fijo.

3.3. Semántica declarativa por modelo mínimo de Herbrand difuso

Nos proponemos definir, por primera vez en la literatura, una semántica declarativa para la programación lógica multi-adjunta en términos del modelo mínimo difuso (y tal como se ha recogido en [Julián *et al.*, 2009]). Esta construcción reproduce, en el contexto difuso citado, la contrucción clásica de modelo mínimo de Herbrand que se puede encontrar en [Lloyd, 1987] para la programación lógica pura, que ha sido aceptada tradicionalmente como la semántica declarativa de programas lógicos.

En los últimos años, se han realizado algunas adaptaciones de este concepto, usando teoría de modelos ([Vojtáš y Paulík, 1996; Sessa, 2002; Straccia *et al.*, 2009]),

en el área de la programación lógica difusa, aunque no para la programación lógica multi-adjunta. Para evitar esta laguna, en lo que sigue nos proponemos conformar una semántica declarativa, basada en lo que llamaremos modelo mínimo de Herbrand difuso, para este tipo de programas.

Además, relacionaremos en las secciones posteriores nuestra noción de modelo mínimo difuso con la ya existente semántica procedural y de punto fijo a la vez que aportamos ejemplos reveladores en los que nuestra semántica declarativa sigue teniendo sentido más allá del marco multi-adjunto mientras las anteriormente mencionadas no están definidas.

Definición 3.3.1. *Sea $\mathcal{P}$ un programa lógico multi-adjunto con retículo asociado $(L;\leq)$. La interpretación $\mathcal{I}_{\mathcal{P}} = inf\{\mathcal{I}_j : \mathcal{I}_j$ es un modelo de $\mathcal{P}\}$ se dice el modelo mínimo de Herbrand difuso[7] de $\mathcal{P}$.*

El siguiente resultado justifica que la interpretación $\mathcal{I}_{\mathcal{P}}$ anterior pueda ser tomada como el modelo mínimo de Herbrand difuso.

Teorema 3.3.2. *Sea $\mathcal{P}$ un programa lógico multi-adjunto con retículo asociado L. La aplicación $\mathcal{I}_{\mathcal{P}} = inf\{\mathcal{I}_j : \mathcal{I}_j$ es un modelo de $\mathcal{P}\}$ es el menor modelo de $\mathcal{P}$.*

Demostración. En lo que sigue, denotamos por $\mathcal{K}$ el conjunto de interpretaciones que son modelos de $\mathcal{P}$, es decir, $\mathcal{K} = \{\mathcal{I}_j : \mathcal{I}_j$ es un modelo de $\mathcal{P}\}$.

Entonces, como $\mathcal{I}_{\mathcal{P}} = inf(\mathcal{K})$, $\mathcal{I}_{\mathcal{P}}$ es una interpretación de Herbrand: por ser el conjunto de interpretraciones $(\mathcal{H},\leq)$ un retículo completo, existe el ínfimo del subconjunto $\mathcal{K}$ y es un elemento de $\mathcal{H}$.

Veamos, ahora, que $\mathcal{I}_{\mathcal{P}}$ es un modelo de $\mathcal{P}$, es decir que satisface todas las reglas del programa.

En efecto, sea $\mathcal{R} : \langle A \leftarrow_i \mathcal{B}; v\rangle$ una regla cualquiera de $\mathcal{P}$. Puesto que $\mathcal{I}_{\mathcal{P}}$ es el ínfimo de $\mathcal{K}$, se tiene que $\mathcal{I}_{\mathcal{P}} \leq \mathcal{I}_j$, para cada modelo $\mathcal{I}_j$ de $\mathcal{P}$. Por tanto, $\mathcal{I}_{\mathcal{P}}(A) \leq \mathcal{I}_j(A)$ para cada átomo A. Por otra parte, dado que $\mathcal{I}_j$ es un modelo de $\mathcal{P}$, por definición de modelo, $\mathcal{I}_j$ satisface la regla $\mathcal{R}$, es decir, $v \leq \mathcal{I}_j(A \leftarrow_i \mathcal{B})$.

Haciendo uso de la definición de interpretación, la monotonía[8] de $\dot{\&}_i$ y de $\dot{\leftarrow}_i$, y el resultado $\mathcal{I}_{\mathcal{P}}(A) \leq \mathcal{I}_j(A)$ anterior, se tiene:

$$v \leq \mathcal{I}_j(A \leftarrow_i \mathcal{B}) = \mathcal{I}_j(A) \dot{\leftarrow}_i \mathcal{I}_j(\mathcal{B}) \leq \mathcal{I}_{\mathcal{P}}(A) \dot{\leftarrow}_i \mathcal{I}_j(\mathcal{B})$$

[7] En ocasiones diremos sólo modelo mínimo difuso o modelo mínimo.
[8] Véase la Definición 2.5.1 de par adjunto.

Por la propiedad adjunta, $v \leq \mathcal{I}_{\mathcal{P}}(A) \dot{\leftarrow}_i \mathcal{I}_j(\mathcal{B})$ si, y sólo si, $v \dot{\&}_i \mathcal{I}_j(\mathcal{B}) \leq \mathcal{I}_{\mathcal{P}}(A)$. Entonces, como la función $\dot{\&}_i$ es creciente en ambos argumentos e $\mathcal{I}_{\mathcal{P}}(\mathcal{B}) \leq \mathcal{I}_j(\mathcal{B})$, se tiene $v \dot{\&}_i \mathcal{I}_{\mathcal{P}}(\mathcal{B}) \leq \mathcal{I}_{\mathcal{P}}(A)$.

Ahora, aplicando de nuevo la propiedad adjunta, $v \dot{\&}_i \mathcal{I}_{\mathcal{P}}(\mathcal{B}) \leq \mathcal{I}_{\mathcal{P}}(A)$ si, y sólo si, $v \leq \mathcal{I}_{\mathcal{P}}(A) \dot{\leftarrow}_i \mathcal{I}_{\mathcal{P}}(\mathcal{B}) = \mathcal{I}_{\mathcal{P}}(A \leftarrow_i \mathcal{B})$.

En consecuencia, $\mathcal{I}_{\mathcal{P}}$ satisface la regla $\mathcal{R}$ considerada y es un modelo de $\mathcal{P}$, como queríamos.

Finalmente, como $\mathcal{I}_{\mathcal{P}} = inf(\mathcal{K})$, por la definición de ínfimo $\mathcal{I}_{\mathcal{P}} \leq \mathcal{I}_j, \forall j$ por lo que $\mathcal{I}_{\mathcal{P}}$ es el menor modelo de $\mathcal{P}$, lo que termina la prueba. □

En esta demostración, resulta esencial que el retículo asociado al programa sea un retículo multi-adjunto. En el ejemplo que sigue se muestra la necesidad de esta hipótesis para el Teorema 3.3.2.

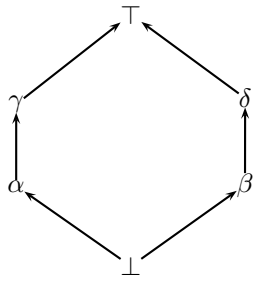
Ejemplo 3.3.3. *Si* $(\&_{\mathtt{G}}, \leftarrow_{\mathtt{G}})$ *es, de nuevo, el par de conectivas cuyas funciones de verdad están definidas por*

$$\dot{\&}_{\mathtt{G}}(x,y) = inf\{x,y\} \qquad y \qquad x \dot{\rightarrow}_{\mathtt{G}} y = \begin{cases} \top, & si\ x \leq y \\ y, & en\ otro\ caso \end{cases}$$

no forman un par adjunto en el retículo que se muestra en la figura posterior puesto que, verificándose $\alpha \dot{\&} \delta \leq \beta$, *no se cumple* $\alpha \leq \beta \dot{\leftarrow} \delta$.

$\mathcal{R}_1 : \quad \langle p \quad \leftarrow_{\mathtt{G}} q; \quad \alpha \rangle$

$\mathcal{R}_2 : \quad \langle q \quad \leftarrow_{\mathtt{G}} ; \quad \beta \rangle$



	$\mathcal{I}_1$	$\mathcal{I}_2$	$\mathcal{I}_3$	$\mathcal{I}_4$	$\mathcal{I}_5$	$\mathcal{I}_6$	$\mathcal{I}_7$	$\mathcal{I}_8$	$\mathcal{I}_9$	$\mathcal{I}_{10}$	$\mathcal{I}_{11}$	$\mathcal{I}_{12}$
p	β	δ	$\top$	α	γ	δ	$\top$	α	γ	$\top$	α	γ
q	β	β	β	β	β	δ	δ	δ	δ	$\top$	$\top$	$\top$

Para el programa $\mathcal{P}$ dado, la tabla anterior muestra el conjunto $\mathcal{K} = \{\mathcal{I}_1, \ldots, \mathcal{I}_{12}\}$ de todos los modelos de $\mathcal{P}$.

En efecto, por definición de modelo, $\mathcal{I}_j$ es modelo de la la regla $\mathcal{R}_2$ si, y sólo si, $\beta \leq \mathcal{I}_j(q)$, por lo que $\mathcal{I}_j(q)$ puede tomar los valores $\beta, \delta, \top \in L$. Además, $\mathcal{I}_j$ es un modelo de $\mathcal{R}_1$ si, y sólo si, $\alpha \leq \mathcal{I}_j(p \leftarrow_{\mathtt{G}} q) = \mathcal{I}_j(p) \dot{\leftarrow}_{\mathtt{G}} \mathcal{I}_j(q)$. Entonces, si fijamos como $\mathcal{I}_j(q)$ uno de los valores anteriores, y añadimos la condición $\alpha \leq \mathcal{I}_j(p) \dot{\leftarrow}_{\mathtt{G}} \mathcal{I}_j(q)$, quedan determinados los posibles valores de $\mathcal{I}_j(p)$ y, en consecuencia, los posibles modelos $\mathcal{I}_j \in \mathcal{K}$.

Finalmente, es sencillo comprobar que el ínfimo de $\mathcal{K}$ es la interpretación $\mathcal{I}_{\mathcal{P}}$ tal que $\mathcal{I}_{\mathcal{P}}(p) = \bot$ y $\mathcal{I}_{\mathcal{P}}(q) = \beta$), pero $\mathcal{I}_{\mathcal{P}} \notin \mathcal{K}$, es decir, $\mathcal{I}_{\mathcal{P}}$ no es un modelo de $\mathcal{P}$.

El concepto de interpretación y de modelo puede ser expresado en términos conjuntistas, entendiendo una interpretación de $\mathcal{P}$, en lugar de como una aplicación, como la correspondiente relación binaria. Damos a continuación el resultado elemental que justifica este hecho.

Proposición 3.3.4. *Sea $\mathcal{P}$ un programa lógico multi-adjunto con retículo asociado $(L, \leq)$. Sea $\mathcal{I}$ una interpretación de $\mathcal{P}$, es decir, una aplicación $\mathcal{I} : B_{\mathcal{P}} \longrightarrow L$. Entonces $\mathcal{I}$ determina una única relación binaria $R_{\mathcal{I}} \subset B_{\mathcal{P}} \times L$.*

Demostración. Es suficiente considerar que la aplicación $\mathcal{I}$ está determinada por su conjunto de imágenes, y la relación $R_{\mathcal{I}}$ es un conjunto de pares del tipo $(A, \mathcal{I}(A)), A \in B_{\mathcal{P}}$. En consecuencia, la aplicación $\mathcal{I}$ puede entenderse como una relación binaria, es decir, como un cierto conjunto pares ordenados cuya primera componente es una fórmula básica de la base de Herbrand y cuya segunda componente es un elemento del retículo L. □

En virtud del resultado anterior, cabe dar una interpretación $\mathcal{I}$ por su expresión conjuntista $R_{\mathcal{I}}$ que, cuando no sea necesario enfatizar este aspecto conjuntista, denotaremos también por $\mathcal{I}$.

En consecuencia, sin más que pensar cada modelo de $\mathcal{P}$ como un conjunto (subconjunto de $B_{\mathcal{P}} \times L$), esto es, $\mathcal{I}_j = \{\langle A, \alpha \rangle : A \in B_{\mathcal{P}}, \alpha \in L\}$, el modelo mínimo difuso puede caracterizarse también por el siguiente resultado.

Teorema 3.3.5. *El modelo mínimo difuso de $\mathcal{P}$ es la intersección de todos los modelos $\mathcal{P}$, es decir, $\mathcal{I} = \cap \mathcal{I}_j$, donde $\mathcal{I}_j$ es un modelo de $\mathcal{P}$, para todo j.*

3.4. Equivalencias

En lo que sigue abordamos todas las equivalencias entre las semánticas descritas anteriormente para la programación lógica multi-adjunta.

3.4.1. Equivalencia entre las semánticas declarativa y de punto fijo

Veremos que la semántica declarativa de modelo mínimo difuso es equivalente a la semántica de punto fijo que se construye en [Medina *et al.*, 2004] para la programación lógica multi-adjunta.

Teorema 3.4.1. *Dado el programa lógico multi-adjunto $\mathcal{P}$ con retículo (completo) asociado $(L, \leq)$, $\mathcal{I}_\mathcal{P}$ es modelo mínimo de Herbrand difuso si, y sólo si, $\mathcal{I}_\mathcal{P}$ es el menor punto fijo de $T_\mathcal{P}$.*

Demostración. Por el Teorema 3.3.2, $\mathcal{I}_\mathcal{P}$ es el modelo mímino difuso si, y sólo si, $\mathcal{I}_\mathcal{P} = \inf\{\mathcal{I}_j : \mathcal{I}_j \text{ es modelo de } \mathcal{P}\}$ y, por el Teorema 3.2.5, $\mathcal{I}_\mathcal{P} = \inf\{\mathcal{I}_j : T_\mathcal{P}(\mathcal{I}_j) \leq \mathcal{I}_j\}$. Además, por ser $T_\mathcal{P}$ un operador monótono en el retículo completo L (véase [Medina *et al.*, 2001d]), existe el menor punto fijo de $T_\mathcal{P}$ y coincide con el $\inf\{\mathcal{I}_j : T_\mathcal{P}(\mathcal{I}_j) \leq \mathcal{I}_j\}$ (véase [Lloyd, 1987]). En consecuencia, $\mathcal{I}_\mathcal{P}$ es el menor punto fijo del operador $T_\mathcal{P}$. □

Este último resultado afirma que la semántica declarativa de un programa lógico multi-adjunto $\mathcal{P}$ puede obtenerse por iteración de $T_\mathcal{P}$ sobre la interpretación mínima. Debe tenerse en cuenta que $T_\mathcal{P}$ puede no ser continuo, en cuyo caso –a diferencia de la programación lógica pura– pueden ser necesarias un número infinito no numerable de iteraciones para alcanzar el punto fijo [Medina *et al.*, 2001d].

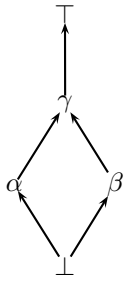
En el ejemplo posterior, se ilustra esta equivalencia para un programa multi-adjunto.

Ejemplo 3.4.2. *Consideremos el siguiente programa lógico multi-adjunto $\mathcal{P}$ formado por hechos, reglas en la que podemos tomar cuerpo vacío o cuerpo con el elemento $\top$ del retículo asociado $(L, \leq)$ determinado por el diagrama de Hasse de la figura:*

$\mathcal{R}_1 : \quad \langle p \quad \leftarrow_{\mathtt{G}} ; \quad \alpha \rangle$

$\mathcal{R}_2 : \quad \langle p \quad \leftarrow_{\mathtt{G}} ; \quad \beta \rangle$

$\mathcal{R}_3 : \quad \langle q \quad \leftarrow_{\mathtt{G}} ; \quad \beta \rangle$



	$\mathcal{I}_1$	$\mathcal{I}_2$	$\mathcal{I}_3$	$\mathcal{I}_4$	$\mathcal{I}_5$	$\mathcal{I}_6$
p	γ	γ	γ	$\top$	$\top$	$\top$
q	β	γ	$\top$	β	γ	$\top$

Las funciones de verdad de las conectivas $(\&_{\mathtt{G}}, \leftarrow_{\mathtt{G}})$ *(que siguen la lógica intuicionista de* Gödel*), forman un par adjunto en el retículo* $(L, \leq)$ *escogido.*

Para este programa existen seis modelos distintos (las interpretaciones $\mathcal{I}_1, \ldots, \mathcal{I}_6$ *de la tabla anterior) siendo* $\mathcal{I}_\mathcal{P} = \mathcal{I}_1$ *el modelo mínimo de Herbrand difuso. En efecto, se puede comprobar fácilmente que* $\mathcal{I}_\mathcal{P}$ *es el ínfimo del conjunto de* $\{\mathcal{I}_1, \ldots, \mathcal{I}_6\}$.

Además, resulta ser un punto fijo, puesto que

$$T_\mathcal{P}(\mathcal{I}_\mathcal{P})(p) = sup\{\alpha \dot{\&}_{\mathtt{G}} \top, \beta \dot{\&}_{\mathtt{G}} \top\} = sup\{\alpha, \beta\} = \gamma = \mathcal{I}_\mathcal{P}(p)$$

$$T_\mathcal{P}(\mathcal{I}_\mathcal{P})(q) = sup\{\beta \dot{\&}_{\mathtt{G}} \top\} = \beta = \mathcal{I}_\mathcal{P}(q)$$

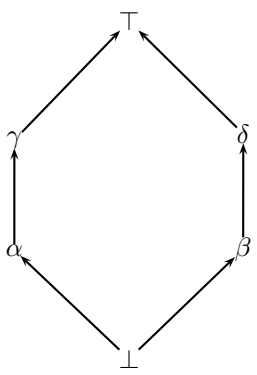
y su unicidad es también sencilla de comprobar. Tengamos en cuenta que, como era de esperar, $\mathcal{I}_\mathcal{P}$ *satisface los Teoremas 3.3.2 y 3.3.5.*

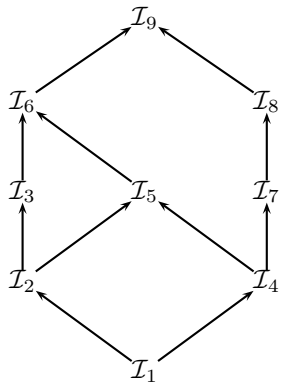
Observamos, por otra parte, que la semántica operacional (cuya equivalencia con la de punto fijo se trata en [Medina et al.*, 2001d]) no goza en general (como ocurre en este caso) de garantías de completitud:* α, β *son respuestas correctas para el programa* $\mathcal{P}$ *y el objetivo* p *y, por tanto,* γ *también es una respuesta correcta; sin embargo,* γ *no es una respuesta computada. Tal como veremos en el Capítulo 7, será necesario incorporar el* reductante $\mathcal{R} \equiv \langle p \leftarrow_{\mathtt{G}} @_{sup}(\alpha \& \top, \beta \& \top); \top \rangle$*, donde* $\dot{@}_{sup}(\alpha, \beta) = sup\{\alpha, \beta\} = \gamma$*, para garantizar la completitud.*

En el ejemplo que sigue se pone de manifiesto la relevancia del carácter multi-adjunto del retículo asociado para poder definir su semántica operacional y de punto fijo pero, sorprendentemente, no siempre es requerido para caracterizar su modelo mínimo difuso. Como ya dijimos, si el conjunto $(L, \leq)$ no es un retículo multi-adjunto, entonces el operador $T_\mathcal{P}$ no está definido y no existe semántica de punto fijo para el programa $\mathcal{P}$ considerado. La principal razón está en que en ambas semánticas se explota la relación que existe entre las aplicaciones de las reglas y sus conjunciones

adjuntas ya que las primeras deben ser "sustituidas" explícitamente por las segundas en los nuevos estados que se obtienen por aplicación de pasos admisibles $AS1$ o al iterar el operador $T_{\mathcal{P}}$. Tampoco existe semántica procedural, es decir, no hay respuestas computadas difusas. Pero conviene incidir de nuevo en que, también en este caso, tiene sentido definir el modelo mínimo difuso en los términos considerados y sigue existiendo la semántica declarativa como detallamos a continuación.

Ejemplo 3.4.3. *Consideremos de nuevo el retículo* $(L, \leq)$ *que sigue y el par* $(\&_{\mathtt{G}}, \leftarrow_{\mathtt{G}})$ *del Ejemplo 3.3.3, para el que ya hemos detallado que no satisface la propiedad adjunta (recordemos que, mientras* $\alpha \dot{\&} \delta \leq \beta$*, no se cumple que* $\alpha \leq \delta \dot{\rightarrow} \beta$*).*





$\mathcal{R}_1$: $\langle p \quad \leftarrow_{\mathtt{G}} q; \quad \alpha \rangle$

$\mathcal{R}_2$: $\langle q \quad \leftarrow_{\mathtt{G}} ; \quad \gamma \rangle$

	$\mathcal{I}_1$	$\mathcal{I}_2$	$\mathcal{I}_3$	$\mathcal{I}_4$	$\mathcal{I}_5$	$\mathcal{I}_6$	$\mathcal{I}_7$	$\mathcal{I}_8$	$\mathcal{I}_9$
p	$\bot$	α	γ	$\bot$	α	γ	β	δ	$\top$
q	γ	γ	γ	$\top$	$\top$	$\top$	$\top$	$\top$	$\top$

Para el programa $\mathcal{P}$ *anterior con retículo asociado* L*, el conjunto de modelos de* $\mathcal{P}$*, que se muestra en la tabla superior, es* $\mathcal{K} = \{\mathcal{I}_1, \ldots, \mathcal{I}_9\}$*. Tengamos en cuenta que* $\mathcal{K}$ *tiene estructura de retículo, con el orden heredado del conjunto de interpretaciones* $(\mathcal{H}, \leq)$*, como se puede observar en el grafo (adjunto) correspondiente. Dicho grafo evidencia no sólo la existencia del modelo mínimo de Herbrand difuso de* $\mathcal{P}$*, dado por* $\mathcal{I}_1$*, para el que se satisfacen los Teoremas 3.3.2 (minimalidad) y 3.3.5 (intersección de modelos).*

Este hecho tiene especial importancia toda vez que no está definida la semántica operacional ni de punto fijo de este programa (recordemos que ambas nociones exigen que se cumpla la propiedad adjunta, lo que no es cierto en este caso).

3.4.2. Relaciones de la semántica procedural con la de modelo mínimo y de punto fijo

Después de haber justificado en la sección anterior la equivalencia entre la semántica declarativa y la de punto fijo, y usando la equivalencia que se contempla en [Medina *et al.*, 2004]) entre esta última y la procedural, se obtiene como consecuencia que la semántica declarativa de modelo mínimo difuso es equivalente a la procedural en los términos precisados en [Medina *et al.*, 2004]. Es decir, está siempre garantizada la corrección fuerte pero sólo se alcanza la completitud aproximada incorporando a los programas multi-adjuntos la noción de reductante que estudiaremos en los Capítulos 6 y 7. En cualquier caso, en este apartado, aportamos (Teorema 3.4.8) una demostración original para la corrección de la semántica procedural.

Además, veremos que, en el contexto multi-adjunto, es posible caracterizar las respuestas correctas mediante el modelo mínimo difuso. Ha de tenerse en cuenta que en programación lógica pura el modelo mínimo de Herbrand no permite caracterizar completamente las respuestas correctas; así que, en este aspecto, lo obtenido en programación lógica multi-adjunta mejora los resultados clásicos de la programación lógica[9]. Dicha caracterización será útil en la práctica para obtener el conjunto de respuestas correctas de un programa a partir del modelo mínimo difuso, tal como ilustra el Ejemplo 3.4.6 de esta sección.

Definición 3.4.4 ([Medina *et al.*, 2004])**.** *Sea $\mathcal{P}$ un programa multi-adjunto y G un objetivo. El par $\langle \lambda; \theta \rangle$ es una respuesta correcta para $\mathcal{P}$ y G si, y sólo si, $\lambda \leq \mathcal{I}_j(G\theta)$, para todo modelo $\mathcal{I}_j$ de $\mathcal{P}$.*

El teorema que sigue enuncia la caracterización referida para la respuesta correcta $\langle \lambda; \theta \rangle$, a la vez que muestra que $\mathcal{I}_{\mathcal{P}}(G\theta)$ es la mejor respuesta correcta que puede obtenerse para el objetivo G en el programa $\mathcal{P}$.

Teorema 3.4.5. *Sea $\mathcal{P}$ un programa multi-adjunto y G un objetivo. El par $\langle \lambda; \theta \rangle$ es una respuesta correcta para $\mathcal{P}$ y G si, y sólo si, $\lambda \leq \mathcal{I}_{\mathcal{P}}(G\theta)$, donde $\mathcal{I}_{\mathcal{P}}$ es el modelo mínimo difuso de $\mathcal{P}$.*

[9] En ello influirá decisivamente el carácter no refutacional del lenguaje multi-adjunto.

Demostración. Inmediata, sin más que usar la definición de modelo mínimo de Herbrand difuso. □

Obsérvese que si $G\theta$ es básica, $\mathcal{I}_{\mathcal{P}}(G\theta) = \alpha$ y el modelo mínimo difuso se concibe como un conjunto, podríamos escribir $(G\theta, \alpha) \in \mathcal{I}_{\mathcal{P}}$ para adaptar al contexto difuso lo que es habitual considerar en la programación lógica. Además, advertimos que la definición anterior de respuesta correcta difiere de la dada en programación lógica clásica (puede verse [Lloyd, 1987] a tal efecto) puesto que que la resolución-SLD es refutacional y la semántica procedural de los programas multi-adjuntos no lo es. El ejemplo posterior sugiere adecuadamente cómo se pueden obtener las respuestas correctas a partir del modelo mínimo difuso.

Ejemplo 3.4.6. *Para el programa $\mathcal{P}$ siguiente cuyo retículo asociado $(L, \leq)$ es el considerado en el Ejemplo 3.4.2,*

$$\begin{array}{lllll} \mathcal{R}_1: & \langle p(a) & \leftarrow_{\mathtt{G}} ; & & \alpha\rangle \\ \mathcal{R}_2: & \langle p(b) & \leftarrow_{\mathtt{G}} ; & & \beta\rangle \\ \mathcal{R}_3: & \langle q(a) & \leftarrow_{\mathtt{G}} p(a); & & \gamma\rangle \end{array}$$

es sencillo comprobar que el modelo mínimo de Herbrand difuso $\mathcal{I}_{\mathcal{P}}$ está definido por $\mathcal{I}_{\mathcal{P}}(p(a)) = \alpha$, $\mathcal{I}_{\mathcal{P}}(p(b)) = \beta$, $\mathcal{I}_{\mathcal{P}}(q(a)) = \alpha$, $\mathcal{I}_{\mathcal{P}}(q(b)) = \bot$. Entonces:

i) Para el objetivo $p(a)$ se obtiene el conjunto de respuestas correctas

$$\{\langle \lambda; id\rangle : \lambda \in L, \lambda \leq \alpha\}$$

ii) Para el objetivo $p(b)$ se obtiene el conjunto de respuestas correctas

$$\{\langle \lambda; id\rangle : \lambda \in L, \lambda \leq \beta\}$$

iii) Para el objetivo $q(a)$ se obtiene el conjunto de respuestas correctas

$$\{\langle \lambda; id\rangle : \lambda \in L, \lambda \leq \alpha\}$$

iv) Para el objetivo $p(x)$, el conjunto de respuestas correctas es

$$\{\langle \lambda; \theta\rangle : \lambda \in L, \lambda \leq \mathcal{I}_{\mathcal{P}}(p(x)\theta)\} = \{\langle \bot; \{X/a\}\rangle, \langle \alpha; \{X/a\}\rangle, \langle \bot; \{X/b\}\rangle, \langle \beta, \{X/b\}\rangle\}$$

Recuérdese que el resultado de aplicar el modelo mímimo $\mathcal{I}_\mathcal{P}$ sobre $p(x)$ nos da $\mathcal{I}_\mathcal{P}(p(x)) = inf\{\mathcal{I}_\mathcal{P}(p(x)\sigma) : p(x)\sigma$ es básica $\} =^{10} inf\{\mathcal{I}_\mathcal{P}(p(a)), \mathcal{I}_\mathcal{P}(p(b))\}$ $= inf\{\alpha, \beta\} = \bot$.

v) Para el objetivo $q(x)$, el conjunto de respuestas correctas es

$$\{\langle\lambda;\theta\rangle : \lambda \in L, \lambda \leq \mathcal{I}_\mathcal{P}(q(x)\theta)\} = \{\langle\alpha;\{X/a\}\rangle, \langle\bot;\{X/b\}\rangle\}$$

De manera análoga al caso anterior, se tiene ahora que la interpretación $\mathcal{I}_\mathcal{P}(q(x)) = inf\{\mathcal{I}_\mathcal{P}(q(a)), \mathcal{I}_\mathcal{P}(q(b))\} = inf\{\alpha, \bot\} = \bot$ (resulta sencillo comprobar que el modelo mínimo ha de estar definido de este modo a partir de las fórmulas $q(a), q(b)$).

En el teorema siguiente aportamos una demostración original de la corrección de la semántica procedural de la programación multi-adjunta. En ella se observa cierta analogía con la que se recoge en [Lloyd, 1987] para el caso de la programación lógica pura, pese a que el carácter no refutacional de nuestro lenguaje junto con la componente borrosa del mismo, marcan diferencias muy significativas entre ambas. Antes de abordar el resultado mencionado, enunciamos el lema que sigue que tiene carácter instrumental.

Lema 3.4.7. *Sea $(L, \leq)$ un retículo ordenado completo. Para cualesquiera subconjuntos A, B de L se verifica que si $A \subset B$, entonces $inf(B) \leq inf(A)$.*

Demostración. Es suficiente considerar la definición de ínfimo y el carácter completo de $(L, \leq)$. □

Obsérvese que, en virtud de lema anterior, resulta que $\mathcal{I}(A\theta) \geq \mathcal{I}(A)$[11], para toda sustitución θ, para toda interpretación $\mathcal{I}$, toda vez que el conjunto de instancias básicas de la fórmula $A\theta$ está contenido en el conjunto de instancias básicas de A.

Teorema 3.4.8 (Corrección parcial)**.** *Sea $\mathcal{P}$ un programa lógico multi-adjunto, A un objetivo atómico y $\langle\lambda;\theta\rangle$ una respuesta computada difusa para A en $\mathcal{P}$. Entonces, $\langle\lambda;\theta\rangle$ es una respuesta correcta para A en $\mathcal{P}$.*

Demostración. Sea $D : [G_1, \ldots, G_n]$ una derivación donde $G_1 = \langle A; id\rangle \rightarrow^n_{AS/IS}$ $\langle\lambda;\theta\rangle = G_n$. Hacemos la demostracción por inducción en n, siendo n es la longitud de D.

[10] Las sustituciones contemplarán sólo términos del universo de Herbrand del programa en lugar de variables.

[11] Véanse, por ejemplo, los apartados $iv), v)$ del Ejemplo 3.4.6.

Veamos en primer lugar que el resultado es cierto para $n = 1$. En efecto, si para el objetivo A existe la derivación $\langle A; id\rangle \rightarrow_{AS} \langle\lambda;\theta\rangle$, entonces la regla $\mathcal{R} : \langle H\leftarrow_i;\lambda\rangle \in \mathcal{P}$ y $A\theta = H\theta$. En tal caso, todo modelo $\mathcal{I}$ de $\mathcal{P}$ satisface la regla $\mathcal{R}$ y, por tanto, $\mathcal{I}(H\leftarrow_i) \geq \lambda$, es decir, $\mathcal{I}(H) \geq \lambda$. Además, de la igualdad $A\theta = H\theta$ resulta $\mathcal{I}(A\theta) = \mathcal{I}(H\theta)$ y por el Lema 3.4.7, $\mathcal{I}(H\theta) \geq \mathcal{I}(H)$. En consecuencia, se tiene $\mathcal{I}(A\theta) \geq \lambda$ y $\langle\lambda;\theta\rangle$ es una respuesta correcta para A en $\mathcal{P}$, como queríamos.

Supongamos que el resultado es cierto para toda derivación de longitud k y veamos que se verifica para una de longitud $k+1$ arbitraria, $D : [G_1, \ldots, G_{k+1}]$. Observando el primer paso de la derivación D, tenemos $G_1 = \langle A; id\rangle \rightarrow_{AS} \langle v\&_i\mathcal{B}\sigma;\sigma\rangle = G_2$. Es decir, el paso admisible ha sido ejecutado usando la regla de programa $\mathcal{R} : \langle H\leftarrow_i \mathcal{B}; v\rangle$, donde el átomo A unifica con la cabeza de la regla $\mathcal{R}$ mediante el mgu σ. Para cada átomo $B_i\sigma$[12], $i = 1, \ldots, n$, de $\mathcal{B}\sigma$ existe una derivación de longitud menor o igual que k que conduce a una respuesta computada $\langle b_i;\tau_i\rangle$. De manera más precisa, D comprende los siguientes pasos de derivación

$$
\begin{array}{ll}
D : [\,\langle A; id\rangle & \rightarrow_{AS} \\
\langle v\&_i\mathcal{B}\sigma;\sigma\rangle & = \\
\langle v\&_i@(B_1\sigma, \ldots, B_n\sigma);\sigma\rangle & \rightarrow^{l_1}_{AS/IS} \\
\langle v\&_i@(b_1, \ldots, B_n\sigma);\sigma\circ\tau_1\rangle & \rightarrow^{l_n}_{AS/IS} \\
\langle v\&_i@(b_1, \ldots, b_n);\sigma\circ\tau_1\circ\cdots\circ\tau_n\rangle^{13} & \rightarrow_{IS} \\
\langle v\&_i b;\sigma\circ\tau\rangle & \rightarrow_{IS} \\
\langle\lambda;\theta\rangle] &
\end{array}
$$

donde $\tau = \tau_1 \circ \tau_2 \circ \cdots \circ \tau_n$, $\theta = \sigma \circ \tau$, $\lambda = v\dot{\&}_i b$, $l_1 + l_2 + \cdots + l_n = k - 2$ y $b = \dot{@}(b_1, \ldots, b_n)$, siendo @ la combinación de todos las conjunciones, disyunciones y agregadores que enlazan los elementos $b_i \in L$ para obtener la respuesta computada $\langle b;\tau\rangle$ correspondiente al objetivo $\mathcal{B}\sigma$.

Por la hipótesis de inducción, para cada $B_i\sigma$, $\langle b_i;\tau_i\rangle$ es una respuesta correcta y, entonces, $\mathcal{I}(B_i\sigma\tau_i) \geq b_i$, para toda interpretación $\mathcal{I}$ que sea modelo de $\mathcal{P}$. En tal caso, de $\mathcal{I}(B_i\sigma\tau_i) \geq b_i$ resulta que $\mathcal{I}(\mathcal{B}\sigma) \geq b$ puesto que $\mathcal{I}(\mathcal{B}\sigma)$ se obtiene a

[12]Sin pérdida de generalidad, podemos suponer que en la derivación considerada se ejecutan en primer término todos los pasos admisibles antes de dar un paso interpretativo.

partir de $\mathcal{I}(B_i\tau_i)$ como resultado de aplicar las funciones de verdad de conjunciones, disyunciones o agregadores y todas ellas son monótonas en cada componente.

Entonces, de la igualdad $A\sigma = H\sigma$ se obtiene $A\theta = H\theta$ y, por tanto, $\mathcal{I}(A\theta) = \mathcal{I}(H\theta)$. Además, usando el Lema 3.4.7 en primer término y teniendo en cuenta posteriormente que $(\leftarrow_i, \&_i)$ son un par adjunto, resulta $\mathcal{I}(H\theta) \geq \mathcal{I}(H) \geq v\dot{\&}_i\mathcal{I}(\mathcal{B}\sigma) \geq v\dot{\&}_i b = \lambda$.

En consecuencia, $\mathcal{I}(A\theta) \geq \lambda$ y $\langle\lambda;\theta\rangle$ es una respuesta correcta para el programa $\mathcal{P}$ y el átomo A, como queríamos. □

El siguiente teorema, recogido en [Medina *et al.*, 2004], enuncia un resultado de completitud aproximada para la programación lógica multi-adjunta. Es preciso observar que su justificación exige incorporar el concepto de reductante [Medina *et al.*, 2004; Julián *et al.*, 2006b, 2007b, 2009] que abordaremos en los Capítulos 6 y 7.

Teorema 3.4.9 (Completitud aproximada)**.** *Sea $\mathcal{P}$ un programa, A un átomo básico y $\langle\lambda; id\rangle$ una respuesta correcta para A en $\mathcal{P}$. Entonces, existe una secuencia de respuestas computadas $\langle\lambda_n; id\rangle$ para A en $\mathcal{P}'$ tal que $\lambda \leq sup\{\lambda_n : n \in \mathbb{N}\}$.*

Además, puede verse en [Medina y Ojeda-Aciego, 2002] cómo, en el caso proposicional, el operador $T_{\mathcal{P}}$ aporta la mayor respuesta correcta para un objetivo A y un programa $\mathcal{P}$ (entendida como el supremo, en el retículo L, del conjunto de respuestas correctas) y, suponiendo terminación y continuidad, la mayor respuesta computada (es decir, una respuesta computada λ verificando que $\lambda' \leq \lambda$, para toda respuesta computada λ').

3.5. Conclusiones

En este capítulo hemos realizado las siguientes aportaciones:

- Clarificamos la semántica procedural del lenguaje, añadiendo a la fase operacional, debida a los autores del lenguaje (Medina *et al.* [2004]), una fase interpretativa que se concibe como un sistema de transición de estados. Este hecho resultará imprescindible a la hora de formular posteriormente, para este tipo de programas, la transformación de desplegado interpretativo.

- Demostramos la independencia de la regla de computación para la programación lógica multi-adjunta, extendiendo el resultado clásico de la programación lógica.

- Definimos el concepto de modelo mínimo de Herbrand difuso para programas lógicos multi-adjuntos, adaptando los conceptos clásicos de la programación lógica pura para esta clase de programas lógicos borrosos.

- Expresamos la semántica declarativa de los programas mencionados en términos del modelo mínimo de Herbrand difuso, con la ventaja añadida de que permite caracterizar las respuestas correctas, mejorando así el caso de la programación lógica pura.

- Demostramos la equivalencia de dicha semántica con la de punto fijo ya obtenida por los autores del lenguaje.

- Obtenemos una demostración original de la corrección para la semántica procedural, considerando también sus conexiones con las semánticas anteriores.

- Por último, incidimos cómo puede afectar el carácter multi-adjunto del retículo a la hora de disponer de las correspondientes semánticas de estos programas constatando que, sorprendentemente, la semántica declarativa es mucho menos dependiente de esta característica que las otras dos.

Capítulo 4

Desplegado difuso

El desplegado de un programa declarativo se entiende, en general, como el proceso en virtud del cual una regla (o cláusula) del mismo es sustituida por un nuevo conjunto de reglas obtenidas a partir de la original, sobre la que se realiza uno (o varios) pasos de ejecución *simbólica*. Este tipo de ejecución se refiere a un tipo de evaluación donde, en ausencia de datos reales, se ejecutan llamadas "simbólicas".

Más concretamente, la idea básica subyacente a toda operación de desplegado consiste en efectuar esta clase de evaluación simbólica ya no sobre un objetivo, sino sobre el cuerpo (o la parte derecha) de una regla del programa.

Dependiendo del tipo de paradigma declarativo que estemos considerando (lógico puro, funcional puro, lógico-funcional o lógico difuso) se trabaja con un mecanismo operacional diferente (resolución-SLD, reescritura, *estrechamiento* (del inglés *narrowing*), resolución-SLD difusa y otros, sobre el que basar (de alguna manera) la computación simbólica e implementar el desplegado.

Es bien conocido en la literatura de transformación de programas que, en general, el desplegado incrementa el tamaño de los programas residuales y, por tanto, se necesita más memoria para su almacenamiento. Sin embargo, es habitual que se reduzca la cantidad de memoria requerida en tiempo de ejecución, debido principalmente a la reducción de la longitud de las derivaciones (en programación declarativa, existe una correspondencia directa entre los estados de derivación y elementos almacenados en la pila del sistema). Estos hechos justifican los beneficios (no sólo en tiempo, sino también en espacio) del uso del desplegado.

Comenzamos haciendo un breve recorrido histórico que explora los antecedentes

del desplegado en contextos funcionales y lógicos puros y esbozando el problema de la adaptación de esta regla a un contexto lógico difuso.

Posteriormente, centrándonos ya en el marco difuso, recogemos dos tipos de desplegado, uno primitivo para el lenguaje lf-Prolog y otro para el lenguaje lef-Prolog, que han sido considerados, respectivamente, en [Julián *et al.*, 2004b,c] y en [Julián *et al.*, 2004a, 2005c]. Asimismo, formalizamos el conjunto de reglas reemplazamiento de t-norma para lef-Prolog tal como detallamos en [Julián *et al.*, 2004a, 2005c].

Contemplamos, además, dos tipos de desplegado fuertemente relacionados: el operacional, introducido por primera vez en [Julián *et al.*, 2005a], y el interpretativo, introducido por primera vez en [Julián *et al.*, 2006c]. Ambos se formulan para el lenguaje multi-adjunto (estudiado en el Capítulo 2), es decir, en un marco difuso muy general y potente.

4.1. Motivación, antecedentes y aplicaciones

El primer trabajo sobre transformación de programas basado en las reglas de plegado y desplegado es original de Burstall y Darlington [1977]. Se trata de un trabajo pionero en esta línea, referido a programas funcionales puros definidos como un conjunto de ecuaciones recursivas.

El desplegado se define en [Burstall y Darlington, 1977] textualmente como sigue:

> *Si $E = E'$ y $F = F'$ son ecuaciones y existe alguna ocurrencia de F' que es instancia de E, entonces se reemplaza por la instancia correspondiente de E', obteniendo F'' y añadiendo la ecuación $F = F''$.*

Cabe destacar que este proceso puede verse como un paso de reescritura sobre la parte derecha de la ecuación a desplegar. En esta primera aproximación no se hace explícito el hecho de que la regla desplegada debe retirarse del conjunto resultante de ecuaciones y no se dan tampoco resultados de corrección ni de completitud para la transformación.

En trabajos posteriores sobre transformación de programas funcionales [Scherlis, 1981; Kott, 1985; Zhu, 1994; Sands, 1996], la operación de desplegado se formula prácticamente sin variaciones, destacando siempre el hecho de que esta regla se basa en la reescritura y sólo involucra emparejamiento.

Puesto que las llamadas a función en el cuerpo de las reglas del programa pueden contener variables, las transformaciones de Burstall y Darlington hacen uso de una operación previa de *instanciación* [Burstall y Darlington, 1977].

De esta forma, si una llamada a función no empareja con ninguna de las funciones definidas en el programa, sus variables pueden instanciarse para permitir así su desplegado.

Por todo ello, se puede considerar que el proceso de transformación básico en los programas funcionales consta realmente de instanciación + desplegado. En particular, la ejecución de una llamada a función conteniendo variables (valores simbólicos), recibe el nombre de ejecución simbólica. Así, en el caso de los programas funcionales el mecanismo de ejecución (reescritura) y el mecanismo de ejecución simbólica (instanciación + reescritura) no coinciden, lo que significa que hay que desarrollar técnicas *ad-hoc* para los métodos de transformación, a diferencia de lo que ocurre en los contextos lógicos puros, lógico-funcionales y lógicos difusos.

La aproximación más reciente a las técnicas de transformación de programas funcionales es la presentada por Sands [1995, 1996]. En general, las transformaciones de plegado/desplegado, definidas por Burstall y Darlington [1977], no garantizan ni la mejora en eficiencia ni la corrección total de la transformación. En [Sands, 1995] se introduce una condición (semántica) para la corrección total de las transformaciones sobre programas funcionales perezosos de orden superior. El principal resultado técnico consiste en que, si los pasos locales de transformación se realizan siguiendo un cierto criterio, entonces se puede asegurar la corrección total de la transformación.

En [Sands, 1996], se muestra cómo el resultado anterior permite demostrar la corrección total de algunas de las principales técnicas automáticas de transformación de programas, incluyendo la deforestación de Wadler [1990] y la supercompilación de Turchin [1986].

La primera adaptación de las reglas de plegado/desplegado al caso de los programas lógicos se debe a Tamaki y Sato, quienes presentaron sus resultados a mediados de la década de los 80. En [Tamaki y Sato, 1984] se extienden las ideas originales de Burstall y Darlington [1977] reemplazando el emparejamiento por la unificación en las reglas de transformación.

El desplegado de un programa lógico consiste, por tanto, en aplicar un paso de resolución a un subobjetivo en el cuerpo de una cláusula de todas las formas posibles. Gracias al mecanismo de unificación, el desplegado de los programas lógicos permite propagar información sintáctica (como es la estructura de los términos) y no sólo valores constantes (como en el caso del desplegado de los programas funcionales).

De esta forma, se obtiene un mecanismo más potente que, además, no requiere la presencia de una regla de instanciación independiente, ya que dicho proceso va

implícito en la propia regla de desplegado basada en el mecanismo de resolución.

Como se indica en [Pettorossi y Proietti, 1996b], el desplegado de los programas lógicos no sólo se diferencia del de los programas funcionales en el reemplazamiento del emparejamiento de patrones por la unificación sintáctica, sino también en la cantidad de reglas desplegadas que genera, como consecuencia, de ellos.

Ya que en un programa funcional es frecuente considerar funciones definidas mediante reglas cuyas partes izquierdas no solapan, el desplegado (previa instanciación) de una de ellas genera una y solamente una regla nueva, a diferencia del desplegado de una cláusula en un programa lógico que, en general, devuelve un conjunto de cláusulas.

Para el caso de los programas lógicos se han propuesto en la literatura diversas aproximaciones a la regla de desplegado (ver por ejemplo los diferentes trabajos de Proietti y Petorossi para un estudio extenso y recopilatorio del tema [Proietti y Pettorossi, 1993; Pettorossi y Proietti, 1994, 1996b, 1998]).

Todas ellas son básicamente similares a la original de Tamaki y Sato [1984], aunque en ocasiones se permite que la regla a desplegar y las reglas desplegantes pertenezcan a programas distintos [Pettorossi y Proietti, 1994], o que puedan ser varias las reglas desplegadas simultáneamente [Levi y Mancarella, 1988; Bossi *et al.*, 1994], etc.

En el contexto lógico-funcional, no existen en la literatura muchos trabajos relacionados con el desplegado de este tipo de programas. En [Alpuente *et al.*, 1997b, 1999, 2004; Moreno, 2000] se presentan diversas variantes de desplegado (y otras reglas de transformación) basadas en *estrechamiento* (que en esencia consiste en reescritura precedida de unificación), donde el propio método de evaluación estándar de los lenguajes integrados se revela muy adecuado para implementar la regla de desplegado.

De la misma forma que este mecanismo operacional admite diferentes refinamientos, se pueden plantear diferentes modelos de desplegado utilizando diferentes variantes del *estrechamiento* para definir la regla de transformación. Esto es, es posible dar una definición genérica de desplegado de programas lógico-funcionales de forma paramétrica con respecto a la estrategia de *estrechamiento* que se use en cada caso. Cada instancia exige una serie de condiciones de aplicabilidad y disfruta de una serie de propiedades.

La transformación de desplegado se ha mostrado muy útil desde que fue concebida y descrita por primera vez tanto en contextos funcionales como lógicos puros.

Sus aplicaciones se extienden sobre una amplia gama de áreas de trabajo: desde las puramente teóricas (semánticas por desplegado, composicionales, modulares, etc.) hasta otras más aplicadas (síntesis de programas, transformación y/o especialización de programas, etc.).

Además, merece ser destacado el papel fundamental que jugará la transformación de desplegado (en el contexto de la programación lógica multi-adjunta) tanto a nivel de producir evaluadores parciales y mejorar el cálculo de reductantes, lo que abordamos en el Capítulo 7, como a la hora de construir semánticas por desplegado para este lenguaje. En el Capítulo 9 mencionaremos con mayor detalle esta última aplicación del desplegado que pretendemos abordar como trabajo futuro.

4.2. Primera aproximación al desplegado difuso

La transformación de programas es un técnica útil para la optimización de programas que, partiendo de un programa inicial $\mathcal{P}_0$ deriva una secuencia $\mathcal{P}_1, \ldots, \mathcal{P}_n$ de programas trasformados mediante *reglas de transformación elementales*. El objetivo es que el programa final $\mathcal{P}_n$ tenga el mismo significado que $\mathcal{P}_0$, pero mejore su eficiencia en algún aspecto.

La transformación de programas puede verse como una metodología para desarrollo de software, de ahí su importancia (primero se desarrolla un programa sin preocuparse de la eficiencia del código, y después se optimiza con técnicas más o menos automáticas). Entre las reglas mencionadas, la llamada regla de *desplegado* ha sido una de las más ampliamente estudiadas.

En esencia, la transformación de desplegado aplicada sobre un programa (declarativo) consiste básicamente en el reemplazamiento de una de sus reglas (o cláusulas) por todas aquéllas que se obtienen al aplicar un paso de *computación simbólica* de todas las formas posibles sobre el cuerpo de la misma [Pettorossi y Proietti, 1996b]. En función del paradigma concreto que se esté utilizando (por ejemplo, funcional [Burstall y Darlington, 1977], lógico [Tamaki y Sato, 1984] o lógico-funcional [Alpuente *et al.*, 2004], que son quizás los campos donde mejor se ha estudiado esta transformación) el paso de computación estará basado en (alguna variante) del mecanismo operacional asociado al paradigma en cuestión (reescritura, resolución o estrechamiento, respectivamente).

La regla de desplegado permite, incluso sin combinarse con otro tipo de transformaciones, importantes optimizaciones en el programa código original. Aunque

la operación de desplegado (en sus distintas formulaciones) ha demostrado ampliamente su utilidad en técnicas de análisis, depuración, compilación, síntesis, etc., de programas declarativos, donde posiblemente alcanza sus mejores cotas de aplicación es en sistemas de especialización por evaluación parcial y en sistemas de transformación por plegado/desplegado basados en "reglas+estrategias" [Pettorossi y Proietti, 1996a].

En principio, y centrándonos ya estrictamente en la operación de desplegado de forma aislada (independientemente de su posterior uso en sucesivos capítulos), una buena formulación de la transformación debe garantizar que al ser aplicada a un programa concreto, nos permita derivar otro semánticamente equivalente pero con un mejor comportamiento con respecto a algún criterio de eficiencia.

Intuitivamente, este segundo aspecto lo garantiza el hecho de que los pasos de computación que se adelantan durante el proceso de desplegado (y que de alguna manera quedan "compilados" en el programa transformado) se ahorran en las computaciones finales sobre objetivos reales, que de esta forma se benefician del hecho de poder ser evaluados contra el programa desplegado de forma más rápida (es decir, mediante derivaciones más cortas).

Mucho más dificil es, por otra parte, garantizar que se preserva la equivalencia semántica entre los programas, ya que en este apartado entran en juego distintas variantes, como por ejemplo:

- la identificación de condiciones de aplicabilidad seguras y suficientemente relajadas que permitan cubrir un mayor número de casos (al menos un conjunto significativo), lo que muchas veces viene condicionado por
- la sintaxis del lenguaje, que a veces no da siquiera cobertura suficiente para plantear la transformación o demostrar sus propiedades,
- el observable que se pretende preservar (por ejemplo, modelo mínimo de Herbrand, c.a.s's, valores, terminación, etc.) y el grado de precisión de los resultados perseguidos (coincidencia total, parcial, aproximada, etc.)
- la semántica operacional considerada que será la base sobre la que se establezca la definición de desplegado, así como sus requisitos técnicos, propiedades de corrección/completitud, clase de programas a los que es aplicable de forma segura, etc.

Puesto que la programación lógica difusa es un área de investigación emergente, de muy recientemente desarrollo (tanto es así, que todavía no puede hablarse de estándares claros, sino más bien de una amplia gama de propuestas con diferentes características, defectos y bondades, como hemos detallado en los capítulos previos) y para la que se vislumbra un futuro prometedor, en esta tesis surge el interés por aplicar/adaptar las diferentes técnicas conocidas de optimización de programas basadas en alguna forma de desplegado.

Hemos considerado distintos lenguajes lógicos borrosos descritos en la literatura reciente, todos ellos basados en alguna variante de resolución difusa con más o menos fortuna en cuanto a su implantación real, propiedades teóricas (en especial, corrección/completitud de su semántica operacional con respecto a la declarativa, cuando esta última está definida), apariencia y posibilidades sintácticas, potencia expresiva y computacional, campos de aplicación, etc.

En particular, nos propusimos definir inicialmente la transformación de desplegado para el lenguaje contemplado en:

- [Arcelli y Formato, 1999] que, entre sus mejores bazas, propone una visión bastante purista de la noción de "difuminación" –entendida aquí como "similaridad"– adaptada a la programación lógica mediante la extensión del mecanismo de unificación (por similaridad). Pero la sintaxis de este lenguaje no es la adecuada para nuestros objetivos, dado que no es posible propagar el grado de verdad de la regla original sobre la regla desplegada. Véase para mayor detalle la Sección 2.1, en especial el Ejemplo 2.1.3 en el que se justifica que sería necesario modificar la sintaxis del lenguaje original y, lo más relevante, modificar la semántica operacional para que pueda acometerse el desplegado de programas LIKELOG.

Nos hemos centrado en formular la transformación de desplegado para los lenguajes definidos en:

- [Vojtáš y Paulík, 1996] que propone una visión casi antagónica del anterior en cuanto al mecanismo operacional (una extensión de la resolución-SLD) y la unificación (sintáctica). Este lenguaje mantiene la unificación sintáctica clásica incorporando la componente difusa en los predicados y en las conectivas, y también dispone de semanticas declarativa (modelo mínimo de Herbrand difuso), operacional (resolución-SLD difusa) y por punto fijo (así como de

sus propiedades e interrelaciones asociadas) bien asentadas (al estilo de Lloyd [1987]).

A nivel sintáctico, se adecúa más a nuestros propósitos, especialmente después de plantear las versiones etiquetadas del mismo que vimos en el Capítulo 2.

- [Medina *et al.*, 2001d], esto es, abordamos el estudio del desplegado en el contexto de la programación multi-adjunta, un marco de programación lógica difusa muy general para el que existe abundante literatura. Este lenguaje tiene una sintaxis mucho más rica que el considerado en [Vojtáš y Paulík, 1996] y posee también semántica procedural, declarativa por modelo mínimo –que nosotros hemos formulado– y de punto fijo tal como hemos detallado en el Capítulo 3.

Desplegado difuso de programas f-Prolog

En una primera aproximación, nos centramos en el desplegado de programas f-Prolog antes de pasar al caso extendido de ef-Prolog que veremos posteriormente.

Como ya consideramos, las diferencias entre los programas f-Prolog y lf-Prolog aparecen sólo en el nivel sintáctico: mientras el cuerpo B de una cláusula de programa (no unitaria) f-Prolog (que, en esencia, no es más que un objetivo simple, esto es, un átomo o una conjunción de átomos) respeta la gramática libre de contexto

$$B \rightarrow B, \ldots, B \mid atom,$$

necesitamos enriquecer este conjunto de reglas gramaticales del modo

$$B \rightarrow \boxed{\mathrm{L}_p}\ B\ \boxed{\mathrm{R}_p} \mid number,$$

si deseamos incluir marcas y números reales en el cuerpo de cláusulas lf-Prolog (que intuitivamente tienen la misma estructura que algún objetivo inicial, intermedio o final que aparezca en las derivaciones-SLD difusas).

Esto supone que todo programa f-Prolog es también un programa lf-Prolog, aunque el recíproco no es cierto (es decir, el conjunto de programas f-Prolog es un subconjunto del de programas lf-Prolog). Además de este hecho puntual (que, por otra parte, es preceptivo para definir el principio de resolución-SLD) ambos lenguajes comparten todo tipo de semánticas: obviamente la operacional, pero también la declarativa y la de punto fijo, y sus propiedades de corrección y completitud.

Como ya dijimos, la regla de desplegado consiste, en esencia, en la aplicación de un paso de computación simbólico en el cuerpo de la cláusula que, en nuestro entorno difuso, corresponde a la aplicación de cualquiera de las cuatro reglas descritas en la Definición 2.2.4.

Obsérvese que este proceso siempre genera cláusulas cuyos cuerpos incluyen marcas o números reales. Más exactamente, las dos últimas reglas reemplazan marcas y/o números por un nuevo número, lo que significa que tanto la expresión de entrada como la de salida son siempre expresiones válidas If-Prolog, pero nunca expresiones f-Prolog.

Algo similar ocurre con las dos primeras reglas de la Definición 2.2.4 que introduce marcas y números, respectivamente, en la expresión de salida, incluso en el caso en que la expresión de entrada es una expresión válida f-Prolog.

A priori, todos estos hechos garantizan que una transformación de desplegado basada en resolución-SLD difusa preserva la estructura sintáctica de los programas etiquetados If-Prolog pero, incluso en el caso en que los programas originales son siempre f-Prolog, el transformado nunca será de esta subclase: las marcas o números reales incorporados en las cláusulas transformadas por pasos de desplegado basados en la primera o segunda regla de resolución, respectivamente, supone la pérdida de la sintaxis f-Prolog original. Además, incluso en el caso en que el programa de entrada pertenezca a esta subclase, el resultante del desplegado será únicamente un programa If-Prolog.

Para evitar este inconveniente, la siguiente definición se enuncia en el entorno más general de programas If-Prolog en lugar de referirla a la subclase de programas f-Prolog.

Definición 4.2.1. *Sea $\mathcal{P}$ un programa* If-Prolog *y sea $\mathcal{C} \equiv \langle A \leftarrow \overline{B};\ p\rangle \in \mathcal{P}$ una cláusula de programa (no unitaria)* If-Prolog. *Entonces, el desplegado borroso de un programa $\mathcal{P}$ con respecto a la cláusula $\mathcal{C}$ es el nuevo programa* If-Prolog *$\mathcal{P}' = (\mathcal{P} - \{\mathcal{C}\}) \cup \mathcal{U}$ tal que:*

$$\mathcal{U} = \{\langle A\sigma \leftarrow \overline{B'};\ p\rangle \mid\ \langle \overline{B}; id\rangle \rightarrow_{FR} \langle \overline{B'}; \sigma\rangle\}$$

Contemplemos algunas observaciones acerca de la definición anterior. De manera análoga a la regla de desplegado clásico basada en resolución-SLD planteada en [Tamaki y Sato, 1984], las sustituciones obtenidas en los pasos de computación, en el proceso de desplegado, son incorporadas a las reglas transformadas de manera natural, esto es, aplicándolas en la cabeza de la cláusula.

Por otra parte, en cuanto a la propagación de los grados de verdad, resolvemos este problema de manera muy natural: la cláusula desplegada hereda directamente el grado p de la original.

Sin embargo, un análisis en profundidad de la transformación de desplegado revela cómo también se incluye información compilada de ambos componentes de una respuesta computada difusa (sustitución y grado de verdad) en el cuerpo de la cláusula transformada.

Acerca de los grado de verdad, p, observamos también que, si el paso de desplegado está basado en la tercera o cuarta regla de la Definición 2.2.4, las marcas y los números del cuerpo de la cláusula transformada han sido simplificados. Por otra parte, el grado p de la cláusula involucrada en un paso de desplegado basado en la Regla 1 o la Regla 2, se acumula en el cuerpo de la transformada mediante marcas y números reales. Por tanto, la propagación de los p's en el desplegado se realiza en dos niveles: incorporando directamente el grado p de la cláusula original en el p de la transformada, y a través de la simplificación/introdución de marcas y números reales en su cuerpo.

Esas manipulaciones en el cuerpo de cláusula repercutirán drásticamente en la computación/propagación de los grados de verdad cuando resolvemos objetivos contra los programas transformados.

Ilustremos todo ello con el siguiente ejemplo.

Ejemplo 4.2.2. *Considemos de nuevo el programa $\mathcal{P}$ mostrado en el Ejemplo 2.2.6. Es sencillo ver que el desplegado del programa $\mathcal{P}$ con respecto a la cláusula $\mathcal{C}_2$ (explotando la segunda regla de la Definición 2.2.4) genera el nuevo programa*

$$\mathcal{P}' = (\mathcal{P} - \{\mathcal{C}_2\}) \cup \{\mathcal{C}_{25}\},$$

donde $\mathcal{C}_{25}$ es la nueva regla desplegada:

$$q(a,b) \leftarrow 0.9 \quad \textit{with } 0.7$$

Por otra parte, si deseamos desplegar ahora la cláusula $\mathcal{C}_1$ en el programa $\mathcal{P}'$, debemos generar primeramente el siguiente par de pasos de derivación-SLD difusa (que sólo usa la primera regla de la Definición 2.2.4):

$$\langle \underline{q(X,Y)}, r(Y); id\rangle \quad \rightarrow_{FR1}{}^{\mathcal{C}_{25}} \langle \boxed{\mathrm{L}_{0.7}}\ 0.9\ \boxed{\mathrm{R}_{0.7}}, r(b); \{X/a, Y/b\}\rangle$$

$$\langle \underline{q(X,Y)}, r(Y); id\rangle \quad \rightarrow_{FR1}{}^{\mathcal{C}_{3}} \langle \boxed{\mathrm{L}_{0.8}}\ r(Y_1)\ \boxed{\mathrm{R}_{0.8}}, r(a); \{X/Y_1, Y/a\}\rangle$$

Así, resulta el nuevo programa desplegado $\mathcal{P}'' = (\mathcal{P}' - \{\mathcal{C}_1)\} \cup \mathcal{U}$ *donde* $\mathcal{U}$ *contiene las reglas:*

$$\mathcal{C}_{125}: \quad p(a) \leftarrow \boxed{\mathrm{L}_{0.7}}\ 0.9\ \boxed{\mathrm{R}_{0.7}}, r(b) \qquad \textit{with}\ 0.8$$

$$\mathcal{C}_{13}: \quad p(Y_1) \leftarrow \boxed{\mathrm{L}_{0.8}}\ r(Y_1)\ \boxed{\mathrm{R}_{0.8}}, r(a) \qquad \textit{with}\ 0.8$$

Además, ejecutando otro paso de resolución con la segunda regla de la Definición 2.2.4 en el cuerpo de la cláusula $\mathcal{C}_{125}$*, obtenemos la nueva regla desplegada*

$$p(a) \leftarrow \boxed{\mathrm{L}_{0.7}}\ 0.9\ \boxed{\mathrm{R}_{0.7}}, 0.7 \quad \textit{with}\ 0.8$$

Si damos ahora un posterior paso de desplegado con la tercera regla de la Definición 2.2.4 nos conduce a

$$p(a) \leftarrow 0.63, 0.7 \quad \textit{with}\ 0.8$$

que, finalmente, se convierte en

$$p(a) \leftarrow 0.63 \quad \textit{with}\ 0.8$$

después del último paso de desplegado dado con la cuarta regla de la Definición 2.2.4.

Es importante tener en cuenta que la aplicación de esta última regla al objetivo $\leftarrow p(X)$ *simula el efecto de los seis primeros pasos de computación mostrados en la derivación del Ejemplo 2.2.6, lo que evidencia la mejora alcanzada tras el desplegado.*

Los sistemas de transformación basados en plegado/desplegado clásico optimizan programas devolviendo código en el mismo lenguaje fuente del programa original, pero el desplegado ha jugado también un papel importante en el diseño de compiladores (ver [Julián y Villamizar, 2004]) generando código escrito en lenguaje objeto (normalmente de más bajo nivel que el código fuente).

En este sentido, nuestra transformación de desplegado puede verse como una técnica mixta que optimiza programas f-Prolog y los compila a programas lf-Prolog, con la ventaja (sorprendente) en nuestro caso de que ambos programas son ejecutables exactamente con el mismo principio operacional.

4.3. Transformaciones basadas en desplegado de programas lef-Prolog

Nuestro principal objetivo en esta sección consiste en la definición de dos transformaciones basadas en desplegado sobre un contexto difuso más general que el considerado

en la sección anterior a la hora de formular la Definición 4.2.1. En particular, además de definir una variante difusa de la transformación de desplegado de Tamaki y Sato [1984], introducimos una nueva transformación, que llamamos *reemplazamiento de t-norma*, que opera sobre la componente difusa de una expresión y tiene semejanzas con el reemplazamiento algebraico de Burstall y Darlington [1977]. Todo ello pasará por el enriquecimiento del lenguaje de base.

Como veremos, la adaptación de la regla de desplegado clásica de programación lógica pura a la programación lógica difusa no resulta simple si realmente deseamos definirla de forma natural.

En concreto, necesitamos capturar la dimensión "funcional" (gestión de los grados de verdad) asociada a la componente difusa y, lo que es más importante, resulta crucial la necesidad de utilizar un lenguaje intermedio (no necesario en otros contextos no difusos) para la codificación de los programas que se obtienen tras el proceso de transformación.

En la Definición 4.2.1 (ver [Julián *et al.*, 2004b]) introducíamos, por primera vez, la regla de desplegado en el contexto borroso; en lo que sigue, desarrollamos la primera aproximación descrita en la literatura para la introducción de un sistema de transformación difuso. La incorporación de las reglas de reemplazamiento de t-norma mejora ampliamente el desplegado estudiado en la sección anterior en los siguientes puntos:

i) Lenguaje: Usamos una extensión enriquecida del lenguaje difuso descrito en [Vojtáš y Paulík, 1996] que afecta directamente a todas las definiciones, resultados y pruebas, y que describimos en [Julián *et al.*, 2004a].

ii) Transformaciones de desplegado difuso: Presentamos ahora un esquema de transformaciones más claro al contemplar dos reglas basadas en el desplegado, cada una de ellas dirigida a una componente del lenguaje: mientras que la regla de desplegado difuso afecta a la componente lógica, la regla de reemplazamiento de t-norma realiza cómputos con los grados de verdad, es decir, afecta a la componente difusa del lenguaje.

   Además, el efecto obtenido por (el tipo cuatro de) la transformación de reemplazamiento de t-norma no puede realizarse por la definición de desplegado de la sección anterior recogida en [Julián *et al.*, 2004b].

iii) Propiedades: Obtendremos el resultado de corrección total fuerte y justificaremos la reducción en la longitud de las derivaciones-SLD difusas (cuando se

computan en los programas transformados) [Julián *et al.*, 2005c].

El desplegado difuso y las reglas reemplazamiento de t-norma constituirán un sistema básico de transformaciones que definimos en lo que sigue, y para las que consideramos los resultados de corrección en el próximo capítulo.

4.3.1. Desplegado difuso de programas lef-Prolog

A continuación, consideramos que $(\mathcal{P}_0, \ldots, \mathcal{P}_k)$ es una secuencia de programas que parte de un programa ef-Prolog inicial $\mathcal{P}_0$ y los restantes son todos programas transformados lef-Prolog tal que cada uno de ellos, excepto el inicial $\mathcal{P}_0$, es obtenido del inmediato precedente aplicando desplegado difuso.

Definición 4.3.1 (Desplegado Difuso)**.** *Sea* $(\mathcal{P}_0, \ldots, \mathcal{P}_k)$ *una secuencia de programas* lef-Prolog *que parte del programa* ef-Prolog $\mathcal{P}_0$*. Supongamos que*

$$\mathcal{C} \equiv (H \leftarrow \overline{B} \textit{ with } \Psi) \in \mathcal{P}_k$$

es una cláusula de programa (no unitaria) lef-Prolog*, con* $\Psi \equiv \langle p, \dot{\text{et}}_1, \dot{\text{et}}_2 \rangle$*, y* A *es un átomo incluido en el cuerpo* $\overline{B}$*.*

Entonces, el siguiente programa $\mathcal{P}_{k+1}$*, de la secuencia de trasformación, puede obtenerse por desplegado difuso de la cláusula* $\mathcal{C}$ *sobre un átomo* A *en el programa* $\mathcal{P}_k$ *como sigue:*

$$\mathcal{P}_{k+1} = (\mathcal{P}_k - \{\mathcal{C}\}) \cup \{H\sigma \leftarrow \overline{B'} \textit{ with } \Psi \mid \langle \boxed{\text{L}_\Psi}\, \overline{B}\, \boxed{\text{R}_\Psi}; id\rangle \rightarrow_{FR}{}^{A} \langle \boxed{\text{L}_\Psi}\, \overline{B'}\, \boxed{\text{R}_\Psi}; \sigma\rangle\}$$

Tengamos en cuenta algunas observaciones acerca de esta definición. El programa original $\mathcal{P}_0$ es un programa ef-Prolog, mientras que los restantes programas desplegados de la secuencia, por incorporar las etiquetas referidas, son todos lef-Prolog.

De manera análoga al caso clásico (y como también ocurriera con los lenguajes f-Prolog y lf-Prolog anteriores), cuya regla de desplegado basada en resolución-SLD se presenta en [Tamaki y Sato, 1984], las sustituciones computadas en los pasos de resolución durante el desplegado son incorporadas a las reglas transformadas de manera natural, esto es, aplicándolas a la cabeza de la cláusula.

Por otra parte, la propagación de los grados de verdad en este nuevo proceso de desplegado difuso se realiza en dos niveles diferentes: por asignación directa de la tupla Ψ (que contiene el grado de verdad p y etiquetas para $\dot{\text{et}}_1$ y $\dot{\text{et}}_2$) de la cláusula original a la cláusula transformada, y por la introducción de marcas y/o números reales en su cuerpo (así se hizo también para f-Prolog y lf-Prolog).

Sin embargo, destacamos que –a diferencia de lo que ocurría con el desplegado sobre un programa f-Prolog que introducimos en [Julián *et al.*, 2004b] y abordamos en la sección anterior– en esta formulación siempre se explota un átomo. Merece ser destacado, asimismo, que en la programación lógica pura es equivalente explotar un átomo del cuerpo de una cláusula a dar un paso de resolución (en el cuerpo). Ahora, en la programación lógica difusa, se pierde esa equivalencia: si explotamos un átomo aplicamos la Definición 4.3.1 de desplegado y si damos un paso de resolución aplicamos la Definición 4.2.1.

El siguiente ejemplo ilustra estos aspectos.

Ejemplo 4.3.2. *Consideremos de nuevo el conjunto de cláusulas del Ejemplo 2.3.7 como el programa inicial, $\mathcal{P}_0$, de una secuencia de transformación.*

Es sencillo comprobar que el desplegado de la cláusula $\mathcal{C}_2$ con respecto a $\mathcal{P}_0$ (explotando la segunda regla de la Definición 2.3.5) genera el nuevo programa

$$\mathcal{P}_1 = (\mathcal{P}_0 - \{\mathcal{C}_2\}) \cup \{\mathcal{C}_6\},$$

siendo $\mathcal{C}_6$ es la nueva regla desplegada:

$$q(a,b) \leftarrow 0.9 \quad \textit{with } \langle 0.7, \texttt{product}, \texttt{void} \rangle$$

Por otra parte, si deseamos desplegar ahora la cláusula $\mathcal{C}_1$ en $\mathcal{P}_1$, debemos generar primeramente los dos pasos de derivación-SLD difusa que siguen[1]*:*

$$\langle \boxed{\mathrm{L}_{\Psi_1}}\ \underline{q(X,Y)}, r(Y)\ \boxed{\mathrm{R}_{\Psi_1}}; id \rangle \quad \rightarrow_{FR1}{}^{\mathcal{C}_6} \quad \langle \boxed{\mathrm{L}_{\Psi_1}}\ \boxed{\mathrm{L}_{\Psi_6}}\ 0.9\ \boxed{\mathrm{R}_{\Psi_6}}, r(b)\ \boxed{\mathrm{R}_{\Psi_1}}; \sigma_1 \rangle$$

$$\langle \boxed{\mathrm{L}_{\Psi_1}}\ \underline{q(X,Y)}, r(Y)\ \boxed{\mathrm{R}_{\Psi_1}}; id \rangle \quad \rightarrow_{FR1}{}^{\mathcal{C}_3} \quad \langle \boxed{\mathrm{L}_{\Psi_1}}\ \boxed{\mathrm{L}_{\Psi_3}}\ r(Y_1)\ \boxed{\mathrm{R}_{\Psi_3}}, r(a)\ \boxed{\mathrm{R}_{\Psi_1}}; \sigma_2 \rangle$$

donde

$$\begin{aligned} \Psi_1 &\equiv \langle 0.8, \texttt{prod}, \texttt{min} \rangle \\ \Psi_6 &\equiv \langle 0.7, \texttt{prod}, \texttt{void} \rangle \\ \Psi_3 &\equiv \langle 0.8, \texttt{luka}, \texttt{void} \rangle \end{aligned}$$

[1] Ambos pasos se ejecutan con la Regla 1 de la Definición 2.3.5 (el primero usa la cláusula C_6 y el segundo la cláusula C_3) y explotan el mismo átomo $q(X,Y)$ –subrayado–, que (asumimos) selecciona la correspondiente regla de computación.

son los triples correspondientes a las cláusulas $\mathcal{C}_1$, $\mathcal{C}_6$ y $\mathcal{C}_3$ respectivamente; además los unificadores son:

$$\sigma_1 = \{X/a, Y/b\}$$

$$\sigma_2 = \{X/Y_1, Y/a\}$$

En consecuencia, resulta el programa desplegado $\mathcal{P}_2 = (\mathcal{P}_1 - \{\mathcal{C}_1\}) \cup \mathcal{U}$ donde $\mathcal{U}$ contiene las cláusulas:

$$\mathcal{C}_7 : p(a) \leftarrow \boxed{\mathrm{L}_{\langle 0.7,\mathtt{prod},\mathtt{max}\rangle}}\ 0.9\ \boxed{\mathrm{R}_{\langle 0.7,\mathtt{prod},\mathtt{max}\rangle}}, r(b) \qquad \textit{with } \langle 0.8, \mathtt{prod}, \mathtt{min}\rangle$$

$$\mathcal{C}_8 : p(Y_1) \leftarrow \boxed{\mathrm{L}_{\langle 0.8,\mathtt{luka},\mathtt{min}\rangle}}\ r(Y_1)\ \boxed{\mathrm{R}_{\langle 0.8,\mathtt{luka},\mathtt{min}\rangle}}, r(a) \quad \textit{with } \langle 0.8, \mathtt{prod}, \mathtt{min}\rangle$$

Por último, ejecutando un nuevo paso de resolución con la segunda regla de la Definición 2.3.5 sobre el átomo $r(b)$ del cuerpo de la cláusula $\mathcal{C}_7$, obtenemos el programa desplegado

$$\mathcal{P}_3 = \{\mathcal{C}_3, \mathcal{C}_4, \mathcal{C}_5, \mathcal{C}_6, \mathcal{C}_8, \mathcal{C}_9\}$$

(obsérvese que las cláusulas $\mathcal{C}_1, \mathcal{C}_2$ y $\mathcal{C}_7$ han sido eliminadas después de haber sido desplegadas) siendo

$$\mathcal{C}_9 \equiv p(a) \leftarrow \boxed{\mathrm{L}_{\langle 0.7,\mathtt{prod},\mathtt{max}\rangle}}\ 0.9\ \boxed{\mathrm{R}_{\langle 0.7,\mathtt{prod},\mathtt{max}\rangle}}, 0.7 \quad \textit{with } \langle 0.8, \mathtt{prod}, \mathtt{min}\rangle$$

Es importante tener en cuenta que la aplicación de esta última regla al objetivo "$\leftarrow p(X), r(a)$ with `min`*" simula el efecto de los cuatro primeros pasos de resolución mostrados en la derivación del Ejemplo 2.3.7, lo que evidencia la mejora alcanzada en la transformación de programas por desplegado.*

4.3.2. Reemplazamiento de t-norma

Aunque hemos visto en los apartados anteriores que las acciones de la transformación de desplegado repercuten en la computación/propagación de los grados de verdad sobre el cuerpo de cláusulas transformadas cuando resolvemos objetivos contra los programas transformados, la información acumulada ('compilada') en el cuerpo de la regla desplegada admite manipulaciones numéricas significativas para eliminar, o al menos simplificar, marcas y números reales.

La siguiente transformación realiza esta tarea de una manera similar a las reglas 3 y 4 de la Definición 2.3.5.

Definición 4.3.3 (Reemplazamiento de t-norma). *Sea* $(\mathcal{P}_0, \ldots, \mathcal{P}_k)$ *una secuencia de transformación de programas* lef-Prolog *que parte de un programa* ef-Prolog $\mathcal{P}_0$. *Supongamos que*

$$\mathcal{C} \equiv (H \leftarrow \overline{B} \textit{ with } \Psi) \in \mathcal{P}_k$$

es una cláusula de programa (no unitaria) lef-Prolog, *tal que* $\Psi \equiv \langle p, \dot{\text{et}}_1, \dot{\text{et}}_2 \rangle$. *Entonces, el siguiente programa* $\mathcal{P}_{k+1}$ *de la secuencia de transformación puede obtenerse por reemplazamiento de t-norma de la cláusula* $\mathcal{C}$ *en el programa* $\mathcal{P}_k$ *como sigue:*

$$\mathcal{P}_{k+1} = (\mathcal{P}_k - \{\mathcal{C}\}) \cup \{H \leftarrow \overline{B'} \textit{ with } \Psi'\},$$

tal que :

1. *si* $\overline{B} \equiv (\overline{X}, \boxed{\mathrm{L}_{\langle p', \dot{\text{et}}_1', \dot{\text{et}}_2' \rangle}}\, r \, \boxed{\mathrm{R}_{\langle p', \dot{\text{et}}_1', \dot{\text{et}}_2' \rangle}}, \overline{Y})$, *donde* r *es un número real, entonces*
   $\overline{B'} \equiv (\overline{X}, \dot{\text{et}}_1'(p', r), \overline{Y})$ *y* $\Psi' \equiv \Psi$.

2. *si* $\overline{B} \equiv (\overline{X}, \boxed{\mathrm{L}_{\langle p', \dot{\text{et}}_1', \dot{\text{et}}_2' \rangle}}\, \overline{r} \, \boxed{\mathrm{R}_{\langle p', \dot{\text{et}}_1', \dot{\text{et}}_2' \rangle}}, \overline{Y})$, *donde* $\overline{r} \equiv r_1, .., r_n$ $(n > 1)$ *son números reales, entonces*
   $\overline{B'} \equiv (\overline{X}, \boxed{\mathrm{L}_{\langle p', \dot{\text{et}}_1', \dot{\text{et}}_2' \rangle}}\, \dot{\text{et}}_2'(\overline{r}) \boxed{\mathrm{R}_{\langle p', \dot{\text{et}}_1', \dot{\text{et}}_2' \rangle}}, \overline{Y})$ *y* $\Psi' \equiv \Psi$.

3. *si* $\overline{B} \equiv r$, *donde* r *es un número real, entonces* $\overline{B'}$ *es vacío y* $\Psi' \equiv \dot{\text{et}}_1(p, r)$.

4. *si* $\overline{B} \equiv \overline{r}$, *donde* $\overline{r} \equiv r_1, \ldots, r_n$ $(n > 1)$ *son números reales, entonces*
   $\overline{B'} \equiv \dot{\text{et}}_2(\overline{r})$ *y* $\Psi' \equiv \langle p, \dot{\text{et}}_1, \mathit{void} \rangle$.

La definición anterior recuerda el llamado "reemplazamiento algebraico" usado tradicionalmente en los sistemas de transformación de programas funcionales puros [Burstall y Darlington, 1977; Pettorossi y Proietti, 1996b], donde una expresión funcional, en una regla de programa, es reemplazada por otra expresión equivalente con respecto a una propiedad algebraica dada.

Por ejemplo, la regla funcional pura $\mathtt{f(X,Y)} \rightarrow \mathtt{X + Y}$ puede ser transformada por reemplazamiento algebraico en $\mathtt{f(X,Y)} \rightarrow \mathtt{Y + X}$, gracias a la conmutatividad de la suma.

Obsérvese también que, análogamente a muchas otras reglas de transformación de la programación funcional (incluyendo el reemplazamiento algebraico y el desplegado funcional puro), nuestra reemplazamiento de t-norma reemplaza una cláusula

original por (solamente) una nueva, lo que contrasta, en general, con la regla de desplegado definida para programas lógicos puros y/o lógicos difusos.

No sorprenden estos hechos debido al carácter funcional de la componente difusa de nuestro lenguaje, en el sentido en que las manipulaciones numéricas ejecutadas por las reglas 3 y 4 de la Definición 2.3.5 recuerdan las evaluaciones funcionales.

Esta cualidad se hereda en la Definición 4.3.3, en la que consideramos algunos ejemplos particulares de reemplazamiento algebraico, pero esta vez exclusivamente enfocados a operaciones con t-normas.

Por esta razón, en nuestro entorno difuso, decidimos usar el nombre de "reemplazamiento de t-norma" en lugar de "reemplazamiento algebraico" para designar a este tipo de transformaciones.

Por otra parte, la transformación de reemplazamiento de t-norma tiene semejanzas con el desplegado difuso. En efecto, observemos que la siguiente definición de desplegado subsume el primer tipo de reemplazamiento de t-norma:

$$\mathcal{P}_{k+1} = (\mathcal{P}_k - \{\mathcal{C}\}) \cup \{H \leftarrow \overline{B'} \text{ with } \Psi \mid \langle \boxed{\mathrm{L}_\Psi}\ \overline{B}\ \boxed{\mathrm{R}_\Psi}; id\rangle \rightarrow_{FR3} \langle \boxed{\mathrm{L}_\Psi}\ \overline{B'}\ \boxed{\mathrm{R}_\Psi}; \sigma\rangle\},$$

donde un simple paso de resolución-SLD difusa de tipo 3 ha sido aplicado para ejecutar un reemplazamiento de t-norma del tipo 1.

Observemos también que, reemplazando $\rightarrow_{FR3}$ por $\rightarrow_{FR4}$ en la definición de desplegado anterior, obtenemos una alternativa al segundo y cuarto tipo de reemplazamiento de t-norma (dependiendo de cuál es la subexpresión explotada, referida al cálculo de la t-norma de et_2, procediendo sobre $\overline{B}$ o sobre la expresión completa $\boxed{\mathrm{L}_\Psi}\ \overline{B}\ \boxed{\mathrm{R}_\Psi}$).

En cualquier caso, el reemplazamiento de t-norma de tipo 3 nunca había sido considerado en la literatura, ni implícita ni explícitamente (ha sido propuesto por primera vez como trabajo futuro en [Julián *et al.*, 2004b]), y su aplicación permite transformar una cláusula de programa con cuerpo en un hecho (esto es, una cláusula con cuerpo vacío) actuando, por primera vez, sobre el grado de verdad de la regla transformada.

Observemos que ninguna otra transformación clásica ni difusa tiene esta cualidad que indirectamente supone que, en el mejor de los casos, aunque los programas transformados (por reemplazamiento de t-norma) pertenecen a la clase lef-Prolog, una vez trasformados por el reemplazamiento de t-norma de tipo 3, pueden recuperar la sintaxis ef-Prolog.

El siguiente ejemplo ilustra la aplicación de las transformaciones basadas en reemplazamiento de t-norma y alguna de sus ventajas.

Ejemplo 4.3.4. *Continuemos con la secuencia de transformación iniciada en el Ejemplo 4.3.2 ejecutando ahora alguna de las reglas de reemplazamiento de t-norma.*

Por el reemplazamiento de t-norma de tipo 1 sobre la cláusula $\mathcal{C}_9$ *obtenemos, ya que* $\texttt{prod}(0.9, 0.7) = 0.63$*, el siguiente programa*

$$\mathcal{P}_4 = (\mathcal{P}_3 - \{\mathcal{C}_9\}) \cup \{\mathcal{C}_{10}\},$$

donde

$$\mathcal{C}_{10} \equiv p(a) \leftarrow 0.63, 0.7 \quad \textit{with } \langle 0.8, \texttt{prod}, \texttt{min}\rangle.$$

Además, puesto que $\texttt{min}(0.63, 0.7) = 0.63$*, un reemplazamiento de t-norma de tipo 4 sobre esta última cláusula genera la nueva*

$$\mathcal{C}_{11} \equiv p(a) \leftarrow 0.63 \quad \textit{with } \langle 0.8, \texttt{prod}, \texttt{void}\rangle.$$

Finalmente la cláusula $\mathcal{C}_{11}$ *se transforma en el hecho*

$$\mathcal{C}_{12} \equiv p(a) \leftarrow \quad \textit{with } 0.504$$

(ya que $\texttt{prod}(0.63, 0.8) = 0.504$*) después del último reemplazamiento de t-norma de tipo 3.*

En consecuencia, el programa final es

$$\mathcal{P}_6 = \{\mathcal{C}_3, \mathcal{C}_4, \mathcal{C}_5, \mathcal{C}_6, \mathcal{C}_8, \mathcal{C}_{12}\}$$

y ahora la derivación mostrada en el Ejemplo 2.3.7 puede reducir su longitud en seis pasos gracias al uso de la cláusula $\mathcal{C}_{12}$*, que evidencia de nuevo la mejora alcanzada no sólo por desplegado, sino también por reemplazamiento de t-norma, en los programas transformados.*

4.4. Desplegado operacional de programas multi--adjuntos

En lo que sigue nos proponemos la adaptación de la regla de transformación de desplegado al caso de programas lógicos multi-adjuntos. Lo más relevante de esta

aportación lo constituye el hecho de que, por primera vez en la literatura, se contemplan técnicas de transformación de programas para esta clase de programas lógicos difusos que son mucho más generales y potentes que lo vistos hasta ahora.

En el capítulo siguiente, aportaremos los resultados que justifican la idoneidad de esta regla de transformación: la corrección total fuerte y su capacidad para originar mejoras significativas en la eficiencia de los programas residuales.

Notemos que los cuerpos de las $\mathcal{L}$-reglas desplegadas (que intuitivamente tienen la misma estructura de cualquier objetivo inicial, intermedio o final que aparezca en las derivaciones admisibles difusas) incluyen elementos del retículo (grados de verdad) y posiblemente conjunciones adjuntas $\&_i$.

Por otra parte, sabemos que la regla de desplegado consiste en esencia en la aplicación de un paso de computación sobre (el átomo seleccionado en) el cuerpo de una regla que, en este ámbito, corresponde a la aplicación de cualquiera de las tres reglas descritas en la Definición 3.1.1. Obsérvese que este proceso genera siempre, como venimos observando, reglas cuyos cuerpos incluyen grados de verdad y acaso conjunciones adjuntas. Por tanto, para que una transformación de desplegado basada en computaciones admisibles difusas preserve la estructura sintáctica de los $\mathcal{L}$-programas resulta imprescidible admitir que el lenguaje multi-adjunto $\mathcal{L}$ contiene elementos del retículo y conjunciones adjuntas (en tal caso las reglas desplegadas son efectivamente $\mathcal{L}$-reglas).

Definición 4.4.1. *Sea $\mathcal{P}$ un programa lógico multi-adjunto y $\mathcal{R} : \langle A \leftarrow_i \mathcal{B};\ v\rangle \in \mathcal{P}$ una regla (no unitaria) de $\mathcal{L}$. Entonces, el desplegado difuso del programa $\mathcal{P}$ con respecto a la regla $\mathcal{R}$ es el nuevo programa $\mathcal{P}' = (\mathcal{P} - \{\mathcal{R}\}) \cup \mathcal{U}$ tal que:*

$$\mathcal{U} \ = \ \{\langle A\sigma \leftarrow_i \mathcal{B}';\ v\rangle |\ \ \langle \mathcal{B}; id\rangle \rightarrow_{AS} \langle \mathcal{B}'; \sigma\rangle\}$$

Tal como ocurre con la regla de desplegado clásico basado en resolución-SLD presentada en [Tamaki y Sato, 1984], las sustituciones computadas en los pasos admisibles, en el proceso de desplegado, son incorporadas a las reglas transformadas de manera natural, esto es, aplicándolas en la cabeza de la regla. Por otra parte, la propagación de los grados de verdad se aborda también de manera sencilla: la regla desplegada hereda directamente el grado de verdad v de la la regla original.

Sin embargo, un análisis en profundidad de la transformación de desplegado muestra cómo el cuerpo de la regla transformada también incluye información compilada sobre ambos componentes de una respuesta computada difusa (esto es, el grado de verdad y la sustitución). En cuanto al grado de verdad, observamos que el

cuerpo de la regla transformada puede incluir: *i*) el símbolo $\bot$, cuando se ejecuta un paso admisible de tipo $AS3$; *ii*) un grado de verdad, cuando es la segunda regla la que interviene en el paso de desplegado, esto es, cuando el paso admisible es del tipo $AS2$; o *iii*) un grado de verdad junto con la conjunción adjunta si la primera regla es la involucrada en el paso de desplegado, es decir, si el paso admisible es del tipo $AS1$. En resumen, la propagación de los grados de verdad durante el desplegado se realiza en dos niveles diferentes:

1. asignando directamente el grado de verdad de la regla original a la regla transformada, e

2. introduciendo nuevos grados de verdad (de otras reglas o alternativamente $\bot$) y posiblemente conjunciones adjuntas en su cuerpo.

Estas manipulaciones en el cuerpo de la regla repercutirán drásticamente en la computación/propagación de los grados de verdad cuando resolvemos objetivos contra los programas transformados.

Ilustremos estos hechos con un ejemplo.

Ejemplo 4.4.2. *Consideremos de nuevo el programa $\mathcal{P}$ mostrado en el Ejemplo 3.1.5. Resulta sencillo apreciar que el desplegado de $\mathcal{P}$ con respecto a la regla $\mathcal{R}_2$ (explotando la segunda regla admisible de la Definición 3.1.1) genera el programa*

$$\mathcal{P}' = (\mathcal{P} - \{\mathcal{R}_2\}) \cup \{\mathcal{R}_{25}\}$$

donde $\mathcal{R}_{25}$ es la regla desplegada

$$q(a,b) \leftarrow_{\mathtt{prod}} 0.9 \quad \textit{with } 0.7$$

Además, si ahora deseamos desplegar la regla $\mathcal{R}_1$ en el programa $\mathcal{P}'$, debemos ejecutar previamente los siguientes pasos de derivación admisibles[2]*:*

$$\langle \underline{q(X,Y)} \wedge_{\mathtt{G}} r(Y); id\rangle \quad \rightarrow_{AS1}{}^{\mathcal{R}_{25}} \langle (0.7 \&_{\mathtt{prod}} 0.9) \wedge_{\mathtt{G}} r(b); \{X/a, Y/b\}\rangle$$

$$\langle \underline{q(X,Y)} \wedge_{\mathtt{G}} r(Y); id\rangle \quad \rightarrow_{AS1}{}^{\mathcal{R}_3} \langle (0.8 \&_{\mathtt{luka}} r(Y_1)) \wedge_{\mathtt{G}} r(a); \{X/Y_1, Y/a\}\rangle$$

[2] Notemos que los pasos de desplegado son siempre ejecutados usando una regla de computación fija. En este y en otros ejemplos usamos una regla de computación que selecciona los átomos del objetivo de izquierda a derecha, al igual que la implementada en Prolog.

Así, el programa desplegado es $\mathcal{P}'' = (\mathcal{P}' - \{\mathcal{R}_1\}) \cup \{\mathcal{R}_{125}, \mathcal{R}_{13}\}$*, donde:*

$$\mathcal{R}_{125}: \quad p(a) \leftarrow_{\mathtt{prod}} (0.7 \&_{\mathtt{prod}} 0.9) \wedge_{\mathtt{G}} r(b) \qquad \textit{with } 0.8$$

$$\mathcal{R}_{13}: \quad p(Y_1) \leftarrow_{\mathtt{prod}} (0.8 \&_{\mathtt{luka}} r(Y_1)) \wedge_{\mathtt{G}} r(a) \quad \textit{with } 0.8$$

Además, ejecutando un nuevo paso admisible con la segunda regla de la Definición 3.1.1 sobre el cuerpo de la regla $\mathcal{R}_{125}$*, obtenemos la regla desplegada*

$$p(a) \leftarrow_{\mathtt{prod}} (0.7 \&_{\mathtt{prod}} 0.9) \wedge_{\mathtt{G}} 0.7 \quad \textit{with } 0.8$$

La aplicación de esta última regla al objetivo propuesto en el Ejemplo 3.1.5 simula el efecto de los cuatro primeros pasos admisibles mostrados en la derivación del mismo ejemplo, que evidencia la mejora alcanzada por el desplegado sobre los programas transformados.

El objetivo más importante de la transformación de desplegado, además de preservar la semántica del programa, es optimizar el código, independentemente del lenguaje objeto.

4.5. Desplegado interpretativo de programas multi-adjuntos

En el marco de la programación lógica multi-adjunta, el proceso de desplegado puede ser mejor entendido si, a semejanza de las dos fases separadas de la semántica procedural subyacente del lenguaje lógico multi-adjunto, distinguimos entre pasos de desplegado operacional y pasos de desplegado interpretativo.

Por ello deseamos contemplar dos tipos de desplegado fuertemente relacionados: el operacional (primeramente introducido en [Julián *et al.*, 2005a] y ya abordado en la sección anterior) y el interpretativo (introducido por vez primera en [Julián *et al.*, 2006c]).

Definiremos a continuación una regla de desplegado basada en pasos interpretativos que permitirá acelerar el cálculo de los grados de verdad durante la segunda de las fases de la semántica procedural, la fase interpretativa.

Es importante observar que en el lenguaje (lógico multi-adjunto) original propuesto en [Medina *et al.*, 2004] y usado en [Julián *et al.*, 2005a], la fase interpretativa no está modelada en términos de un sistema de transición de estados, lo que impide

la definición/aplicación de la nueva regla de desplegado (interpretativo) por medio de pasos interpretativos. En [Julián *et al.*, 2006c]), salvando estas limitaciones, hemos probado que el desplegado interpretativo, aparte de exhibir propiedades de corrección fuerte análogas (es decir, que preserva la semántica de las sustituciones computadas y los grados de verdad) al desplegado operacional originariamente presentado en [Julián *et al.*, 2005a], permite simplificar y acelerar la fase interpretativa cuando resolvemos objetivos con respecto a un programa dado.

Por otra parte, la regla de reemplazamiento de t-norma introducida en [Julián *et al.*, 2005c], puede verse como un precedente primitivo de la de desplegado interpretativo. De hecho, las cuatro variantes de reemplazamiento de t-norma, también ejecutan manipulaciones de bajo nivel sobre expresiones borrosas que involucran operaciones con t-normas sobre reglas de programa. Sin embargo, la incorporación de esta nueva regla así como el diseño de la fase interpretativa como una fase procedural, mejora el contexto primitivo en los siguientes aspectos:

• Sobre un lenguaje muy potente (el lenguaje de programación lógica multi-adjunta) con una sintaxis simple y expresiva, logramos clarificar la semántica procedural (operacional/interpretativa) y, en general, una mejor formalización.

• Ahora, ni los pasos interpretativos ni el desplegado interpretativo dependen forzosamente (el orden en que se aplican los conectivos suele estar prefijado) de una función de selección (regla de computación), como ocurre con cualquier otra regla de transformación basada en desplegado difuso descrita en la literatura, puesto que el orden de evaluación de expresiones se fija por la semántica interpretativa.

• Como efecto colateral del punto anterior, y como veremos en el Teorema 5.5.1, es la primera vez que nuestro resultado de corrección total admite un esquema de demostración directo que no obliga a contemplar resultados instrumentales previos acerca de la independencia de algún tipo de regla de computación.

• Además, nuestras variantes difusas de las reglas de desplegado recuperan el lenguaje fuente que perdíamos en [Julián *et al.*, 2005a] y [Julián *et al.*, 2005c], donde eran preceptivos lenguajes auxiliares intermedios (con construcciones complejas e intricados artificios que no percibía el usuario final) para codificar programas residuales obtenidos después de ejecutar el desplegado operacional o el reemplazamiento de t-norma.

El principal objetivo de la presente sección es definir una regla de desplegado basada

en pasos interpretativos. Proponemos esta noción de desplegado interpretativo con la intención de facilitar la evaluación de los grados de verdad durante la fase interpretativa. En la Definición 3.1.6 hemos optado por una caracterización procedural de la fase interpretativa (formalizándola en términos de un sistema de transición de estados), evitando el uso de conceptos semánticos (que son necesarios, por ejemplo, en [Medina *et al.*, 2004] y [Julián *et al.*, 2005a]). Este hecho es preceptivo para clarificar la formalización de nuestra regla de desplegado interpretativo como sigue.

Definición 4.5.1 (Desplegado Interpretativo)**.** *Sea $\mathcal{P}$ un programa multi-adjunto y $\mathcal{R} : \langle A \leftarrow_i \mathcal{B};\ v\rangle \in \mathcal{P}$ una regla de programa (no unitaria). Entonces, el desplegado interpretativo de la regla $\mathcal{R}$ en el programa $\mathcal{P}$ con respecto al retículo $(L, \leq)$ asociado a $\mathcal{P}$ es el nuevo programa $\mathcal{P}' = (\mathcal{P} - \{\mathcal{R}\}) \cup \{\langle A \leftarrow_i \mathcal{B}';\ v'\rangle\}$ tal que:*

`IU1` . *Si la expresión $r_1 @ r_2$ aparece en $\mathcal{B}$ entonces $\mathcal{B}' = \mathcal{B}[r_1 @ r_2 / \dot{@}(r_1, r_2)]$, donde $@$ es una conectiva, y $v' = v$.*

`IU2` . *Si $\mathcal{B} = r$, donde $r \in L$, Entonces $\mathcal{B}'$ es vacío y $v' = \dot{\&}_i(v, r)$, donde $(\leftarrow_i, \&_i)$ es un par adjunto en $(L, \leq)$.*

Obsérvese que la primera variante de la regla de desplegado interpretativo `IU1`, consiste simplemente en aplicar un paso interpretativo sobre el cuerpo de una regla del programa. En este sentido, una formalización alternativa, más similar a la Definición 4.2.1, pero reemplazando el uso de $\rightarrow_{FR}$ por $\rightarrow_{IS}$, podría ser:

$$\mathcal{P}' = (\mathcal{P} - \{\mathcal{R}\}) \cup \mathcal{U}, \quad \text{donde } \mathcal{U} = \{\langle A \leftarrow_i \mathcal{B}';\ v\rangle |\ \langle \mathcal{B}; id\rangle \rightarrow_{IS} \langle \mathcal{B}'; id\rangle\}$$

De hecho, ambas formulaciones consisten simplemente en reemplazar una regla de programa $\mathcal{R}$ cuyo cuerpo contiene una connectiva $@$, por una regla análoga, con el mismo grado de verdad, pero con la función verdad de $@$ (con respecto al retículo asociado al programa) ya evaluada en su cuerpo.

En cualquier caso, es importante comparar la transformación `IU1` (en cualquiera de sus formatos alternativos), con la regla de reemplazamiento de t-norma planteada originariamente en [Julián *et al.*, 2005c] en los términos que recogemos en la Definición 4.3.3, puesto que esta nueva transformación compacta en una única formulación tres variantes de bajo nivel de la transformación primitiva.

Centrándonos en el caso `IU2`, debemos tener en cuenta que la segunda alternativa propuesta anteriormente para formalizar `IU1`, no puede ser aplicada ahora: no sólo el grado de verdad de la regla transformada difiere de la original sino que, lo que

es más interesante, la transformación `IU2` permite simplificar reglas de programa eliminando directamente sus cuerpos y, por tanto, produciendo hechos[3].

El siguiente ejemplo ilustra la aplicación del desplegado interpretativo y alguna de sus ventajas.

Ejemplo 4.5.2. *Ejecutamos ahora algunos pasos de desplegado interpretativo sobre las reglas obtenidas por desplegado operacional en el Ejemplo 4.4.2. Por desplegado interpretativo –de tipo* `IU1`*– de* $\mathcal{R}_9$ *(notemos que* $\dot{\&}_{\texttt{prod}}(0.9, 0.7) = 0.63$*) obtenemos la nueva regla desplegada*

$$\mathcal{R}_{10} : p(a) \leftarrow_{\texttt{prod}} 0.63 \&_{\texttt{G}} 0.7 \quad \textit{with } 0.8$$

Además, aplicando un nuevo paso de desplegado interpretativo de tipo `IU1` *en esta última regla, obtenemos*

$$\mathcal{R}_{11} : p(a) \leftarrow_{\texttt{prod}} 0.63 \quad \textit{with } 0.8$$

Finalmente, la regla $\mathcal{R}_{11}$ *se transforma en el hecho*

$$\mathcal{R}_{12} : p(a) \leftarrow_{\texttt{prod}} \quad \textit{with } 0.504$$

después de un último paso de desplegado interpretativo de tipo `IU2`*. Por tanto, el programa final es el conjunto de reglas*

$$\{\mathcal{R}_3, \mathcal{R}_4, \mathcal{R}_5, \mathcal{R}_6, \mathcal{R}_8, \mathcal{R}_{12}\}$$

y ahora la derivación mostrada en el Ejemplo 3.1.5 puede reducir su longitud en seis pasos gracias al uso de la cláusula $\mathcal{R}_{12}$*, como sigue:*

$$\begin{array}{ll}
\langle \underline{p(X)} \&_{\texttt{G}} r(a);\ id \rangle & \rightarrow_{AS2}{}^{\mathcal{R}_{12}} \\
\langle (0.504 \&_{\texttt{G}} \underline{r(a)});\ \{X/a\} \rangle & \rightarrow_{AS2}{}^{\mathcal{R}_4} \\
\langle \underline{0.504 \&_{\texttt{G}} 0.7}; \{X/a\} \rangle & \rightarrow_{IS} \\
\langle 0.504; \{X/a\} \rangle &
\end{array}$$

Obsérvese que hemos evitado tres pasos admisibles y tres pasos interpretarivos, debido al hecho de que la regla $\mathcal{R}_{12}$ *proviene de la regla* $\mathcal{R}_1$ *una vez que ésta ha sido modificada por tres operaciones de desplegado operacional más tres operaciones de desplegado interpretativo. De nuevo, se constata la mejora alcanzada por el uso combinado del desplegado operacional/interpretativo, sobre los programas transformados.*

[3] A esta regla se le llama "facting" en [Guerrero y Moreno, 2008].

4.6. Conclusiones

Las principales contribuciones que recoge este capítulo pueden resumirse como sigue:

- Por primera vez en la literatura, formalizamos en el contexto difuso (para distintos lenguajes) la noción clásica de desplegado, que hereda tanto la simplicidad como la potencia computacional de los lenguajes originales.
- En particular, hemos definido, el desplegado operacional para un lenguaje muy expresivo y potente en el contexto difuso: el lenguaje multi-adjunto.
- Como principales novedades (con respecto a otros contextos no difusos) introducimos las reglas de reemplazamiento de t-norma para (el lenguaje lef-Prolog) que aportan beneficios extra a esta tranformación a la hora de computar los grados de verdad.
- Hemos introducido la noción de desplegado interpretativo, que enriquece muy notablemente las reglas de reemplazamiento de t-norma contempladas para el lenguaje lef-Prolog, complementa el desplegado operacional de programas lógicos multi-adjuntos y permite reducir el número de pasos interpretativos en la ejecución de un objetivo.
- Esto último ha sido viable puesto que previamente hemos clarificado la semántica procedural del lenguaje subyacente, formalizando su fase interpretativa en términos de un sistema de transición de estados.
- Además, la formalización de los desplegados operacional e interpretativo del lenguaje multi-adjunto no depende de lenguajes auxiliares, lo que clarifica y enfatiza nuestra aportación.
- Se ilustran estas definiciones de desplegado con ejemplos representativos, que revelan propiedades de corrección y eficiencia de los programas transformados en los términos que posteriormente se acredita con pruebas formales.

Capítulo 5

Propiedades de corrección y eficiencia

Nos ocupamos en este capítulo de enunciar y demostrar los resultados obtenidos acerca de la corrección de las transformaciones que hemos visto en el capítulo anterior. Justificaremos que, más allá de la ganancia en eficiencia, las respuestas computadas del programa original y del que se obtiene por desplegado o como consecuencia de aplicar cualquiera de las transformaciones consideradas están fuertemente relacionadas entre sí.

En primer término, todos los resultados correspondientes al lenguaje lef-Prolog se pueden refundir en el Teorema 5.3.1 que enunciamos en [Julián *et al.*, 2004b] y probamos en [Julián *et al.*, 2004c], y en el Teorema 5.3.2 enunciado en [Julián *et al.*, 2004a] y probado en [Julián *et al.*, 2005c]. Omitimos toda mención acerca de los resultados análogos para los lenguajes f-Prolog y lf-Prolog Vojtáš, puesto que se obtienen inmediatamente de los teoremas citados.

Por lo que se refiere al lenguaje lenguaje multi-adjunto, los Teoremas 5.4.2, 5.4.3 y 5.4.4 enuncian los resultados de corrección del desplegado operacional, el Teorema 5.5.1 los correspondientes del desplegado interpretativo y, finalmente, el Teorema 5.6.1 los resultados de corrección del sistema de transformación determinado por la combinación de ambos tipos de desplegado.

Los resultados mencionados del desplegado operacional se recogen en [Julián *et al.*, 2005a], y los relativos al desplegado interpretativo y al sistema de transformación en [Julián *et al.*, 2005b] y en [Julián *et al.*, 2006c].

En lo que sigue, utilizaremos los términos de corrección parcial, completitud y corrección total[1] para traducir los correspondientes (más estandarizados y precisos de) *soundness, completeness y correctness.*

Es frecuente adjetivar con "fuerte" (*strong*) aquellos resultados que preservan la igualdad sintáctica de las respuestas computadas (restringidos al conjunto de variables del objetivo, como es nuestro caso), mientras que se usa "débil" (*weak*) o directamente no se adjetivan aquellos resultados que únicamente mantienen equivalencias a nivel de "igual o más general" ($\leq$) entre los observables considerados en los programas original y transformado.

5.1. Corrección parcial fuerte de transformaciones con lef-Prolog

Abordamos la corrección parcial del desplegado difuso y reemplazamiento de t-norma de programas lef-Prolog, por la que obtendremos que todas las respuestas computadas del programa transformado lo son del programa original.

El siguiente Lema, que puede considerarse análogo al Lema 3.1.9 (de conservación de sustituciones en las respuestas computadas o de conmutación de pasos de derivación) es auxiliar.

Intuitivamente, demuestra que, incluso en el caso en que dos pasos de derivación no pueden ser intercambiados puesto que el segundo explota una lf-expresión introducida en el objetivo considerado por el primero, su efecto (con respecto a las sustituciones de la respuesta computada difusa) puede ser simulado por un sólo paso ejecutado con una cláusula transformada obtenida por desplegado difuso.

Lema 5.1.1. *Sea $\mathcal{P}$ un programa* lef-Prolog, $\mathcal{G} \equiv \leftarrow \mathcal{Q}_0$ `with le2 =` $\dot{\text{e}}\text{t}_2^0$ *un objetivo* lef-Prolog, $\Psi_0 \equiv \langle 1, void, \dot{et}_2^0 \rangle$ *y $\mathcal{C}_1, \mathcal{C}_2 \ll \mathcal{P}$ dos reglas no unitarias. Entonces,*

$$\langle \boxed{\mathrm{L}_{\Psi_0}} \mathcal{Q}_0 \boxed{\mathrm{R}_{\Psi_0}} ; \theta_0 \rangle \rightarrow_{FR1}{}^{\mathcal{C}_1} \langle \boxed{\mathrm{L}_{\Psi_0}} \mathcal{Q}_1 \boxed{\mathrm{R}_{\Psi_0}} ; \theta_0\theta_1 \rangle \rightarrow_{FR}{}^{\mathcal{C}_2} \langle \boxed{\mathrm{L}_{\Psi_0}} \mathcal{Q}_2 \boxed{\mathrm{R}_{\Psi_0}} ; \theta_0\theta_1\theta_2 \rangle$$

donde el segundo paso explota un átomo introducido en $\mathcal{Q}_1$ después del primer paso,

si, y sólo si,

$$\langle \boxed{\mathrm{L}_{\Psi_0}} \mathcal{Q}_0 \boxed{\mathrm{R}_{\Psi_0}} ; \theta_0 \rangle \rightarrow_{FR1}{}^{\mathcal{C}_3} \langle \boxed{\mathrm{L}_{\Psi_0}} \mathcal{Q}_3 \boxed{\mathrm{R}_{\Psi_0}} ; \theta_0\theta_3 \rangle$$

[1] Esta terminología, aún a pesar de no ser completamente universal evita ambigüedades y está suficientemente popularizada en la literatura especializada escrita en castellano.

donde $\mathcal{C}_3$ se obtiene por desplegado de C_1 usando C_2 y $\theta_0\theta_1\theta_2 = \theta_0\theta_3$ $[\mathcal{V}ar(\mathcal{Q}_0)]$.

Demostración.
($\Rightarrow$) Sea $\mathcal{C}_1 \equiv (H_1 \leftarrow \overline{X_1}, A_1, \overline{Y_1}; \Psi)$ donde $\Psi \equiv \langle p, \dot{e}t_1, \dot{e}t_2 \rangle$, sea H_2 el átomo de la cabeza de la cláusula C_2 y sea $\mathcal{Q}_0 \equiv \overline{X}, A, \overline{Y}$ donde $\overline{X}, \overline{Y}, \overline{X_1}$ y $\overline{Y_1}$ son secuencias arbitrarias de lef-expresiones válidas y A el átomo seleccionado en $\mathcal{Q}_0$ por la regla de computación considerada.

Entonces, en la f-derivación

$$\langle \boxed{\mathrm{L}_{\Psi_0}} \mathcal{Q}_0 \boxed{\mathrm{R}_{\Psi_0}} ; \theta_0 \rangle \rightarrow_{FR1}{}^{\mathcal{C}_1} \langle \boxed{\mathrm{L}_{\Psi_0}} \mathcal{Q}_1 \boxed{\mathrm{R}_{\Psi_0}} ; \theta_0\theta_1 \rangle \rightarrow_{FR}{}^{\mathcal{C}_2} \langle \boxed{\mathrm{L}_{\Psi_0}} \mathcal{Q}_2 \boxed{\mathrm{R}_{\Psi_0}} ; \theta_0\theta_1\theta_2 \rangle$$

tenemos que:

$$\theta_1 = mgu(\{A = H_1\}), \qquad \mathcal{Q}_1 \equiv (\overline{X}, \boxed{\mathrm{L}_{\Psi}}\ \overline{X_1}, A_1, \overline{Y_1}\ \boxed{\mathrm{R}_{\Psi}}, \overline{Y})\theta_1.$$

Además, si $A_1\theta_1$ es el átomo seleccionado en $\mathcal{Q}_1$ por la regla de computación empleada, sea $\theta_2 = mgu(\{A_1\theta_1 = H_2\})$.

Consideremos ahora $\sigma = mgu(\{A_1 = H_2\})$. Entonces, se satisfacen las siguientes igualdades:

$\theta_1\theta_2$	$=$	
$\theta_1 mgu(\{A_1\theta_1 = H_2\})$	$=$	(ya que $Dom(\theta_1) \cap \mathcal{V}ar(\mathcal{C}_2) = \varnothing$)
$\theta_1 mgu(\widehat{mgu}(\{A_1 = H_2\})\theta_1)$	$=$	
$\theta_1 mgu(\widehat{\sigma}\theta_1)$	$=$	(por el Lemma 2.4.1)
$\theta_1 \Uparrow \sigma$	$=$	(por el Lemma 2.4.1)
$\sigma mgu(\widehat{\theta_1}\sigma)$	$=$	
$\sigma mgu(\widehat{mgu}(\{A = H_1\})\sigma)$	$=$	(ya que $Dom(\sigma) \cap \mathcal{V}ar(\mathcal{Q}_0) = \varnothing$)
$\sigma mgu(\{A = H_1\sigma\})$		

Además, puesto que $\theta_1\theta_2 \not\equiv fallo$, entonces $\sigma \not\equiv fallo$ y por tanto existe una cláusula $\mathcal{C}_3$ obtenida por desplegado de (el átomo A_1 en el cuerpo de) $\mathcal{C}_1$ usando $\mathcal{C}_2$, tal que la cabeza de $\mathcal{C}_3$ es el átomo $H_1\sigma$.
Entonces, como $mgu(\{A = H_1\sigma\}) \not\equiv fallo$, puede ser ejecutado el siguiente paso de resolución-SLD difusa dado en el átomo seleccionado A de $\mathcal{Q}_0 \equiv \overline{X}, A, \overline{Y}$:

$$\langle \boxed{\mathrm{L}_{\Psi_0}} \mathcal{Q}_0 \boxed{\mathrm{R}_{\Psi_0}} ; \theta_0 \rangle \rightarrow_{FR1}{}^{\mathcal{C}_3} \langle \boxed{\mathrm{L}_{\Psi_0}} \mathcal{Q}_3 \boxed{\mathrm{R}_{\Psi_0}} ; \theta_0\theta_3 \rangle,$$

donde $\theta_3 = mgu(\{A = H_1\sigma\})$.

Por último, ya que $\theta_1\theta_2 = \sigma\theta_3$, se tiene $\theta_0\theta_1\theta_2 = \theta_0\sigma\theta_3$, y como $Dom(\sigma) \cap \mathcal{V}ar(\mathcal{Q}_0) = \varnothing$ y $Dom(\sigma) \cap Dom(\theta_0) = \varnothing$, tenemos que

$$\theta_0\theta_1\theta_2 = \theta_0\theta_3 \ [\mathcal{V}ar(\mathcal{Q}_0)],$$

como deseábamos probar.

($\Leftarrow$) Este recíproco puede probarse de manera similar al directo, explotando de nuevo la equivalencia entre $\theta_1\theta_2$ y $\sigma\theta_3$.

□

Teorema 5.1.2 (Corrección parcial fuerte)**.** *Sea $\mathcal{P}$ un programa* lef-Prolog*, $\mathcal{G} \equiv \leftarrow \mathcal{Q}_0$ with $le_2 = \dot{\mathrm{et}}_2^0$ un objetivo* lef-Prolog *y sea $\Psi_0 \equiv \langle 1, void, \dot{et}_2^0 \rangle$. Si $\mathcal{P}'$ es un programa* lef-Prolog *obtenido por desplegado difuso o por reemplazamiento de t-norma de $\mathcal{P}$, entonces*

$$\langle \boxed{\mathrm{L}_{\Psi_0}} \mathcal{Q}_0 \boxed{\mathrm{R}_{\Psi_0}} ; id \rangle \rightarrow_{FR}{}^* \langle r; \theta \rangle \ \textit{en } \mathcal{P},$$

si

$$\langle \boxed{\mathrm{L}_{\Psi_0}} \mathcal{Q}_0 \boxed{\mathrm{R}_{\Psi_0}} ; id \rangle \rightarrow_{FR}{}^* \langle r; \theta' \rangle \ \textit{en } \mathcal{P}',$$

donde $\theta = \theta'[\mathcal{V}ar(\mathcal{Q}_0)]$.

Demostración. Sea

$$\mathcal{D}' : [\langle \boxed{\mathrm{L}_{\Psi_0}} \mathcal{Q}_0 \boxed{\mathrm{R}_{\Psi_0}} ; id \rangle \rightarrow_{FR}{}^* \langle r; \theta \rangle]$$

una derivación (genérica) para $\mathcal{G}$ en $\mathcal{P}'$ que nos proponemos simular construyendo una nueva derivación $\mathcal{D}$ en $\mathcal{P}$.

La construcción de $\mathcal{D}$ se realizará por inducción en la longitud n de $\mathcal{D}'$. Puesto que el caso base, $n = 0$, es trivial, procedemos con el caso general cuando $n > 0$. Entonces,

$$\mathcal{D}' : [\langle \boxed{\mathrm{L}_{\Psi_0}} \mathcal{Q}_0 \boxed{\mathrm{R}_{\Psi_0}} ; id \rangle \rightarrow_{FR} \langle \boxed{\mathrm{L}_{\Psi_0}} \mathcal{Q}' \boxed{\mathrm{R}_{\Psi_0}} ; \vartheta \rangle \rightarrow_{FR}{}^* \langle r; \theta' \rangle]$$

Si el primer paso de $\mathcal{D}'$ ha sido dado con la segunda, tercera o cuarta regla de la Definición 2.3.5, o, incluso si ha sido ejecutado con la primera pero usando una cláusula perteneciente a $\mathcal{P}$, el resultado se sigue por la hipótesis de inducción.

En otro caso, el paso inicial se da con la regla 1 usando una cláusula $\mathcal{C}'$ que ha sido obtenida por desplegado o por reemplazamiento de t-norma de otra cláusula $\mathcal{C} \in \mathcal{P}$.

Puesto que el paso de desplegado ha sido ejecutado con una de las dos reglas de la Definición 2.3.5, y el paso de reemplazamiento de t-norma ha sido ejecutado con una de las cuatro reglas de la Definición 4.3.3 tratamos cada caso separadamente.

1. Desplegado basado en Regla 1 de la Definición 2.3.5.
   Sea
   $$\mathcal{C} \equiv (H_1 \leftarrow \overline{X_1}, A_1, \overline{Y_1}; \Psi) \in \mathcal{P} \text{ with } \Psi \equiv \langle p, \dot{\text{et}}_1, \dot{\text{et}}_2 \rangle$$
   y
   $$\mathcal{C}_2 \equiv (H_2 \leftarrow \overline{B_2}; \Psi') \in \mathcal{P} \quad \text{with } \Psi' \equiv \langle q, \dot{\text{et}}'_1, \dot{\text{et}}'_2 \rangle$$
   tal que, desplegando $\mathcal{C}$ con respecto a $\mathcal{C}_2$ usando la Definición 4.2.1, obtenemos la cláusula
   $$\mathcal{C}' \equiv ((H_1 \leftarrow \overline{X_1}, \boxed{\text{L}_{\Psi'}}\ \overline{B_2}\ \boxed{\text{R}_{\Psi'}}\ , \overline{Y_1})\sigma; \Psi) \in \mathcal{P}'$$
   Supongamos que $\mathcal{Q}_0 \equiv \overline{X}, A, \overline{Y}$ donde $\overline{X}$ y $\overline{Y}$ son secuencias arbitrarias de lef-expresiones válidas y A es el átomo seleccionado. Entonces, $\mathcal{D}'$ tiene la siguiente forma:
   $$\begin{array}{ll} \langle \overline{X}, A, \overline{Y}; id \rangle & \rightarrow_{FR1}{}^{\mathcal{C}'} \\ \langle (\overline{X}, \boxed{\text{L}_\Psi}\ \overline{X_1}, \boxed{\text{L}_{\Psi'}}\ \overline{B_2}\ \boxed{\text{R}_{\Psi'}}\ , \overline{Y_1}\ \boxed{\text{R}_\Psi}, \overline{Y})\sigma\gamma; \sigma\gamma \rangle & \rightarrow_{FR}{}^{*} \\ \langle r; \theta' \rangle & \end{array}$$
   Ahora bien, el primer paso de $\mathcal{D}'$ puede ser simulado en la derivación $\mathcal{D}$ usando las cláusulas $\mathcal{C}$ y $\mathcal{C}_2$ de $\mathcal{P}$ como sigue:
   $$\begin{array}{ll} \langle \overline{X}, A, \overline{Y}; id \rangle & \rightarrow_{FR1}{}^{\mathcal{C}} \\ \langle (\overline{X}, \boxed{\text{L}_\Psi}\ \overline{X_1}, A_1, \overline{Y_1}\ \boxed{\text{R}_\Psi}, \overline{Y})\alpha; \alpha \rangle & \rightarrow_{FR1}{}^{\mathcal{C}_2} \\ \langle (\overline{X}, \boxed{\text{L}_\Psi}\ \overline{X_1}, \boxed{\text{L}_{\Psi'}}\ \overline{B_2}\ \boxed{\text{R}_{\Psi'}}, \overline{Y_1}\ \boxed{\text{R}_\Psi}\ , \overline{Y})\alpha\beta; \alpha\beta \rangle & \rightarrow_{FR}{}^{*} \\ \langle r; \theta \rangle & \end{array}$$

Por el Lema 5.1.1 podemos concluir que $\alpha\beta = \sigma\gamma[\mathcal{V}ar(\mathcal{Q}_0)]$, y entonces el primer estado de $\mathcal{D}$ coincide sintácticamente con el segundo de $\mathcal{D}'$.

Además, por la hipótesis de inducción $\theta = \theta'[\mathcal{V}ar(\mathcal{Q}_0)]$ y en consecuencia las f-derivaciones completas $\mathcal{D}$ y $\mathcal{D}'$ son equivalentes, como queríamos.

2. Desplegado basado en Regla 2 de la Definición 2.3.5.

$$\mathcal{C} \equiv (H_1 \leftarrow \overline{X_1}, A_1, \overline{Y_1}; \Psi) \in \mathcal{P} \quad \text{with } \Psi \equiv \langle p, \dot{\text{et}}_1, \dot{\text{et}}_2 \rangle$$

y

$$\mathcal{C}_2 \equiv (H_2 \leftarrow; q) \in \mathcal{P}$$

tal que, desplegando $\mathcal{C}$ con respecto a $\mathcal{C}_2$ usando la Definición 4.2.1, obtenemos:

$$\mathcal{C}' \equiv ((H_1 \leftarrow \overline{X_1}, q, \overline{Y_1})\sigma; \Psi) \in \mathcal{P}'$$

Por tanto, $\mathcal{D}'$ tiene la siguiente forma:

$$\begin{array}{ll} \langle \overline{X}, A, \overline{Y}; id \rangle & \rightarrow_{FR1}{}^{\mathcal{C}'} \\ \langle (\overline{X}, \boxed{\text{L}_\Psi}\ \overline{X_1}, q, \overline{Y_1}\ \boxed{\text{R}_\Psi}, \overline{Y})\sigma\gamma; \sigma\gamma \rangle & \rightarrow_{FR}{}^{*} \\ \langle r; \theta' \rangle & \end{array}$$

Entonces el primer paso de $\mathcal{D}'$ puede ser simulado en la derivación $\mathcal{D}$ usando las cláusulas $\mathcal{C}$ y $\mathcal{C}_2$ de $\mathcal{P}$ como sigue:

$$\begin{array}{ll} \langle \overline{X}, A, \overline{Y}; id \rangle & \rightarrow_{FR1}{}^{\mathcal{C}} \\ \langle (\overline{X}, \boxed{\text{L}_\Psi}\ \overline{X_1}, A_1, \overline{Y_1}\ \boxed{\text{R}_\Psi}, \overline{Y})\alpha; \alpha \rangle & \rightarrow_{FR2}{}^{\mathcal{C}_2} \\ \langle (\overline{X}, \boxed{\text{L}_\Psi}\ \overline{X_1}, q, \overline{Y_1}\ \boxed{\text{R}_\Psi}, \overline{Y})\alpha\beta; \alpha\beta \rangle & \rightarrow_{FR}{}^{*} \\ \langle r; \theta \rangle & \end{array}$$

Por el Lema 5.1.1 podemos concluir que $\alpha\beta = \sigma\gamma[\mathcal{V}ar(\mathcal{Q}_0)]$, y el tercer estado de $\mathcal{D}$ coincide sintácticamente con el segundo de $\mathcal{D}'$.

Además, por la hipótesis de inducción $\theta = \theta'[\mathcal{V}ar(\mathcal{Q}_0)]$ y por tanto son equivalentes las f-derivaciones completas $\mathcal{D}$ y $\mathcal{D}'$, como deseábamos probar.

3. Reemplazamiento de t-norma de tipo 1.
   Sea

$$\mathcal{C} \equiv (H_1 \leftarrow \overline{X_1}, \boxed{\mathrm{L}_{\Psi'}}\ n \boxed{\mathrm{R}_{\Psi'}}, \overline{Y_1}; \Psi) \in \mathcal{P} \quad \text{with } \Psi \equiv \langle p, \dot{\mathrm{et}}_1, \dot{\mathrm{et}}_2 \rangle,$$

   y sea $\Psi' \equiv \langle q, \dot{\mathrm{et}}_1', \dot{\mathrm{et}}_2' \rangle$.
   Por el reemplazamiento de t-norma en la cláusula $\mathcal{C}$ usando la regla de tipo 1 de la Definición 4.3.3, obtenemos la cláusula

$$\mathcal{C}' \equiv (H_1 \leftarrow \overline{X_1}, \dot{\mathrm{et}}_1'(q, n), \overline{Y_1}; \Psi) \in \mathcal{P}'$$

   Entonces, $\mathcal{D}'$ es la derivación:

$$\begin{array}{ll} \langle \overline{X}, A, \overline{Y}; id \rangle & \rightarrow_{FR1}{}^{\mathcal{C}'} \\ \langle (\overline{X}, \boxed{\mathrm{L}_{\Psi}}\ \overline{X_1}, \dot{\mathrm{et}}_1'(q, n), \overline{Y_1}\ \boxed{\mathrm{R}_{\Psi}}, \overline{Y})\alpha; \alpha \rangle & \rightarrow_{FR}{}^{*} \\ \langle r; \theta' \rangle & \end{array}$$

   Ahora bien, el primer paso de $\mathcal{D}'$ puede ser simulado en $\mathcal{P}$ dando dos pasos de computación en $\mathcal{D}$: el primero con la regla 1 usando la cláusula $\mathcal{C}$ y el segundo con la regla 3, como sigue:

$$\begin{array}{ll} \langle \overline{X}, A, \overline{Y}; id \rangle & \rightarrow_{FR1}{}^{\mathcal{C}} \\ \langle (\overline{X}, \boxed{\mathrm{L}_{\Psi}}\ \overline{X_1}, \boxed{\mathrm{L}_{\Psi'}}\ n\ \boxed{\mathrm{R}_{\Psi'}}, \overline{Y_1}\ \boxed{\mathrm{R}_{\Psi}}, \overline{Y})\alpha; \alpha \rangle & \rightarrow_{FR3} \\ \langle (\overline{X}, \boxed{\mathrm{L}_{\Psi}}\ \overline{X_1}, \dot{\mathrm{et}}_1'(q, n), \overline{Y_1}\ \boxed{\mathrm{R}_{\Psi}}, \overline{Y})\alpha; \alpha \rangle & \rightarrow_{FR}{}^{*} \\ \langle r; \theta \rangle & \end{array}$$

   Puesto que el tercer estado en $\mathcal{D}$ coincide sintácticamente con el segundo en $\mathcal{D}'$, se satisface el resultado por la hipótesis de inducción.

4. Reemplazamiento de t-norma de tipo 2.
   Sea

$$\mathcal{C} \equiv (H_1 \leftarrow \overline{X_1}, \boxed{\mathrm{L}_{\Psi'}}\ \overline{n}\ \boxed{\mathrm{R}_{\Psi'}}, \overline{Y_1}; \Psi) \in \mathcal{P} \quad \text{with } \Psi \equiv \langle p, \dot{\mathrm{et}}_1, \dot{\mathrm{et}}_2 \rangle$$

   y sea $\Psi' \equiv \langle q, \dot{\mathrm{et}}_1', \dot{\mathrm{et}}_2' \rangle$.
   Por reemplazamiento de t-norma de $\mathcal{C}$ usando la regla de tipo 2 de la Definición

4.3.3, obtenemos la nueva cláusula:

$$\mathcal{C}' \equiv (H_1 \leftarrow \overline{X_1}, \boxed{\mathrm{L}_{\Psi'}}\ \dot{\mathrm{et}}_2'(\overline{n})\boxed{\mathrm{R}_{\Psi'}}, \overline{Y_1}; \Psi) \in \mathcal{P}'$$

Entonces, $\mathcal{D}'$ tiene la siguiente forma:

$$\begin{array}{ll} \langle \overline{X}, A, \overline{Y}; id\rangle & \rightarrow_{FR1}{}^{\mathcal{C}'} \\ \langle (\overline{X}, \boxed{\mathrm{L}_{\Psi}}\ \overline{X_1}, \boxed{\mathrm{L}_{\Psi'}}\ \dot{\mathrm{et}}_2'(\overline{n})\boxed{\mathrm{R}_{\Psi'}}, \overline{Y_1}\ \boxed{\mathrm{R}_{\Psi}}, \overline{Y})\alpha; \alpha\rangle & \rightarrow_{FR}{}^{*} \\ \langle r; \theta'\rangle & \end{array}$$

Pero, el primer paso de $\mathcal{D}'$ puede ser simulado con cláusulas de $\mathcal{P}$ dando dos pasos de resolución en $\mathcal{D}$: el primero con la regla 1 usando la cláusula $\mathcal{C}$ y el segundo con la regla 4, del modo:

$$\begin{array}{ll} \langle \overline{X}, A, \overline{Y}; id\rangle & \rightarrow_{FR1}{}^{\mathcal{C}} \\ \langle (\overline{X}, \boxed{\mathrm{L}_{\Psi}}\ \overline{X_1}, \boxed{\mathrm{L}_{\Psi'}}\ \overline{n}\ \boxed{\mathrm{R}_{\Psi'}}, \overline{Y_1}\ \boxed{\mathrm{R}_{\Psi}}, \overline{Y})\alpha; \alpha\rangle & \rightarrow_{FR4} \\ \langle (\overline{X}, \boxed{\mathrm{L}_{\Psi}}\ \overline{X_1}, \boxed{\mathrm{L}_{\Psi'}}\ \dot{\mathrm{et}}_2'(\overline{n})\boxed{\mathrm{R}_{\Psi'}}, \overline{Y_1}\ \boxed{\mathrm{R}_{\Psi}}, \overline{Y})\alpha; \alpha\rangle & \rightarrow_{FR}{}^{*} \\ \langle r; \theta\rangle & \end{array}$$

Dado que el tercer estado en $\mathcal{D}$ coincide sintácticamente con el segundo en $\mathcal{D}'$, se satisface el resultado haciendo uso de la hipótesis de inducción.

5. Reemplazamiento de t-norma de tipo 3.

Sea

$$\mathcal{C} \equiv (H_1 \leftarrow n; \Psi) \in \mathcal{P} \quad \text{with } \Psi \equiv \langle p, \dot{\mathrm{et}}_1, \dot{\mathrm{et}}_2\rangle,$$

tal que, por reemplazamiento de t-norma de $\mathcal{C}$ usando la regla de tipo 3 en la Definición 4.3.3, tenemos la nueva cláusula

$$\mathcal{C}' \equiv (H_1 \leftarrow\ ; \dot{\mathrm{et}}_1(p, n)) \in \mathcal{P}'$$

Entonces, $\mathcal{D}'$ tiene la siguiente forma:

$$\begin{array}{ll} \langle \overline{X}, A, \overline{Y}; id\rangle & \rightarrow_{FR2}{}^{\mathcal{C}'} \\ \langle (\overline{X}, \dot{\mathrm{et}}_1(p, n), \overline{Y})\alpha; \alpha\rangle & \rightarrow_{FR}{}^{*} \\ \langle r; \theta'\rangle & \end{array}$$

Ahora bien, el primer paso de $\mathcal{D}'$ puede ser simulado con cláusulas de $\mathcal{P}$ dando dos pasos resolución en $\mathcal{D}$: el primero con la regla 1 usando la cláusula $\mathcal{C}$ y el segundo con la regla 3, como sigue:

$$\begin{array}{ll}
\langle \overline{X}, A, \overline{Y}; id\rangle & \rightarrow_{FR1}{}^{\mathcal{C}} \\
\langle (\overline{X}, \boxed{\mathrm{L}_\Psi}\ n\ \boxed{\mathrm{R}_\Psi}\ , \overline{Y})\alpha; \alpha\rangle & \rightarrow_{FR3} \\
\langle (\overline{X}, \dot{\mathrm{et}}_1(p,n), \overline{Y})\alpha; \alpha\rangle & \rightarrow_{FR}{}^{*} \\
\langle r; \theta\rangle &
\end{array}$$

Puesto que el tercer estado en $\mathcal{D}$ coincide sintácticamente con el segundo en $\mathcal{D}'$, se tiene el resultado en virtud de la hipótesis de inducción.

6. Reemplazamiento de t-norma de tipo 4.
   Sea
   $$\mathcal{C} \equiv (H_1 \leftarrow \overline{n}; \Psi) \in \mathcal{P} \quad \text{with } \Psi \equiv \langle p, \dot{\mathrm{et}}_1, \dot{\mathrm{et}}_2\rangle,$$
   tal que, por reemplazamiento de t-norma de $\mathcal{C}$ usando la regla de tipo 4 de la Definición 4.3.3, obtenemos la nueva cláusula
   $$\mathcal{C}' \equiv (H_1 \leftarrow\ \dot{\mathrm{et}}_2(\overline{n}); \Psi) \in \mathcal{P}'$$
   Así, $\mathcal{D}'$ tiene la forma:
   $$\begin{array}{ll}
   \langle \overline{X}, A, \overline{Y}; id\rangle & \rightarrow_{FR1}{}^{\mathcal{C}'} \\
   \langle (\overline{X}, \boxed{\mathrm{L}_\Psi}\ \dot{\mathrm{et}}_2(\overline{n})\boxed{\mathrm{R}_\Psi}\ , \overline{Y})\alpha; \alpha\rangle & \rightarrow_{FR}{}^{*} \\
   \langle r; \theta'\rangle &
   \end{array}$$
   Entonces, el primer paso de $\mathcal{D}'$ puede ser simulado con cláusulas de $\mathcal{P}$ usando dos pasos de resolución en $\mathcal{D}$: el primero con la regla 1 usando la cláusula $\mathcal{C}$ y la segunda con la regla 3, del modo:
   $$\begin{array}{ll}
   \langle \overline{X}, A, \overline{Y}; id\rangle & \rightarrow_{FR1}{}^{\mathcal{C}} \\
   \langle (\overline{X}, \boxed{\mathrm{L}_\Psi}\ \overline{n}\ \boxed{\mathrm{R}_\Psi}\ , \overline{Y})\alpha; \alpha\rangle & \rightarrow_{FR4} \\
   \langle (\overline{X}, \boxed{\mathrm{L}_\Psi}\ \dot{\mathrm{et}}_2(\overline{n})\ \boxed{\mathrm{R}_\Psi}\ , \overline{Y})\alpha; \alpha\rangle & \rightarrow_{FR}{}^{*} \\
   \langle r; \theta\rangle &
   \end{array}$$

Puesto que el tercer estado en $\mathcal{D}$ coincide sintácticamente con el segundo en $\mathcal{D}'$, se obtiene el resultado por la hipótesis de inducción.

□

5.2. Completitud fuerte de transformaciones con lef-Prolog

Procedemos ahora con el complementario del teorema anterior, esto es, nos planteamos la completitud fuerte, por la que obtendremos que todas las respuestas computadas del programa original se encuentran en el programa transformado.

Teorema 5.2.1 (Completitud fuerte)**.** *Sea $\mathcal{P}$ un programa* lef-Prolog *y sea $\mathcal{G} \equiv \leftarrow \mathcal{Q}_0$ with $le_2 = \dot{\text{et}}_2^0$ un objetivo* lef-Prolog*. Si $\mathcal{P}'$ es un programa* lef-Prolog *obtenido por desplegado difuso o reemplazamiento de t-norma de $\mathcal{P}$, entonces,*

$$\langle \boxed{\mathrm{L}_{\Psi_0}} \mathcal{Q}_0 \boxed{\mathrm{R}_{\Psi_0}}; id \rangle \rightarrow_{FR}{}^{*} \langle r; \theta' \rangle \ \textit{en}\ \mathcal{P}'$$

si

$$\langle \boxed{\mathrm{L}_{\Psi_0}} \mathcal{Q}_0 \boxed{\mathrm{R}_{\Psi_0}}; id \rangle \rightarrow_{FR}{}^{*} \langle r; \theta \rangle \ \textit{en}\ \mathcal{P},$$

donde $\theta' = \theta[\mathcal{V}ar(\mathcal{Q}_0)]$ y $\Psi_0 \equiv \langle 1, void, \dot{et}_2^0 \rangle$.

Demostración. La demostración consiste en simular en $\mathcal{P}'$ una f-derivación (reordenada) originariamente ejecutada en $\mathcal{P}$. Sea, entonces,

$$\mathcal{D}_0 : [\langle \boxed{\mathrm{L}_{\Psi_0}} \mathcal{Q}_0 \boxed{\mathrm{R}_{\Psi_0}}; id \rangle \rightarrow_{FR}{}^{n} \langle r; \theta_0 \rangle]$$

una f-derivación (genérica) con n-pasos para $\mathcal{G}$ en $\mathcal{P}$.

Supongamos ahora que $\mathcal{C} \in \mathcal{P}$ es la cláusula desplegada en $\mathcal{P}$ que, obviamente, no pertenece a $\mathcal{P}'$.

Cualquier paso dado con la cláusula $\mathcal{C}$ en $\mathcal{D}_0$ introduce una instancia del cuerpo de $\mathcal{C}$ en el siguiente estado de la derivación. En estas derivaciones, el cuerpo *instanciado* de $\mathcal{C}$ debe necesariamente ser reducido en el paso inmediatamente posterior o en uno subsiguiente. En el segundo caso, podemos, con seguridad, intercambiar el paso dado con la cláusula $\mathcal{C}$ y el siguiente, aplicando el Lema 2.4.5 de Conmutación para el lenguaje lef-Prolog.

Además, aplicando repetidamente este lema, podemos obtener una nueva f-derivación de n-pasos

$$\mathcal{D} : [\langle \boxed{\mathrm{L}_{\Psi_0}} \mathcal{Q}_0 \boxed{\mathrm{R}_{\Psi_0}}; id\rangle \rightarrow_{FR}{}^{n} \langle r; \theta\rangle]$$

en $\mathcal{P}$ verificando $\theta = \theta_0[\mathcal{V}ar(\mathcal{Q}_0)]$ módulo renombramiento de variables, donde todo paso (si existe) que use la cláusula $\mathcal{C}$ desplegada $\mathcal{P}$, es seguido de otro paso que explota una lef-expresión introducida justamente por el paso previo (esto es, perteneciente al cuerpo instanciado de $\mathcal{C}$).

Decimos que $\mathcal{D}$ es una f-derivación reordenada de éxito con respecto a la cláusula $\mathcal{C}$ elegida.

Ahora, y de manera análoga a cómo procedimos en el teorema anterior, simularemos $\mathcal{D}$ en $\mathcal{P}'$ construyendo una nueva derivación $\mathcal{D}'$ usando cláusulas de $\mathcal{P}'$; seguiremos un esquema semejante al empleado en el Teorema 5.1.2, pero invirtiendo ahora el uso de los términos $\mathcal{P}$ y $\mathcal{P}'$. La construcción de $\mathcal{D}'$ se realizará por inducción en n, la longitud de $\mathcal{D}$.

Puesto que para el caso base, esto es $n = 0$, el resultado es trivial, procedemos con el caso general en que $n > 0$.

Sea entonces,

$$\mathcal{D} : [\langle \boxed{\mathrm{L}_{\Psi_0}} \mathcal{Q}_0 \boxed{\mathrm{R}_{\Psi_0}}; id\rangle \rightarrow_{FR} \langle \boxed{\mathrm{L}_{\Psi_0}} \mathcal{Q}' \boxed{\mathrm{R}_{\Psi_0}} \vartheta\rangle \rightarrow_{FR}{}^{*} \langle r; \theta\rangle]$$

Si el primer paso de $\mathcal{D}$ ha sido dado con la segunda, tercera o cuarta regla de la Definición 2.3.5 o incluso si ha sido ejecutado con la primera regla, pero usando una cláusula perteneciente a $\mathcal{P}'$, el resultado se verifica por la hipótesis de induccción.

En otro caso, este paso inicial ha sido dado con la regla 1 usando una cláusula $\mathcal{C}$ que, una vez transformada, genera la nueva cláusula $\mathcal{C}' \in \mathcal{P}'$.

Puesto que el paso de desplegado ha sido ejecutado con una de las dos reglas de la Definición 2.3.5, y el paso de reemplazamiento de t-norma satisface una de las cuatro reglas de la Definición 4.3.3 tratamos cada uno de los casos por separado.

1. Desplegado basado en Regla 1 de la Definición 2.3.5.
   Sea
   $$\mathcal{C} \equiv (H_1 \leftarrow \overline{X_1}, A_1, \overline{Y_1}; \Psi) \in \mathcal{P} \quad \text{with } \Psi \equiv \langle p, \dot{\mathrm{e}}\mathrm{t}_1, \dot{\mathrm{e}}\mathrm{t}_2\rangle$$
   y
   $$\mathcal{C}_2 \equiv (H_2 \leftarrow \overline{B_2}; \Psi') \in \mathcal{P} \quad \text{with } \Psi' \equiv \langle q, \dot{\mathrm{e}}\mathrm{t}_1', \dot{\mathrm{e}}\mathrm{t}_2'\rangle$$

Si desplegamos $\mathcal{C}$ con respecto a $\mathcal{C}_2$, haciendo uso de la Definición 4.2.1, obtenemos:

$$\mathcal{C}' \equiv ((H_1 \leftarrow \overline{X_1}, \boxed{\mathrm{L}_{\Psi'}}\ \overline{B_2}\ \boxed{\mathrm{R}_{\Psi'}}, \overline{Y_1})\sigma; \Psi) \in \mathcal{P}'$$

Suponemos que $\mathcal{Q}_0 \equiv \overline{X}, A, \overline{Y}$ donde $\overline{X}$ y $\overline{Y}$ son secuencias arbitrarias de lf-expresiones válidas y A es el átomo seleccionado.

Entonces, como $\mathcal{D}$ es una f-derivación reordenada de éxito con respecto a la cláusula $\mathcal{C}$, tiene la siguiente forma:

$$\begin{array}{ll}
\langle \overline{X}, A, \overline{Y}; id\rangle & \rightarrow_{FR1}{}^{\mathcal{C}} \\
\langle (\overline{X}, \boxed{\mathrm{L}_{\Psi}}\ \overline{X_1}, A_1, \overline{Y_1}\ \boxed{\mathrm{R}_{\Psi}}, \overline{Y})\alpha; \alpha\rangle & \rightarrow_{FR1}{}^{\mathcal{C}_2} \\
\langle (\overline{X}, \boxed{\mathrm{L}_{\Psi}}\ \overline{X_1}, \boxed{\mathrm{L}_{\Psi'}}\ \overline{B_2}\ \boxed{\mathrm{R}_{\Psi'}}, \overline{Y_1}\ \boxed{\mathrm{R}_{\Psi}}, \overline{Y})\alpha\beta; \alpha\beta\rangle & \rightarrow_{FR}{}^{*} \\
\langle r; \theta\rangle &
\end{array}$$

Ahora bien, los dos primeros pasos de $\mathcal{D}$ pueden ser simulados en $\mathcal{P}'$ usando la cláusula $\mathcal{C}'$ en la derivación $\mathcal{D}'$ como sigue:

$$\begin{array}{ll}
\langle \overline{X}, A, \overline{Y}; id\rangle & \rightarrow_{FR1}{}^{\mathcal{C}'} \\
\langle (\overline{X}, \boxed{\mathrm{L}_{\Psi}}\ \overline{X_1}, \boxed{\mathrm{L}_{\Psi'}}\ \overline{B_2}\ \boxed{\mathrm{R}_{\Psi'}}, \overline{Y_1}\ \boxed{\mathrm{R}_{\Psi'}}, \overline{Y})\sigma\gamma; \sigma\gamma\rangle & \rightarrow_{FR}{}^{*} \\
\langle r; \theta'\rangle &
\end{array}$$

Por el Lema 5.1.1 podemos concluir que $\alpha\beta = \sigma\gamma[\mathcal{V}ar(\mathcal{Q}_0)]$, por lo que el tercer estado de $\mathcal{D}$ coincide sintácticamente con el segundo en $\mathcal{D}'$.

Además, por la hipótesis de inducción $\theta = \theta'[\mathcal{V}ar(\mathcal{Q}_0)]$ y, en consecuencia, son equivalentes las f-derivaciones completas $\mathcal{D}$ y $\mathcal{D}'$, como queríamos probar.

2. Desplegado basado en Regla 2 de la Definición 2.3.5.
Sea

$$\mathcal{C} \equiv (H_1 \leftarrow \overline{X_1}, A_1, \overline{Y_1}; \Psi) \in \mathcal{P} \quad \text{with } \Psi \equiv \langle p, \dot{\mathrm{e}}\mathrm{t}_1, \dot{\mathrm{e}}\mathrm{t}_2\rangle$$

y

$$\mathcal{C}_2 \equiv (H_2 \leftarrow; q) \in \mathcal{P}$$

tal que, por desplegado de $\mathcal{C}$ con respecto a $\mathcal{C}_2$ usando la Definición 4.2.1, tenemos:

$$\mathcal{C}' \equiv ((H_1 \leftarrow \overline{X_1}, q, \overline{Y_1})\sigma; \Psi) \in \mathcal{P}'$$

Luego, como $\mathcal{D}$ es una f-derivación reordenada de éxito con respecto a la cláusula $\mathcal{C}$, tiene la forma:

$$\begin{array}{ll} \langle \overline{X}, A, \overline{Y}; id\rangle & \rightarrow_{FR1}{}^{\mathcal{C}} \\ \langle (\overline{X}, \boxed{\mathrm{L}_\Psi}\ \overline{X_1}, A_1, \overline{Y_1}\ \boxed{\mathrm{R}_\Psi}, \overline{Y})\alpha; \alpha\rangle & \rightarrow_{FR2}{}^{\mathcal{C}_2} \\ \langle (\overline{X}, \boxed{\mathrm{L}_\Psi}\ \overline{X_1}, q, \overline{Y_1}\ \boxed{\mathrm{R}_\Psi}, \overline{Y})\alpha\beta; \alpha\beta\rangle & \rightarrow_{FR}{}^{*} \\ \langle r; \theta\rangle & \end{array}$$

Ahora bien, los dos primeros pasos en $\mathcal{D}$ pueden ser simulados en $\mathcal{P}'$ usando la cláusula $\mathcal{C}'$ como sigue:

$$\begin{array}{ll} \langle \overline{X}, A, \overline{Y}; id\rangle & \rightarrow_{FR1}{}^{\mathcal{C}'} \\ \langle (\overline{X}, \boxed{\mathrm{L}_\Psi}\ \overline{X_1}, q, \overline{Y_1}\ \boxed{\mathrm{R}_\Psi}, \overline{Y})\sigma\gamma; \sigma\gamma\rangle & \rightarrow_{FR}{}^{*} \\ \langle r; \theta'\rangle & \end{array}$$

Por el Lema 5.1.1 podemos concluir que $\alpha\beta = \sigma\gamma[\mathcal{V}ar(\mathcal{Q}_0)]$, y en consecuencia el tercer estado en $\mathcal{D}$ coincide sintácticamente con el segundo en $\mathcal{D}'$.

Además, por la hipótesis de inducción $\theta = \theta'[\mathcal{V}ar(\mathcal{Q}_0)]$ y por tanto las f-derivaciones enteras $\mathcal{D}$ y $\mathcal{D}'$ son equivalentes, como deseábamos.

3. Reemplazamiento de t-norma de tipo 1 según la Definición 4.3.3.
Sea

$$\mathcal{C} \equiv (H_1 \leftarrow \overline{X_1}, \boxed{\mathrm{L}_{\Psi'}}\ n\ \boxed{\mathrm{R}_{\Psi'}}, \overline{Y_1}; \Psi) \in \mathcal{P} \quad \text{with } \Psi \equiv \langle p, \dot{\mathrm{e}}\mathrm{t}_1, \dot{\mathrm{e}}\mathrm{t}_2\rangle$$

y sea $\Psi' \equiv \langle q, \dot{\mathrm{e}}\mathrm{t}_1', \dot{\mathrm{e}}\mathrm{t}_2'\rangle$. Por reemplazamiento de t-norma de tipo 1 de $\mathcal{C}$ usando la Definición 4.3.3, obtenemos la nueva cláusula

$$\mathcal{C}' \equiv (H_1 \leftarrow \overline{X_1}, \dot{\mathrm{e}}\mathrm{t}_1(q, n), \overline{Y_1}; \Psi) \in \mathcal{P}'$$

Además, ya que $\mathcal{D}$ es una f-derivación reordenada de éxito con respecto a la cláusula $\mathcal{C}$, tiene la siguiente forma:

$$\begin{array}{ll}
\langle \overline{X}, A, \overline{Y}; id\rangle & \rightarrow_{FR1}{}^{\mathcal{C}} \\
\langle (\overline{X}, \boxed{\mathrm{L}_{\Psi}}\ \overline{X_1}, \boxed{\mathrm{L}_{\Psi'}}\ n \boxed{\mathrm{R}_{\Psi'}}, \overline{Y_1}\ \boxed{\mathrm{R}_{\Psi}}, \overline{Y})\alpha; \alpha\rangle & \rightarrow_{FR3} \\
\langle (\overline{X}, \boxed{\mathrm{L}_{\Psi}}\ \overline{X_1}, \dot{\mathrm{e}}\dot{\mathrm{t}}_1'(q,n), \overline{Y_1}\ \boxed{\mathrm{R}_{\Psi}}, \overline{Y})\alpha; \alpha\rangle & \rightarrow_{FR}{}^{*} \\
\langle r; \theta\rangle &
\end{array}$$

Pero, los dos primeros pasos en $\mathcal{D}$ pueden ser simulados en $\mathcal{P}'$ dando un único paso de resolución usando la cláusula $\mathcal{C}'$ en $\mathcal{D}'$ como sigue:

$$\begin{array}{ll}
\langle \overline{X}, A, \overline{Y}; id\rangle & \rightarrow_{FR1}{}^{\mathcal{C}'} \\
\langle (\overline{X}, \boxed{\mathrm{L}_{\Psi}}\ \overline{X_1}, \dot{\mathrm{e}}\dot{\mathrm{t}}_1'(q,n), \overline{Y_1}\ \boxed{\mathrm{R}_{\Psi}}, \overline{Y})\alpha; \alpha\rangle & \rightarrow_{FR}{}^{*} \\
\langle r; \theta'\rangle &
\end{array}$$

Dado que el tercer estado en $\mathcal{D}$ coincide sintácticamente con el segundo en $\mathcal{D}'$, se satisface el resultado por la hipótesis de inducción.

4. Reemplazamiento de t-norma de tipo 2 según la Definición 4.3.3.
Sea

$$\mathcal{C} \equiv (H_1 \leftarrow \overline{X_1}, \boxed{\mathrm{L}_{\Psi'}}\ \overline{n}\ \boxed{\mathrm{R}_{\Psi'}}, \overline{Y_1}; \Psi) \in \mathcal{P} \quad \text{with } \Psi \equiv \langle p, \dot{\mathrm{e}}\dot{\mathrm{t}}_1, \dot{\mathrm{e}}\dot{\mathrm{t}}_2\rangle$$

y $\Psi' \equiv \langle q, \dot{\mathrm{e}}\dot{\mathrm{t}}_1', \dot{\mathrm{e}}\dot{\mathrm{t}}_2'\rangle$. Por reemplazamiento de t-norma de tipo 2 de $\mathcal{C}$ usando la Definición 4.3.3, obtenemos la nueva cláusula

$$\mathcal{C}' \equiv (H_1 \leftarrow \overline{X_1}, \boxed{\mathrm{L}_{\Psi'}}\ \dot{\mathrm{e}}\dot{\mathrm{t}}_2(\overline{n})\boxed{\mathrm{R}_{\Psi'}}, \overline{Y_1}; \Psi) \in \mathcal{P}'$$

Además, ya que $\mathcal{D}$ es una f-derivación reordenada de éxito con respecto a la cláusula $\mathcal{C}$, tiene la forma:

$$\begin{array}{ll}
\langle \overline{X}, A, \overline{Y}; id\rangle & \rightarrow_{FR1}{}^{\mathcal{C}} \\
\langle (\overline{X}, \boxed{\mathrm{L}_{\Psi}}\ \overline{X_1}, \boxed{\mathrm{L}_{\Psi'}}\ \overline{n}\ \boxed{\mathrm{R}_{\Psi'}}, \overline{Y_1}\ \boxed{\mathrm{R}_{\Psi}}, \overline{Y})\alpha; \alpha\rangle & \rightarrow_{FR4} \\
\langle (\overline{X}, \boxed{\mathrm{L}_{\Psi}}\ \overline{X_1}, \boxed{\mathrm{L}_{\Psi'}}\ \dot{\mathrm{e}}\dot{\mathrm{t}}_2'(\overline{n})\boxed{\mathrm{R}_{\Psi'}}, \overline{Y_1}\ \boxed{\mathrm{R}_{\Psi}}, \overline{Y})\alpha; \alpha\rangle & \rightarrow_{FR}{}^{*} \\
\langle r; \theta\rangle &
\end{array}$$

Ahora bien, los primeros dos pasos en $\mathcal{D}$ pueden ser simulados en $\mathcal{P}'$ dando un único paso de resolución usando la cláusula $\mathcal{C}'$ en $\mathcal{D}'$ del modo:

$$\begin{array}{ll}\langle \overline{X}, A, \overline{Y}; id\rangle & \rightarrow_{FR1}{}^{\mathcal{C}'} \\ \langle (\overline{X}, \boxed{\mathrm{L}_\Psi}\ \overline{X_1}, \boxed{\mathrm{L}_{\Psi'}}\ \dot{\mathrm{et}}_2'(\overline{n})\boxed{\mathrm{R}_{\Psi'}}\ , \overline{Y_1}\ \boxed{\mathrm{R}_\Psi}\ , \overline{Y})\alpha; \alpha\rangle & \rightarrow_{FR}{}^{*} \\ \langle r; \theta'\rangle & \end{array}$$

Como el tercer estado en $\mathcal{D}$ coincide sintácticamente con el segundo en $\mathcal{D}'$, se obtiene el resultado por la hipótesis de inducción.

5. Reemplazamiento de t-norma de tipo 3 según la Definición 4.3.3.
Sea

$$\mathcal{C} \equiv (H_1 \leftarrow n; \Psi) \in \mathcal{P} \quad \text{with } \Psi \equiv \langle p, \dot{\mathrm{et}}_1, \dot{\mathrm{et}}_2\rangle$$

tal que, por reemplazamiento de t-norma de $\mathcal{C}$ usando la regla de tipo 3 de la Definición 4.3.3, tenemos la nueva cláusula

$$\mathcal{C}' \equiv (H_1 \leftarrow\ ; \dot{\mathrm{et}}_1(p, n)) \in \mathcal{P}'$$

Además, dado que $\mathcal{D}$ es una f-derivación reordenada de éxito con respecto a la cláusula $\mathcal{C}$, tiene la siguiente forma:

$$\begin{array}{ll}\langle \overline{X}, A, \overline{Y}; id\rangle & \rightarrow_{FR1}{}^{\mathcal{C}} \\ \langle (\overline{X}, \boxed{\mathrm{L}_\Psi}\ n\ \boxed{\mathrm{R}_\Psi}\ , \overline{Y})\alpha; \alpha\rangle & \rightarrow_{FR3} \\ \langle (\overline{X}, \dot{\mathrm{et}}_1(p, n), \overline{Y})\alpha; \alpha\rangle & \rightarrow_{FR}{}^{*} \\ \langle r; \theta\rangle & \end{array}$$

Entonces, los dos primeros pasos en $\mathcal{D}$ (el primero con la regla 1 usando la cláusula $\mathcal{C}$ y el segundo con la regla 3) pueden ser simulados en $\mathcal{P}'$ dando un único paso de resolución usando la cláusula $\mathcal{C}'$ en $\mathcal{D}'$ como sigue:

$$\begin{array}{ll}\langle \overline{X}, A, \overline{Y}; id\rangle & \rightarrow_{FR2}{}^{\mathcal{C}'} \\ \langle (\overline{X}, \dot{\mathrm{et}}_1(p, n), \overline{Y})\alpha; \alpha\rangle & \rightarrow_{FR}{}^{*} \\ \langle r; \theta'\rangle & \end{array}$$

Ya que el tercer estado en $\mathcal{D}$ coincide sintácticamente con el segundo en $\mathcal{D}'$, se tiene el resultado por la hipótesis de inducción.

6. Reemplazamiento de t-norma de tipo 4 según la Definición 4.3.3.
Sea

$$\mathcal{C} \equiv (H_1 \leftarrow \overline{n}; \Psi) \in \mathcal{P} \quad \text{with } \Psi \equiv \langle p, \dot{\text{et}}_1, \dot{\text{et}}_2 \rangle$$

tal que, por reemplazamiento de t-norma de $\mathcal{C}$ usando la regla de tipo 4 de la Definición 4.3.3, obtenemos la cláusula

$$\mathcal{C}' \equiv (H_1 \leftarrow \ \dot{\text{et}}_2(\overline{n}); \Psi) \in \mathcal{P}'$$

Además, como $\mathcal{D}$ es una f-derivación reordenada de éxito con respecto a la cláusula $\mathcal{C}$, es del modo:

$$\begin{array}{ll}
\langle \overline{X}, A, \overline{Y}; id \rangle & \rightarrow_{FR1}{}^{\mathcal{C}} \\
\langle (\overline{X}, \boxed{\text{L}_\Psi}\ \overline{n}\ \boxed{\text{R}_\Psi}\ , \overline{Y}) \alpha; \alpha \rangle & \rightarrow_{FR4} \\
\langle (\overline{X}, \boxed{\text{L}_\Psi}\ \dot{\text{et}}_2(\overline{n})\ \boxed{\text{R}_\Psi}\ , \overline{Y}) \alpha; \alpha \rangle & \rightarrow_{FR}{}^{*} \\
\langle r; \theta \rangle &
\end{array}$$

Ahora bien, los dos primeros pasos en $\mathcal{D}$ (el primero con la regla 1 usando la cláusula $\mathcal{C}$ y la segunda con la regla 4) pueden ser simulados en $\mathcal{P}'$ dando un único paso de resolución usando la cláusula $\mathcal{C}'$ en $\mathcal{D}'$ como sigue:

$$\begin{array}{ll}
\langle \overline{X}, A, \overline{Y}; id \rangle & \rightarrow_{FR1}{}^{\mathcal{C}'} \\
\langle (\overline{X}, \boxed{\text{L}_\Psi}\ \dot{\text{et}}_2(\overline{n})\ \boxed{\text{R}_\Psi}\ , \overline{Y}) \alpha; \alpha \rangle & \rightarrow_{FR}{}^{*} \\
\langle r; \theta' \rangle &
\end{array}$$

Puesto que el tercer estado en $\mathcal{D}$ coincide sintácticamente con el segundo en $\mathcal{D}'$, se obtiene el resultado por la hipótesis de inducción.

□

5.3. Corrección total fuerte de transformaciones con lef-Prolog

El siguiente resultado, que se obtiene a partir del Teorema 5.1.2 y el Teorema 5.2.1, formaliza para el lenguaje lef-Prolog la mejor propiedad que puede esperarse de esta transformación de desplegado difuso, esto es, la exacta y total correspondencia entre las respuestas computadas difusas para los objetivos ejecutados contra los programas original y transformado.

Teorema 5.3.1 (Corrección Total Fuerte del Desplegado Difuso)**.** *Sea $\mathcal{P}$ un programa* lef-Prolog *y $\mathcal{G} \equiv \leftarrow \mathcal{Q}$ un objetivo* lef-Prolog*. Si $\mathcal{P}'$ es un programa* lef-Prolog *obtenido por desplegado difuso de $\mathcal{P}$, entonces,*

$$\langle \mathcal{Q}; id \rangle \rightarrow_{FR}{}^{*} \langle r; \theta \rangle \text{ en } \mathcal{P}$$

si, y sólo si,

$$\langle \mathcal{Q}; id \rangle \rightarrow_{FR}{}^{*} \langle r; \theta' \rangle \text{ en } \mathcal{P}'$$

donde $\theta = \theta'[\mathcal{V}ar(\mathcal{Q})]$.

Para finalizar, obtenemos también para el lenguaje lef-Prolog las mejores propiedades que pueden esperarse del sistema de transformación considerado, a saber:

- en el aspecto teórico, la exacta y total correspondencia entre las respuesta computadas difusas para objetivos ejecutados en los programas original y transformado, y
- en el aspecto práctico, la ganancia en eficiencia cuando ejecutamos programas transformados reduciendo el número de pasos de resolución-SLD difusa necesarios para resolver un objetivo.

Teorema 5.3.2 (Corrección Total Fuerte del Sistema de Transformación)**.** *Considérese $(\mathcal{P}_0, \ldots, \mathcal{P}_k)$ una secuencia de transformación de programas* lef-Prolog *tal que cada programa de la secuencia, excepto el programa inicial* ef-Prolog *$\mathcal{P}_0$, es obtenido del inmediato precedente aplicando desplegado difuso o reemplazamiento de t-norma. Entonces, para un objetivo* lf-Prolog *$\mathcal{G} \equiv \leftarrow \mathcal{Q}$ with $le_2 = \dot{\text{et}}_2$, tenemos que:*

$$\langle \boxed{\text{L}_\Psi}\ \mathcal{Q}\ \boxed{\text{R}_\Psi}\ ; id \rangle \rightarrow_{FR}{}^{n} \langle r; \theta \rangle \text{ en } \mathcal{P}_0$$

si, y sólo si,

$$\langle \boxed{\mathrm{L}_\Psi}\ \mathcal{Q}\ \boxed{\mathrm{R}_\Psi}\ ; id\rangle \rightarrow_{FR}{}^{m} \langle r;\theta\rangle \text{ en } \mathcal{P}_k$$

donde $\Psi \equiv \langle 1, void, \dot{\mathrm{et}}_2\rangle$ *y el número de pasos de resolución-SLD difusa en cada derivación verifica que* $m \leq n$.

Demostración. Para demostrar este resultado consideraremos separadamente las dos implicaciones de la equivalencia, puesto que la corrección total fuerte del desplegado difuso o reemplazamiento de t-norma supone la corrección parcial fuerte ($\Leftarrow$) y la completitud fuerte ($\Rightarrow$).

Finalmente, estas dos implicaciones pueden probarse fácilmente como sigue:

- La corrección total fuerte de un sistema de transformación es, ahora, inmediata sin más que hacer uso de los Teoremas 5.1.2 y 5.2.1, puesto que ambos establecen la equivalencia entre cualquier par de programas consecutivos de la secuencia de transformación $(\mathcal{P}_0, \ldots, \mathcal{P}_k)$, que supone la equivalencia final entre $\mathcal{P}_0$ y $\mathcal{P}_k$.
- Observando la reducción de la longitud de la derivación de éxito en los programas transformados, hemos visto en la demostración de los referidos Teoremas 5.1.2 y 5.2.1 que cualquier paso de resolución-SLD difuso dado con una nueva cláusula obtenida por desplegado difuso o reemplazamiento de t-norma subsume dos pasos de resolución-SLD difusa dados contra el programa original, que confirma que $m \leq n$, como queríamos probar.

□

5.4. Propiedades de corrección del desplegado operacional del lenguaje multi-adjunto

La siguientes secciones de este capítulo están dedicadas a establecer, de manera similar a cómo planteamos para el lenguaje lef-Prolog, las mejores propiedades que pueden esperarse de nuestras transformaciones de desplegado del lenguaje multi-adjunto, es decir:

- en el aspecto teórico, que coinciden las mismas respuestas computadas difusas de cualesquiera objetivos ejecutados contra los programas original y transformado, y

- en el aspecto práctico, la ganancia en eficiencia de los programas desplegados, toda vez que en estos se reduce el número de los pasos admisibles difusos necesarios para resolver un objetivo.

Comenzamos demostrando el siguiente Lema, que puede considerarse análogo al Lema 3.1.9 (de conservacción de sustituciones en las respuestas computadas difusas). Intuitivamente, este resultado demuestra que, incluso en el caso en que dos pasos admisibles no puedan ser intercambiados, puesto que el segundo explota un átomo que ha sido introducido en el objetivo considerado por el primero, su efecto (con respecto a las sustituciones de las respuestas computadas difusas) puede ser simulado ejecutando un sólo paso que use una regla transformada obtenida por desplegado difuso.

Lema 5.4.1. *Sea $\mathcal{P}$ un programa multi-adjunto, $\mathcal{Q}_0$ un objetivo y $\mathcal{R}_1, \mathcal{R}_2 \ll \mathcal{P}$. Entonces,*

$$\langle \mathcal{Q}_0; \theta_0 \rangle \rightarrow_{AS1}{}^{\mathcal{R}_1} \langle \mathcal{Q}_1; \theta_0\theta_1 \rangle \rightarrow_{AS}{}^{\mathcal{R}_2} \langle \mathcal{Q}_2; \theta_0\theta_1\theta_2 \rangle$$

si, y sólo si,

$$\langle \mathcal{Q}_0; \theta_0 \rangle \rightarrow_{AS1}{}^{\mathcal{R}_3} \langle \mathcal{Q}_3; \theta_0\theta_3 \rangle$$

y, además, $\theta_0\theta_1\theta_2 = \theta_0\theta_3$ $[\mathcal{V}ar(\mathcal{Q}_0)]$, donde el segundo paso de la primera derivación es del tipo $AS1$ o $AS2$ y explota un átomo introducido en $\mathcal{Q}_1$ después del primer paso, y siendo $\mathcal{R}_3$ la regla obtenida por desplegado difuso de $\mathcal{R}_1$ usando $\mathcal{R}_2$.

Demostración.
($\Rightarrow$) Sea $\mathcal{R}_1 : \langle H_1 \leftarrow_1 \mathcal{B}_1[A_1]; v_1 \rangle$, sea H_2 (el átomo de) la cabeza de la regla $\mathcal{R}_2$ y supongamos que A es el átomo seleccionado en $\mathcal{Q}_0$ por la regla de computación fijada. Entonces, en la derivación $\langle \mathcal{Q}_0;\theta_0 \rangle \rightarrow_{AS1}{}^{\mathcal{R}_1} \langle \mathcal{Q}_1;\theta_0\theta_1 \rangle \rightarrow_{AS}{}^{\mathcal{R}_2} \langle \mathcal{Q}_2;\theta_0\theta_1\theta_2 \rangle$ tenemos que: $\theta_1 = mgu(\{A = H_1\})$ y $\mathcal{Q}_1 = \mathcal{Q}_0[A/(v_1 \&_1 \mathcal{B}_1[A_1])]\theta_1$. Además, si $A_1\theta_1$ es el átomo seleccionado en $\mathcal{Q}_1$ por la regla de computación, obtenemos que $\theta_2 = mgu(\{A_1\theta_1 = H_2\})$. Denotemos ahora $\sigma = mgu(\{A_1 = H_2\})$.

Entonces, se satisfacen las siguientes igualdades:

$$
\begin{array}{lcl}
\theta_1\theta_2 & = & \\
\theta_1 mgu(\{A_1\theta_1 = H_2\}) & = & \text{(ya que } Dom(\theta_1) \cap \mathcal{V}ar(\mathcal{R}_2) = \varnothing) \\
\theta_1 mgu(\widehat{mgu}(\{A_1 = H_2\})\theta_1) & = & \\
\theta_1 mgu(\widehat{\sigma}\theta_1) & = & \text{(por la Proposición 2.4.1)} \\
\theta_1 \Uparrow \sigma & = & \text{(por la Proposición 2.4.1)} \\
\sigma mgu(\widehat{\theta_1}\sigma) & = & \\
\sigma mgu(\widehat{mgu}(\{A = H_1\})\sigma) & = & \text{(ya que } Dom(\sigma) \cap \mathcal{V}ar(\mathcal{Q}_0) = \varnothing) \\
\sigma mgu(\{A = H_1\sigma\}) & &
\end{array}
$$

Además, como $\theta_1\theta_2 \neq fallo$, se tiene $\sigma \neq fallo$ y entonces existe una regla $\mathcal{R}_3$ obtenida por desplegado de (el átomo A_1 del cuerpo de) $\mathcal{R}_1$ usando $\mathcal{R}_2$, tal que la cabeza de $\mathcal{R}_3$ es el átomo $H_1\sigma$. Además, ya que $mgu(\{A = H_1\sigma\}) \neq fallo$, puede ejecutarse el siguiente paso admisible sobre el átomo seleccionado A en $\mathcal{Q}_0$: $\langle \mathcal{Q}_0; \theta_0 \rangle \rightarrow_{AS1}{}^{\mathcal{R}_3} \langle \mathcal{Q}_3; \theta_0\theta_3 \rangle$, donde $\theta_3 = mgu(\{A = H_1\sigma\})$. Finalmente, ya que $\theta_1\theta_2 = \sigma\theta_3$, entonces $\theta_0\theta_1\theta_2 = \theta_0\sigma\theta_3$, y como $Dom(\sigma) \cap \mathcal{V}ar(\mathcal{Q}_0) = \varnothing$ y $Dom(\sigma) \cap Dom(\theta_0) = \varnothing$, resulta que $\theta_0\theta_1\theta_2 = \theta_0\theta_3$ $[\mathcal{V}ar(\mathcal{Q}_0)]$, como deseábamos probar.

($\Leftarrow$) El recíproco puede demostrarse de manera análoga, explotando la equivalencia entre $\theta_1\theta_2$ y $\sigma\theta_3$. □

Ahora, estamos en condiciones de probar la corrección fuerte del desplegado operacional del lenguaje multi-adjunto.

Teorema 5.4.2 (Corrección Parcial Fuerte)**.** *Sea $\mathcal{P}$ un programa multi-adjunto y $\mathcal{Q}$ un objetivo. Si $\mathcal{P}'$ es un programa obtenido por desplegado operacional de $\mathcal{P}$, entonces,*

$$\langle \mathcal{Q}; id \rangle \rightarrow_{AS}{}^* \langle @(r_1, \ldots, r_n); \theta \rangle \text{ en } \mathcal{P}$$

si

$$\langle \mathcal{Q}; id \rangle \rightarrow_{AS}{}^* \langle @(r_1, \ldots, r_n); \theta' \rangle \text{ en } \mathcal{P}'$$

donde $r_i \in L$ para todo $i \in \{1, \ldots, n\}$ y $\theta = \theta'[\mathcal{V}ar(\mathcal{Q})]$.

Demostración. Sea $\mathcal{D}' : [\langle \mathcal{Q}; id\rangle \rightarrow_{AS}{}^{*} \langle @(r_1, \ldots, r_n); \theta\rangle]$ la derivación admisible (genérica) para $\mathcal{Q}$ en $\mathcal{P}'$ que nos proponemos simular construyendo una nueva derivación $\mathcal{D}$ para $\mathcal{Q}$ en $\mathcal{P}$. La construcción de $\mathcal{D}$ se realizará por inducción en k, la longitud de $\mathcal{D}'$. Puesto que el caso base, para $k = 0$, es trivial, procedemos con el caso general cuando $k > 0$. Considérese, entonces, la derivación original $\mathcal{D}' : [\langle \mathcal{Q}; id\rangle \rightarrow_{AS} \langle \mathcal{Q}'; \vartheta\rangle \rightarrow_{AS}{}^{*} \langle @(r_1, \ldots, r_n); \theta'\rangle]$. Si el primer paso de $\mathcal{D}'$ ha sido dado con la segunda o la tercera regla de la Definición 3.1.1, o, incluso si ha sido ejecutado con la primera pero usando una regla perteneciente a $\mathcal{P}$, entonces se sigue el resultado por la hipótesis de inducción. En caso contrario, este paso inicial es del tipo $AS1$ usando una regla $\mathcal{R}'$ que ha sido obtenida por desplegado de otra regla $\mathcal{R} \in \mathcal{P}$. Puesto que el paso de desplegado ha sido ejecutado con una de las tres reglas de la Definición 3.1.1, consideramos cada caso por separado.

1. Desplegado basado en la Regla 1 de la Definición 3.1.1.
   Sea $\mathcal{R} : \langle H_1 \leftarrow_1 \mathcal{B}_1[A_1]; v_1\rangle \in \mathcal{P}$ y $\mathcal{R}_2 : \langle H_2 \leftarrow_2 \mathcal{B}_2; v_2\rangle \in \mathcal{P}$ tal que, por desplegado de $\mathcal{R}$ con respecto a $\mathcal{R}_2$ usando la primera regla de la Definición 3.1.1, obtenemos: $\mathcal{R}' : \langle (H_1 \leftarrow_1 \mathcal{B}_1[A_1/(v_2 \&_2 \mathcal{B}_2)])\sigma; v_1\rangle \in \mathcal{P}'$. Si suponemos que A es el átomo seleccionado en $\mathcal{Q}$ al construir $\mathcal{D}'$, entonces $\langle \mathcal{Q}[A]; id\rangle \rightarrow_{AS1}{}^{\mathcal{R}'} \langle (\mathcal{Q}[A/(v_1 \&_1 \mathcal{B}_1[A_1/(v_2 \&_2 \mathcal{B}_2)])])\sigma\gamma; \sigma\gamma\rangle \rightarrow_{AS}{}^{*} \langle @(r_1, \ldots, r_n); \theta'\rangle$. Ahora, el primer paso de $\mathcal{D}'$ puede simularse en la derivación $\mathcal{D}$ usando las reglas $\mathcal{R}$ y $\mathcal{R}_2$ de $\mathcal{P}$ como sigue: $\langle \mathcal{Q}[A]; id\rangle \rightarrow_{AS1}{}^{\mathcal{R}} \langle (\mathcal{Q}[A/(v_1 \&_1 \mathcal{B}_1[A_1])])\alpha; \alpha\rangle \rightarrow_{AS1}{}^{\mathcal{R}_2} \langle (\mathcal{Q}[A/(v_1 \&_1 \mathcal{B}_1[A_1/(v_2 \&_2 \mathcal{B}_2)])])\alpha\beta; \alpha\beta\rangle$. Por el Lema 5.4.1 podemos concluir la igualdad $\alpha\beta = \sigma\gamma[\mathcal{V}ar(\mathcal{Q})]$, y por tanto el tercer estado de la derivación $\mathcal{D}$ coincide sintácticamente con el segundo de $\mathcal{D}'$. Además, por la hipótesis de inducción $\theta = \theta'[\mathcal{V}ar(\mathcal{Q})]$ y, en consecuencia, las derivaciones admisibles completas $\mathcal{D}$ y $\mathcal{D}'$ son equivalentes, como deseábamos probar.

2. Desplegado basado en la Regla 2 de la Definición 3.1.1.
   Sea $\mathcal{R} : \langle H_1 \leftarrow_1 \mathcal{B}_1[A_1]; v_1\rangle \in \mathcal{P}$ y $\mathcal{R}_2 : \langle H_2 \leftarrow; v_2\rangle \in \mathcal{P}$ tal que, por desplegado de $\mathcal{R}$ con respecto a $\mathcal{R}_2$ usando la segunda regla de la Definición 3.1.1, resulta $\mathcal{R}' : \langle (H_1 \leftarrow_1 \mathcal{B}_1[A_1/v_2])\sigma; v_1\rangle \in \mathcal{P}'$. Así, $\mathcal{D}'$ tiene la forma:

   $$\langle \mathcal{Q}[A]; id\rangle \rightarrow_{AS1}{}^{\mathcal{R}'} \langle (\mathcal{Q}[A/(v_1 \&_1 \mathcal{B}_1[A_1/v_2])])\sigma\gamma; \sigma\gamma\rangle \rightarrow_{AS}{}^{*} \langle @(r_1, \ldots, r_n); \theta'\rangle$$

   Ahora, el primer paso de $\mathcal{D}'$ puede simularse en la derivación $\mathcal{D}$ usando $\mathcal{R}$ y $\mathcal{R}_2$ como sigue: $\langle \mathcal{Q}[A]; id\rangle \rightarrow_{AS1}{}^{\mathcal{R}} \langle (\mathcal{Q}[A/(v_1 \&_1 \mathcal{B}_1[A_1])])\alpha; \alpha\rangle \rightarrow_{AS2}{}^{\mathcal{R}_2} \langle (\mathcal{Q}[A/(v_1 \&_1 \mathcal{B}_1[A_1/v_2])])\alpha\beta; \alpha\beta\rangle$. Por el Lema 5.4.1 concluimos que $\alpha\beta = \sigma\gamma[\mathcal{V}ar(\mathcal{Q})]$, y

por tanto el tercer estado de $\mathcal{D}$ coincide sintácticamente con el segundo de $\mathcal{D}'$. Además, por la hipótesis de inducción $\theta = \theta'[\mathcal{V}ar(\mathcal{Q})]$ y en consecuencia las derivaciones admisibles completas $\mathcal{D}$ y $\mathcal{D}'$ son equivalentes, como deseábamos demostrar.

3. Desplegado basado en la Regla 3 de la Definición 3.1.1.
   Sea $\mathcal{R} : \langle H_1 \leftarrow_1 \mathcal{B}_1[A_1]; v_1\rangle \in \mathcal{P}$ tal que el átomo A_1 seleccionado en $\mathcal{B}_1$ no unifica con la cabeza de ninguna regla de $\mathcal{P}$ y, en consecuencia, por desplegado de $\mathcal{R}$ usando la tercera regla de la Definición 3.1.1, obtenemos la nueva regla desplegada $\mathcal{R}' : \langle H_1 \leftarrow_1 \mathcal{B}_1[A_1/\bot]; v_1\rangle \in \mathcal{P}'$. Entonces, la derivación $\mathcal{D}'$ tiene la siguiente forma:

   $$\langle \mathcal{Q}[A]; id\rangle \rightarrow_{AS1}{}^{\mathcal{R}'} \langle (\mathcal{Q}[A/(v_1 \&_1 \mathcal{B}_1[A_1/\bot])])\alpha; \alpha\rangle \rightarrow_{AS}{}^{*} \langle @(r_1, \ldots, r_n); \theta'\rangle.$$

   Ahora, el primer paso de $\mathcal{D}'$ puede simularse en $\mathcal{P}$ dando dos pasos admisibles en $\mathcal{D}$: el primero con la regla 1 usando $\mathcal{R}$ y el segundo con la regla 3. Es decir: $\langle \mathcal{Q}[A]; id\rangle \rightarrow_{AS1}{}^{\mathcal{R}} \langle (\mathcal{Q}[A/(v_1 \&_1 \mathcal{B}_1[A_1])])\alpha; \alpha\rangle \rightarrow_{AS3} \langle (\mathcal{Q}[A/(v_1 \&_1 \mathcal{B}_1[A_1/\bot])])\alpha; \alpha\rangle$.

   Puesto que este último estado coincide sintácticamente con el segundo de $\mathcal{D}'$, se satisface el resultado por la hipótesis de inducción.

□

Procemos ahora con el complementario del teorema previo, esto es, abordamos la completitud fuerte de la transformación de desplegado operacional de programas lógicos multi-adjuntos.

Teorema 5.4.3 (Completitud fuerte)**.** *Sea $\mathcal{P}$ un programa multi-adjunto y $\mathcal{Q}$ un objetivo. Si $\mathcal{P}'$ es un programa obtenido por desplegado operacional de $\mathcal{P}$, entonces*

$$\langle \mathcal{Q}; id\rangle \rightarrow_{AS}{}^{*} \langle @(r_1, \ldots, r_n); \theta'\rangle \quad \textit{en } \mathcal{P}'$$

si

$$\langle \mathcal{Q}; id\rangle \rightarrow_{AS}{}^{*} \langle @(r_1, \ldots, r_n); \theta\rangle \quad \textit{en } \mathcal{P}$$

donde $r_i \in L$ para todo $i \in \{1, \ldots, n\}$ y $\theta' = \theta[\mathcal{V}ar(\mathcal{Q})]$.

Demostración. La prueba consiste en simular en $\mathcal{P}'$ una derivación admisible reordenada, ejecutada originalmente en el programa $\mathcal{P}$. Por tanto, consideremos la siguiente derivación admisible (genérica) de k–pasos para $\mathcal{Q}$ en $\mathcal{P}$,

$$\mathcal{D}_0 : \ [\langle \mathcal{Q}; id\rangle \rightarrow_{AS}{}^{k} \langle @(r_1, \ldots, r_n); \theta_0\rangle]$$

Supongamos ahora que $\mathcal{R} \in \mathcal{P}$ es la regla desplegada en $\mathcal{P}$ que obviamente no pertenece a $\mathcal{P}'$. Cualquier paso dado con la regla $\mathcal{R}$ en $\mathcal{D}_0$ introduce una instancia del cuerpo de $\mathcal{R}$ en el siguiente estado de la derivación. Puesto que estamos considerando una derivación admisible de éxito, el cuerpo *instanciado* de $\mathcal{R}$ debe reducirse necesariamente en el siguiente paso immediato o en uno subsiguiente. Para el segundo caso, podemos intercambiar el paso dado con la regla $\mathcal{R}$ y el siguiente, aplicando el Lema de Conmutación 3.1.10 para el lenguaje multi-adjunto.

Además, aplicando repetidamente este lema, podemos obtener una nueva derivación admisible de k–pasos en $\mathcal{P}$

$$\mathcal{D} : [\langle \mathcal{Q}; id\rangle \rightarrow_{AS}{}^{k} \langle @(r_1, \ldots, r_n); \theta\rangle]$$

verificando $\theta = \theta_0[\mathcal{V}ar(\mathcal{Q})]$, donde cualquier paso (si existe) que use la regla $\mathcal{R}$ desplegada en $\mathcal{P}$, es seguido de otro paso que explota un átomo que ha sido justamente introducido por el paso previo (es decir, perteneciente al cuerpo instanciado de $\mathcal{R}$). Decimos que $\mathcal{D}$ es una derivación admisible reordenada con respecto a la regla $\mathcal{R}$.

Ahora, y de manera similar a cómo procedimos en el teorema previo, vamos a simular $\mathcal{D}$ en $\mathcal{P}'$ construyendo una nueva derivación $\mathcal{D}'$ que use reglas de $\mathcal{P}'$, siguiendo un esquema totalmente análogo al considerado en el Teorema 5.4.2, pero invirtiendo ahora el uso de los términos $\mathcal{P}$ y $\mathcal{P}'$ (y sus relacionados).

La construcción de $\mathcal{D}'$ se realiza por inducción en la longitud, k, de $\mathcal{D}$. Puesto que el caso base, para $k = 0$, es trivial, abordamos el caso general cuando $k > 0$. Sea, entonces, $\mathcal{D} : [\langle \mathcal{Q}; id\rangle \rightarrow_{AS} \langle \mathcal{Q}'; \vartheta\rangle \rightarrow_{AS}{}^{*} \langle @(r_1, \ldots, r_n); \theta\rangle]$. Si el primer paso de $\mathcal{D}$ ha sido dado con la segunda o tercera regla de la Definición 3.1.1, o incluso si ha sido ejecutado con la primera pero usando una regla que también pertenece a $\mathcal{P}'$, se satisface el resultado por la hipótesis de inducción. En otro caso, este paso inicial es del tipo $AS1$ usando una regla $\mathcal{R}$ que, una vez desplegada, genera la nueva regla transformada $\mathcal{R}' \in \mathcal{P}'$. Teniendo en cuenta que el paso de desplegado ha sido ejecutado con una de las tres reglas de la Definición 3.1.1, tratamos cada caso por separado.

1. Desplegado basado en la Regla 1 de la Definición 3.1.1.
   Sea $\mathcal{R} : \langle H_1 \leftarrow_1 \mathcal{B}_1[A_1]; v_1\rangle \in \mathcal{P}$ y $\mathcal{R}_2 : \langle H_2 \leftarrow_2 \mathcal{B}_2; v_2\rangle \in \mathcal{P}$ tal que, por

desplegado de $\mathcal{R}$ con respecto a $\mathcal{R}_2$ usando la primera regla de la Definición 3.1.1, obtenemos $\mathcal{R}' : \langle (H_1 \leftarrow_1 \mathcal{B}_1[A_1/(v_2 \&_2 \mathcal{B}_2)])\sigma; v_1\rangle \in \mathcal{P}'$. Supongamos que A es el átomo seleccionado en $\mathcal{Q}$ cuando construimos la derivación $\mathcal{D}$, y teniendo en cuenta que $\mathcal{D}$ es una derivación admisible (de éxito) reordenada con respecto a la regla $\mathcal{R}$, entonces tiene la siguiente forma: $\langle \mathcal{Q}[A]; id\rangle \rightarrow_{AS1}{}^{\mathcal{R}} \langle (\mathcal{Q}[A/(v_1 \&_1 \mathcal{B}_1[A_1])])\alpha; \alpha\rangle \rightarrow_{AS1}{}^{\mathcal{R}_2} \langle (\mathcal{Q}[A/(v_1 \&_1 \mathcal{B}_1[A_1/(v_2 \&_2 \mathcal{B}_2)])])\alpha\beta; \alpha\beta\rangle \rightarrow_{AS}{}^* \langle @(r_1, \ldots, r_n); \theta\rangle$. Pero los dos primeros pasos de $\mathcal{D}$ pueden simularse en $\mathcal{P}'$ usando $\mathcal{R}'$ en $\mathcal{D}'$ del siguiente modo: $\langle \mathcal{Q}[A]; id\rangle \rightarrow_{AS1}{}^{\mathcal{R}'} \langle (\mathcal{Q}[A/(v_1 \&_1 \mathcal{B}_1[A_1/(v_2 \&_2 \mathcal{B}_2)])])\ \sigma\gamma; \sigma\gamma\rangle \rightarrow_{AS}{}^* \langle @(r_1, \ldots, r_n); \theta'\rangle$. Por el Lema 5.4.1 tenemos $\alpha\beta = \sigma\gamma[\mathcal{V}ar(\mathcal{Q})]$, y por tanto el tercer estado de $\mathcal{D}$ coincide sintácticamente con el segundo de $\mathcal{D}'$. Además, por la hipótesis de inducción, $\theta = \theta'[\mathcal{V}ar(\mathcal{Q})]$ y en consecuencia las derivaciones admisibles completas $\mathcal{D}$ y $\mathcal{D}'$ son equivalentes, como deseábamos demostrar.

2. Desplegado basado en la Regla 2 de la Definición 3.1.1.
   Sea $\mathcal{R} : \langle H_1 \leftarrow_1 \mathcal{B}_1[A_1]; v_1\rangle \in \mathcal{P}$ y $\mathcal{R}_2 : \langle H_2 \leftarrow; v_2\rangle \in \mathcal{P}$ tal que, por desplegado de la regla $\mathcal{R}$ con respecto a $\mathcal{R}_2$ usando la segunda regla de la Definición 3.1.1, obtenemos: $\mathcal{R}' : \langle (H_1 \leftarrow_1 \mathcal{B}_1[A_1/v_2])\sigma; v_1\rangle \in \mathcal{P}'$. Puesto que $\mathcal{D}$ es una derivación admisible (de éxito) reordenada con respecto a la regla $\mathcal{R}$, y suponiendo que A es el átomo seleccionado en $\mathcal{Q}$, entonces $\mathcal{D}$ tiene la forma: $\langle \mathcal{Q}[A]; id\rangle \rightarrow_{AS1}{}^{\mathcal{R}} \langle (\mathcal{Q}[A/(v_1 \&_1 \mathcal{B}_1[A_1])])\alpha; \alpha\rangle \rightarrow_{AS2}{}^{\mathcal{R}_2} \langle (\mathcal{Q}[A/(v_1 \&_1 \mathcal{B}_1[A_1/v_2])])\ \alpha\beta; \alpha\beta\rangle \rightarrow_{AS}{}^* \langle @(r_1, \ldots, r_n); \theta\rangle$. Ahora, los dos primeros pasos de $\mathcal{D}$ pueden simularse en el programa transformado $\mathcal{P}'$ usando $\mathcal{R}'$ como sigue:

   $$\langle \mathcal{Q}[A]; id\rangle \rightarrow_{AS1}{}^{\mathcal{R}'} \langle (\mathcal{Q}[A/(v_1 \&_1 \mathcal{B}_1[A_1/v_2])])\sigma\gamma; \sigma\gamma\rangle \rightarrow_{AS}{}^* \langle @(r_1, \ldots, r_n); \theta'\rangle$$

   Por el Lema 5.4.1 podemos concluir que $\alpha\beta = \sigma\gamma[\mathcal{V}ar(\mathcal{Q})]$, y por tanto el tercer estado de $\mathcal{D}$ coincide sintácticamente con el segundo de $\mathcal{D}'$. Además, por la hipótesis de inducción, $\theta = \theta'[\mathcal{V}ar(\mathcal{Q})]$ y resulta que las derivaciones admisibles completas $\mathcal{D}$ y $\mathcal{D}'$ son equivalentes, como deseábamos.

3. Desplegado basado en Regla 3.
   Sea $\mathcal{R} : \langle H_1 \leftarrow_1 \mathcal{B}_1[A_1]; v_1\rangle \in \mathcal{P}$ tal que, el átomo seleccionado A_1 en $\mathcal{B}_1$ no unifica con la cabeza de ninguna regla de $\mathcal{P}$ y, por tanto, por desplegado de $\mathcal{R}$ usando la tercera regla de la Definición 3.1.1, obtenemos la nueva regla

desplegada $\mathcal{R}' : \langle H_1 \leftarrow_1 \mathcal{B}_1[A_1/\bot]; v_1\rangle \in \mathcal{P}'$. Suponiendo que A es el átomo seleccionado en $\mathcal{Q}$ cuando construimos la derivación $\mathcal{D}$, y teniendo en cuenta que $\mathcal{D}$ es una derivación admisible reordenada (de éxito) con respecto a la regla $\mathcal{R}$, tiene la forma: $\langle \mathcal{Q}[A]; id\rangle \rightarrow_{AS1}{}^{\mathcal{R}} \langle (\mathcal{Q}[A/(v_1 \&_1 \mathcal{B}_1[A_1])])\alpha; \alpha\rangle \rightarrow_{AS3} \langle (\mathcal{Q}[A/(v_1 \&_1 \mathcal{B}_1[A_1/\bot])])\alpha; \alpha\rangle \rightarrow_{AS}{}^{*} \langle @(r_1, \ldots, r_n); \theta\rangle$. Ahora, los dos primeros pasos de $\mathcal{D}$ pueden simularse en la derivación $\mathcal{D}'$ usando la regla $\mathcal{R}'$ como sigue: $\langle \mathcal{Q}[A]; id\rangle \rightarrow_{AS1}{}^{\mathcal{R}'} \langle (\mathcal{Q}[A/(v_1 \&_1 \mathcal{B}_1[A_1/\bot])])\alpha; \alpha\rangle$.

Puesto que este último estado coincide sintácticamente con el tercero de $\mathcal{D}$, se obtiene el resultado por la hipótesis de inducción.

□

Por último, estamos en condiciones de formalizar y demostrar las mejores propiedades del desplegado operacional (a saber, su corrección fuerte y la mejora en eficiencia de los programas transformados) en los siguiente términos:

Teorema 5.4.4 (Corrección Total Fuerte del Desplegado Operacional)**.** *Sea $\mathcal{P}$ un programa multi-adjunto, y sea $\mathcal{Q}$ un objetivo. Si $\mathcal{P}'$ es un programa obtenido por desplegado operacional de $\mathcal{P}$, entonces,*

$$\langle \mathcal{Q}; id\rangle \rightarrow_{AS}{}^{n} \langle @(r_1, \ldots, r_k); \theta\rangle \quad \textit{en} \quad \mathcal{P}$$

si, y sólo si,

$$\langle \mathcal{Q}; id\rangle \rightarrow_{AS}{}^{m} \langle @(r_1, \ldots, r_k); \theta'\rangle \quad \textit{en} \quad \mathcal{P}'$$

donde $r_i \in L$ para todo $i \in \{1, \ldots, k\}$, $\theta = \theta'[\mathcal{V}ar(\mathcal{Q})]$ y $m \leq n$.

Demostración. Las dos implicaciones del Teorema 5.4.4 puede probarse como sigue:

- La corrección total fuerte de la transformación, es decir, la equivalencia de las respuestas computadas difusas obtenidas cuando ejecutamos un objetivo con respecto a los programas original y desplegado, es imediata sin más que aplicar los Teoremas 5.4.2 y 5.4.3.
- Observando la reducción de la longitud de las derivaciones admisibles en los programas transformados, hemos justificado en las demostraciones de los Teoremas 5.4.2 y 5.4.3 que cualquier paso admisible difuso dado con una nueva regla obtenida por desplegado difuso, subsume dos pasos admisibles difusos dados con reglas del programa original, lo que confirma que $m \leq n$, como queríamos probar.

□

5.5. Propiedades de corrección del desplegado interpretativo del lenguaje multi-adjunto

Abordamos ahora las propiedades de corrección del desplegado interpretativo, para obtener en el siguiente teorema un resultado análogo al del Teorema 5.4.4. Los beneficios en este caso son sólo apreciados durante la segunda fase (interpretativa) de la ejecución de objetivos. En este sentido, aunque no podemos hablar propiamente de preservación de respuestas computadas admisibles[2], probaremos que es posible conservar el conjunto de respuestas computadas difusas asociadas al objetivo dado (cuando las a.c.a.'s se interpretan con respecto al mismo retículo usado durante el proceso de desplegado interpretativo). Observando la reducción de la longitud de las derivaciones en los programas transformados, el desplegado interpretativo permite reducir el número de pasos interpretativos necesarios para resolver un objetivo, de manera similar a como el desplegado operacional reduce el número de pasos admisibles. En el último resultado de esta sección abordamos las propiedades de corrección del sistema de transformación que se obtiene al combinar los dos tipos de desplegado estudiados para el lenguaje multi-adjunto: el operacional y el interpretativo.

Teorema 5.5.1 (Corrección Total Fuerte del Desplegado Interpretativo)**.** *Sea $\mathcal{P}$ un programa y sea $\mathcal{Q}$ un objetivo. Si $\mathcal{P}'$ es un programa obtenido por desplegado interpretativo de $\mathcal{P}$, entonces,*

$$\langle \mathcal{Q}; id \rangle \rightarrow^{n}_{AS/IS} \langle r; \theta \rangle \text{ en } \mathcal{P}$$

si, y sólo si,

$$\langle \mathcal{Q}; id \rangle \rightarrow^{m}_{AS/IS} \langle r; \theta \rangle \text{ en } \mathcal{P}'$$

donde

1. *$r \in L$, siendo $(L, \leq)$ el retículo asociado a $\mathcal{P}$ usado durante el proceso de desplegado interpretativo, y*
2. *$m \leq n$.*

Demostración. Para demostrar este teorema, abordaremos por separado las dos implicaciones que recoge la equivalencia formulada.

Corrección Parcial Fuerte ($\Leftarrow$).

[2]Tal como concretamos posteriormente en el Ejemplo 5.5.2.

Considérese la derivación completa (genérica) para $\mathcal{Q}$ en $\mathcal{P}'$

$$\mathcal{D}' : [\langle Q; id\rangle \rightarrow_{AS}^{k} \langle e'; \theta\rangle \rightarrow_{IS}^{l} \langle r; \theta\rangle]$$

donde $k + l = m$, que nos proponemos simular usando reglas del programa $\mathcal{P}$. Consideremos también que la regla

$$\mathcal{R}' : \langle A \leftarrow_i \mathcal{B}[r_1 @ r_2 / \dot{@}(r_1, r_2)];\ v\rangle$$

ha sido obtenida por desplegado interpretativo de la regla $\mathcal{R} : \langle A \leftarrow_i \mathcal{B};\ v\rangle$. Recordemos que $\mathcal{R} \in \mathcal{P}$ y $\mathcal{R}' \in \mathcal{P}'$, pero $\mathcal{R} \notin \mathcal{P}'$ y $\mathcal{R}' \notin \mathcal{P}$. Puesto que el desplegado interpretativo sólo afecta a expresiones con conectivas y elementos pertenecientes a L, el conjunto de átomos de las cabezas y cuerpos de $\mathcal{R}$ y $\mathcal{R}'$ son exactamente el mismo. Por tanto, podemos contruir la siguiente derivación admisible completa

$$\mathcal{D} : [\langle Q; id\rangle \rightarrow_{AS}^{k} \langle e; \theta\rangle]$$

en $\mathcal{P}$, donde:

- la longitud de $\mathcal{D}$ coincide con el número de pasos admisibles, k, aplicados en la derivación $\mathcal{D}'$,
- el átomo reducido en el paso i-esimo de $\mathcal{D}$ coincide con el átomo reducido en el paso i-esimo de $\mathcal{D}'$, para $1 \leq i \leq k$, y
- la regla de programa usada en el paso i-esimo de $\mathcal{D}$ es la misma que la usada en el paso i-esimo de $\mathcal{D}'$, para $1 \leq i \leq k$, excepto cuando ésta última es $\mathcal{R}'$: en este caso, usamos $\mathcal{R}$ en $\mathcal{D}$.

Téngase en cuenta que las respuestas computadas admisibles asociadas a ambas derivaciones no son exactamente las mismas (lo que muestra que el desplegado interpretativo, a diferencia del desplegado operacional no preserva las a.c.a.'s) pero sí están fuertemente relacionadas: ambas comparten la misma sustitución θ, mientras que las expresiones e y e' son muy similares. En efecto, cualquier paso admisible dado con la regla $\mathcal{R}'$ en $\mathcal{D}'$, introduce un valor (proveniente de la interpretación de una expresión) de la forma $\dot{@}(r_1, r_2)$ en e', mientras que los correspondientes pasos dados con la regla $\mathcal{R}$ en $\mathcal{D}$, deja descendientes de la (todavía no interpretada) expresión $r_1 @ r_2$ en e.

Formalmente, si P_j es el conjunto de posiciones de las j ocurrencias de $r_1 @ r_2$ introducidas en e por la aplicación de j pasos admisibles usando $\mathcal{R}$ en $\mathcal{D}$ (o, equivalentemente, si P_j es el conjunto de posiciones de las j ocurrencias de $\dot{@}(r_1, r_2)$

introducidas en e' por la aplicación de j pasos admisibles usando $\mathcal{R}'$ en $\mathcal{D}'$), entonces $e' = e[r_1@r_2/\dot{@}(r_1, r_2)]_{P_j}$. Por tanto, e' puede verse como una versión *parcialmente interpretada* de e, y entonces es fácil observar que, sin más que aplicar varios pasos interpretativos sobre las correspondientes j ocurrencias de $r_1@r_2$ en e, podemos reemplazarlas por $\dot{@}(r_1, r_2)$, hasta alcanzar la expresión deseada e'. A partir de aquí, podemos finalizar las derivaciones completas en ambos programas aplicando los mismos pasos interpretativos, hasta obtener la misma respuesta computada difusa $\langle r, \theta\rangle$.

Observando la reducción de la longitud de la derivación en los programas transformados, hemos visto que cualquier paso dado con la regla desplegada $\mathcal{R}'$ en la derivación $\mathcal{D}'$, evita un posterior paso interpretativo que, de otra manera, es inevitable cuando construimos una derivación usando reglas del programa original $\mathcal{P}$. Así, la derivación completa que simula $\mathcal{D}'$ en $\mathcal{P}$ tiene la forma:

$$[\langle Q; id\rangle \rightarrow_{AS}^{k} \langle e; \theta\rangle \rightarrow_{IS}^{j} \langle e'; \theta\rangle \rightarrow_{IS}^{l} \langle r; \theta\rangle]$$

donde $k + j + l = n$, lo que implica que $m \leq n$ (recordemos que $m = k + l$) como deseábamos probar.

Completitud Fuerte ($\Rightarrow$).

Aunque esta implicación puede probarse de manera similar a la anterior, preferimos detallarla puesto que introduce algunas sutilezas relativas al orden en que se ejecutan los pasos de computación en la derivación original. Sea la derivación completa (genérica) para $\mathcal{Q}$ en $\mathcal{P}$

$$\mathcal{D} : [\langle Q; id\rangle \rightarrow_{AS}^{k} \langle e; \theta\rangle \rightarrow_{IS}^{l} \langle r; \theta\rangle]$$

donde $k + l = n$, que nos proponemos simular construyendo una nueva derivación $\mathcal{D}'$ en $\mathcal{P}'$. Consideremos también que la regla $\mathcal{R} : \langle A \leftarrow_i \mathcal{B}[r_1@r_2] \ ; v\rangle \in \mathcal{P}$ tal que, por desplegado interpretativo de $\mathcal{R}$ en el programa $\mathcal{P}$, obtenemos $\mathcal{R}' : \langle A \leftarrow_i \mathcal{B}[r_1@r_2/\dot{@}(r_1, r_2)]; \ v\rangle$. Recordemos que $\mathcal{R} \in \mathcal{P}$ y $\mathcal{R}' \in \mathcal{P}'$, pero $\mathcal{R} \notin \mathcal{P}'$ y $\mathcal{R}' \notin \mathcal{P}$. Puesto que el desplegado interpretativo sólo afecta a expresiones con conectivas y elementos pertenecientes a L, coinciden los conjuntos de átomos formados por las cabezas y cuerpos de $\mathcal{R}$ y $\mathcal{R}'$, respectivamente. Además, puesto que los pasos interpretativos no dependen de ningún tipo de función de selección (o regla de computación), podemos suponer sin pérdida de generalidad que los primeros pasos en la fase interpretativa en $\mathcal{D}$ se aplican a cada expresión de la forma $r_1@r_2$ introducida

en e por pasos admisibles previos dados con la regla $\mathcal{R}$. Esto es, con toda seguridad podemos suponer que la derivación $\mathcal{D}$ tiene la forma

$$\mathcal{D} : [\langle Q; id\rangle \rightarrow_{AS}^{k} \langle e; \theta\rangle \rightarrow_{IS}^{l_1} \langle e'; \theta\rangle \rightarrow_{IS}^{l_2} \langle r; \theta\rangle]$$

con $l_1 + l_2 = l$. En consecuencia, podemos construir fácilmente la siguiente derivación admisible $\mathcal{D}' : [\langle Q; id\rangle \rightarrow_{AS}^{k} \langle e'; \theta\rangle]$ en $\mathcal{P}'$, donde:

- la longitud de $\mathcal{D}'$ coincide con el número de pasos admisibles, k, aplicados en la derivación $\mathcal{D}$,
- el átomo reducido en el paso i-esimo de $\mathcal{D}'$ coincide con el átomo reducido en el paso i-esimo de $\mathcal{D}$, para $1 \leq i \leq k$, y
- la regla usada en el paso i-esimo de $\mathcal{D}'$ es la misma que la usada en el paso i-esimo de $\mathcal{D}$, para $1 \leq i \leq k$, excepto cuando esta última es $\mathcal{R}$: en este caso, usamos $\mathcal{R}'$ en $\mathcal{D}'$.

Obsérvese que las a.c.a.'s asociadas a ambas derivaciones no son exactamente las mismas (lo que pone de manifiesto que el desplegado interpretativo, a diferencia del operacional, no conserva las a.c.a.'s) pero están muy relacionadas: ambas comparten la misma sustitución θ, y las expresiones e y e' son muy similares. En efecto, cualquier paso admisible dado con la regla $\mathcal{R}'$ en $\mathcal{D}'$, introduce un valor de la forma $\dot{@}(r_1, r_2)$ en e', mientras que los correspondientes pasos dados con la regla $\mathcal{R}$ en $\mathcal{D}$, dejan descendientes de la expresión (todavía no interpretada) $r_1@r_2$ en e.

Formalmente, si P_j es el conjunto de posiciones de las j ocurrencias de $r_1@r_2$ introducidas e aplicando j pasos admisibles usando $\mathcal{R}$ en $\mathcal{D}$ (o, equivalentemente, si P_j es el conjunto de posiciones de las j ocurrencias de $\dot{@}(r_1, r_2)$ introducidas en e' por la aplicación de j pasos admisibles usando $\mathcal{R}'$ en $\mathcal{D}'$), entonces $e' = e[r_1@r_2/\dot{@}(r_1, r_2)]_{P_j}$. Por tanto, e' puede verse como una versión *parcialmente interpretada* de e, y entonces resulta sencillo ver que, mediante un simple ejecución de varios pasos interpretativos sobre las correspondientes j ocurrencias de $r_1@r_2$ en e, podemos reemplazarlas por $\dot{@}(r_1, r_2)$, hasta alcanzar la expresión deseada e'. Por tanto, podemos finalizar las derivaciones completas en ambos programas aplicando los mismos pasos interpretativos, hasta obtener la misma respuesta computada difusa $\langle r, \theta\rangle$.

De manera similar a como vimos en la prueba de la corrección, se reduce la longitud de la derivación en los programas transformados. Hemos visto que cualquier paso

dado con la regla desplegada $\mathcal{R}'$ en la derivación $\mathcal{D}'$ evita un posterior paso interpretativo que, de otro modo, es inevitable cuando construimos una derivación usando reglas del programa original $\mathcal{P}$. De este modo, la derivación completa simulando $\mathcal{D}$ en $\mathcal{P}'$ tiene la forma:

$$[\langle Q; id\rangle \rightarrow_{AS}^{k} \langle e'; \theta\rangle \rightarrow_{IS}^{l_2} \langle r; \theta\rangle]$$

donde $l_2 \leq l$ como dijimos, y por tanto $m = k + l_2 \leq k + l = n$, lo que termina la prueba de la completitud fuerte.
Finalmente, la corrección total fuerte del desplegado interpretativo se sigue de ambos, la corrección parcial fuerte ($\Leftarrow$) y la completitud fuerte ($\Rightarrow$), como deseábamos probar. □

Por último, y completando el resultado anterior, en el siguiente ejemplo confirmamos que, tal como adelantábamos al principio de la sección, el desplegado interpretativo no conserva necesariamente las respuestas computadas admisibles.

Ejemplo 5.5.2. *Sea $\mathcal{P}$ el programa multi-adjunto formado por la única regla*

$$\mathcal{R} : p(X) \leftarrow 0.5 \&_{\texttt{prod}} 0.6 \quad \textit{with } 1$$

Por desplegado interpretativo de $\mathcal{R}$ obtenemos la nueva regla desplegada

$$\mathcal{R}' : p(X) \leftarrow 0.3 \quad \textit{with } 1$$

siendo $\mathcal{P}' = (\mathcal{P} - \{\mathcal{R}\}) \cup \{\mathcal{R}'\}$ el programa transformado. Entonces, podemos generar la siguiente derivación para el programa $\mathcal{P}$ y el objetivo $p(a)$

$$\langle p(a); id\rangle \ \rightarrow_{IS}^{\mathcal{R}} \ \langle 1 \& 0.5 \&_{\texttt{prod}} 0.6; \{X/a\}\rangle \rightarrow_{IS} \ \langle 1 \& 0.3; \{X/a\}\rangle \rightarrow_{IS} \ \langle 0.3; \{X/a\}\rangle$$

mientras que para el programa $\mathcal{P}'$ y el mismo objetivo tenemos

$$\langle p(a); id\rangle \ \rightarrow_{IS}^{\mathcal{R}'} \ \langle 1 \& 0.3; \{X/a\}\rangle \rightarrow_{IS} \ \langle 0.3; \{X/a\}\rangle$$

En consecuencia, para el objetivo $p(a)$, se obtiene la respuesta computada admisible $1 \& 0.5 \&_{\texttt{prod}} 0.6$ en el programa $\mathcal{P}$ que no coincide con la respuesta computada admisible $1 \& 0.3$ en el programa $\mathcal{P}'$, pese a que –según garantiza el teorema anterior– ambas derivaciones conducen a la misma respuesta computada difusa.

5.6. Corrección total fuerte del desplegado operacional/interpretativo

Para finalizar este capítulo, presentamos el siguiente resultado que combina el uso del desplegado operacional/interpretativo considerando una secuencia de transformación de programas $(\mathcal{P}_0, \ldots, \mathcal{P}_k)$, $k \geq 0$. El teorema que sigue formaliza las mejores propiedades del sistema de transformación resultante, a saber, su corrección total fuerte y la garantía de mayor eficiencia de los programas residuales. El resultado se obtiene directamente como un simple corolario de los Teoremas 5.4.4 y 5.5.1.

Teorema 5.6.1 (Corrección Total Fuerte del Sistema de Transformación)**.** *Considérese la secuencia de transformación $(\mathcal{P}_0, \ldots, \mathcal{P}_k)$ donde cada programa de la secuencia, excepto el programa multi-adjunto inicial $\mathcal{P}_0$, se obtiene del inmediato precedente por desplegado operacional/interpretativo. Entonces,*

$$\langle Q; id\rangle \rightarrow^{n}_{AS/IS} \langle r; \theta\rangle \text{ en } \mathcal{P}_0$$

si, y sólo si,

$$\langle Q; id\rangle \rightarrow^{m}_{AS/IS} \langle r; \theta'\rangle \text{ en } \mathcal{P}_k$$

donde

1. $r \in L$, *siendo* $(L, \leq)$ *el retículo asociado a* $\mathcal{P}_0$,
2. $\theta' = \theta[\mathcal{V}ar(Q)]$, *y*
3. $m \leq n$.

5.7. Conclusiones

En este capítulo se han recogido los mejores resultados, de corrección y eficiencia, que cabe esperar acerca de las transformaciones de desplegado difuso abordadas para el lenguaje lef-Prolog (que subsume al lenguaje de Vojtáš) y para el lenguaje multi-adjunto.

Todos estos resultados se obtienen por primera vez en el contexto difuso y, además, los correspondientes al lenguaje lef-Prolog extienden a otros análogos que obtuvimos también para lenguajes más primitivos (f-Prolog de Vojtáš y lf-Prolog).

En virtud de los citados resultados, se garantiza que, desde el punto de vista teórico, los programas transformados obtienen las mismas respuestas computadas difusas que el programa original; y, desde el punto de vista práctico, se obtiene mejora en eficiencia cuando ejecutamos los programas residuales, puesto que se reduce el número de pasos de computación necesarios para resolver un objetivo.

Hay que tener en cuenta que los resultados relativos al desplegado de programas multi-adjuntos no extienden directamente a los correspondientes del lenguaje lef-Prolog dado que ambos lenguajes poseen distinta semántica procedural, aparte de otras características que los distancian a nivel expresivo y procedural.

Es también de destacar que en el Teorema 5.5.1, de corrección total fuerte del desplegado interpretativo del lenguaje multi-adjunto, contemplamos una demostración novedosa que, en particular, no hace uso de ningún resultado de independencia de la regla de computación, lo cual no tiene precedentes en la literatura relacionada con transformaciones de programas no difusos.

Capítulo 6

Reductantes

6.1. Conceptos básicos

Los reductantes son una interesante herramienta teórica introducida inicialmente en el contexto de la programación lógica generalizada con anotaciones (véase [Kifer y Subrahmanian, 1992]) para demostrar propiedades de corrección y completitud en este marco. Este concepto ha sido recientemente adaptado al entorno más rico y flexible de la programación lógica multi-adjunta (véase [Medina *et al.*, 2001c]), intentando resolver el problema de incompletitud que surge cuando se trabaja en retículos (sólo) parcialmente ordenados.

En efecto, puede ocurrir que sea imposible computar la mayor respuesta correcta, si el retículo $(L, \leq)$ no es totalmente ordenado [Medina *et al.*, 2004]: sean a, b elementos no comparables de L; supongamos que para un objetivo (básico) A existen sólo dos reglas (hechos) $\langle A \leftarrow; a \rangle$ y $\langle B \leftarrow; b \rangle$; la primera regla conduce a la respuesta computada borrosa a (con sustitución vacía); análogamente, la segunda deriva la respuesta computada borrosa b; en consecuencia, por la corrección de la semántica procedural de la programación lógica multi-adjunta (véase el Teorema 3.4.8), a y b son respuestas correctas y por la definición de respuesta correcta, $c = sup\{a, b\} \in L$ es también una respuesta correcta; sin embargo, nuestro principio computacional descrito en la Sección 3.1 nunca devolverá c como respuesta computada, por lo que se pierde la completitud deseada.

Este problema puede resolverse extendiendo el programa original con una regla especial $\langle A \leftarrow sup\{a; b\}; \top \rangle$, muy cercana a la noción de *reductante*, que nos permite

obtener el supremo de todas las contribuciones para el objetivo A.

Por tanto, un programa lógico multi-adjunto, interpretado en un retículo completo, precisa contener este tipo especial de reglas llamadas reductantes a fin de garantizar su completitud (aproximada). Pero esto introduce severos inconvenientes prácticos a la hora de implementar un sistema eficiente y completo de programación lógica multi-adjunta, ya que pueden dispararse tanto el tamaño de los programas, como los tiempos de ejecución. Es decir, la noción de reductante, como hemos considerado, puede introducir una importante pérdida de eficiencia en las implementaciones prácticas. Por tanto, si deseamos desarrollar un sistema completo y eficiente para este entorno de programación, la programación lógica multi-adjunta, resulta crucial definir técnicas adecuadas para el cálculo de reductantes.

En este capítulo introduciremos un nuevo concepto de reductante que tendrá ligeras diferencias sintácticas con el reductante de Medina *et al.* [2004], pero que tiene la ventaja de que puede generalizarse posteriormente usando conceptos de evaluación parcial que facilitarán notablemente su cálculo. Además, demostraremos que este reductante y el reductante original aportan la misma respuesta computada difusa (y con la misma eficiencia) para un objetivo básico A en un programa $\mathcal{P}$ y, por tanto, pueden considerarse equivalentes desde un punto de vista procedural. Demostraremos, asimismo, la equivalencia semántica de ambos reductantes.

Asimismo, introducimos una medida de coste computacional que estimará convenientemente el coste de la fase interpretativa de los programas multi-adjuntos. Este criterio nos será útil para contrastar la eficiencia de la nueva noción de reductante.

Finalmente, para abordar la completitud de la programación lógica multi-adjunta, incorporamos el concepto de compleción de un programa, para mostrar de manera más transparente la utilidad de los reductantes cuando calculamos respuestas computadas. La compleción de un cierto programa será un nuevo programa equivalente (en un cierto sentido) al dado, que sea completo y cuyas reglas son reductantes. Proponemos, además, una nueva semántica operacional que (evitando modificar el programa) simule los efectos del uso de reductantes y, en ejemplos representativos, mostramos las respuestas computadas de un programa y de una compleción del mismo. Además, contemplamos una nueva semántica operacional para la programación lógica multi-adjunta que supera el inconveniente que supone, en la práctica, trabajar con programas infinitos.

Los conceptos y resultados introducidos en este capítulo se recogen en [Julián *et al.*, 2007c,b,a, 2006b].

Iniciamos el desarrollo de estos contenidos refiriendo en primer lugar la definición presentada en [Medina *et al.*, 2004], donde la noción clásica de reductante ha sido adaptada inicialmente al marco de la programación lógica multi-adjunta en los términos que siguen.

Definición 6.1.1 (Reductante de Medina *et al.* [2004])**.** *Sea $\mathcal{P}$ un programa, A átomo básico, y $\langle H_i \leftarrow_i \mathcal{B}_i; v_i\rangle$ el conjunto (no vacío) de reglas de $\mathcal{P}$ cuya cabeza unifica con A (existen θ_i tales que $A = H_i\theta_i$). Un reductante para A en $\mathcal{P}$ es una regla $\langle A \leftarrow @(\mathcal{B}_1, \ldots, \mathcal{B}_n)\theta; \top\rangle$ donde $\theta = \theta_1 \cdots \theta_n$, $\leftarrow$ es cualquier implicación con un conjuntor adjunto, y la función de verdad del agregador $@$ se define como $\dot{@}(b_1, \ldots, b_n) = sup\{v_1 \dot{\&}_1 b_1, \ldots, v_n \dot{\&}_n b_n\}$.*

Por nuestra parte, en la siguiente definición, contemplamos un concepto de reductante que es una ligera modificación del original debido a Medina *et al.* [2004]. En ella, y en lo que sigue, consideraremos que el agregador $@_{sup}$ tiene función de verdad $\dot{@}_{sup}$ definida por $\dot{@}_{sup}(x_1, \ldots, x_n) = sup\{x_1, \ldots, x_n\}$.

Definición 6.1.2 (1-reductante)**.** *Sea $\mathcal{P}$ un programa y sea A un átomo básico. Si $\{\langle H_i \leftarrow_i \mathcal{B}_i; v_i\rangle \in \mathcal{P} : \exists \theta_i / A = H_i\theta_i\}$ es el conjunto (no vacío) de reglas de $\mathcal{P}$ cuya cabeza unifica con A, entonces el* 1-reductante *de A en $\mathcal{P}$ es la regla $\langle A \leftarrow @_{sup}((v_1 \&_1 \mathcal{B}_1)\theta_1, \ldots, (v_n \&_n \mathcal{B}_n)\theta_n); \top\rangle$.*

En el ejemplo que sigue damos un 1-reductante y lo contrastamos con el reductante de la Definición 6.1.1.

Ejemplo 6.1.3. *Dado el retículo $([0,1], \leq)$, donde "$\leq$" es el orden usual, sea $\mathcal{P}$:*

$$\mathcal{R}_1 : \langle p(a) \leftarrow_{\mathtt{L}} q(X, a); \quad 0.7\rangle$$

$$\mathcal{R}_2 : \langle p(a) \leftarrow_{\mathtt{G}} s(Y); \quad 0.5\rangle$$

$$\mathcal{R}_3 : \langle p(Y) \leftarrow ; \quad 0.6\rangle$$

$$\mathcal{R}_4 : \langle p(Y) \leftarrow_{\mathtt{G}} q(b, Y) \&_{\mathtt{L}} t(Y); \quad 0.8\rangle$$

$$\mathcal{R}_5 : \langle q(b, a) \leftarrow ; \quad 0.9\rangle$$

$$\mathcal{R}_6 : \langle s(a) \leftarrow_{\mathtt{G}} t(a); \quad 0.5\rangle$$

$$\mathcal{R}_7 : \langle s(b) \leftarrow ; \quad 0.8\rangle$$

$$\mathcal{R}_8 : \langle t(a) \leftarrow_{\mathtt{L}} p(X); \quad 0.9\rangle$$

un programa lógico multi-adjunto. Para el programa $\mathcal{P}$ y el átomo $p(a)$, teniendo en cuenta que $\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3, \mathcal{R}_4$ dan el conjunto (no vacío) de reglas de $\mathcal{P}$ cuya cabeza unifica con $p(a)$, se obtiene el 1*-reductante:*

$$\langle p(a) \leftarrow @_{sup}(0.7 \&_{\mathtt{L}} q(X_1, a), 0.5 \&_{\mathtt{G}} s(Y_2), 0.6 \& 1, 0.8 \&_{\mathtt{G}} (q(b, a) \&_{\mathtt{L}} t(a)));\ 1\rangle$$

Obsérvense las diferencias sintácticas entre esta regla y el siguiente reductante debido a Medina et al. *[2004].*

$$\langle p(a) \leftarrow @(q(X_1, a), s(Y_2), 0.6 \& 1, q(b, a) \&_{\mathtt{L}} t(a));\ 1\rangle$$

donde $\dot{@}(b_1, b_2, b_3, b_4) = sup\{0.7 \dot{\&}_{\mathtt{L}} b_1, 0.5 \dot{\&}_{\mathtt{G}} b_2, b_3, 0.8 \dot{\&}_{\mathtt{G}} b_4\}$.

Efectivamente, el 1-reductante tiene sólo ligeras diferencias sintácticas son el reductante de Medina *et al.* [2004], que básicamente se deben al lugar en el que se introducen los grados de verdad de las reglas del programa dentro del reductante (que será en la definición del agregador del reductante o en el cuerpo del 1-reductante). Además, el siguiente resultado muestra que ambos tipos de reductantes son semánticamente equivalentes[1], puesto que dan el mismo valor cuando se interpretan.

Teorema 6.1.4. *Sea $\mathcal{P}$ un programa multi-adjunto, A un átomo básico y considérese $\mathcal{R} \equiv \langle A \leftarrow @(\mathcal{B}_1, \ldots, \mathcal{B}_n)\theta; \top\rangle$ el reductante para A en $\mathcal{P}$, donde $\theta = \theta_1 \cdots \theta_n$ y cada sustitución θ_i es un unificador de A y la cabeza de una regla $\langle H_i \leftarrow_i \mathcal{B}_i; v_i\rangle$. El* 1*-reductante $\mathcal{R}' \equiv \langle A \leftarrow @_{sup}(\mathcal{D}_1, \ldots, \mathcal{D}_n); \top\rangle$ (donde $\mathcal{D}_i \equiv v_i \&_i \mathcal{B}_i\theta$, $1 \leq i \leq n$) es semánticamente equivalente al reductante $\mathcal{R}$.*

Demostración. Para obtener la equivalencia semántica entre ambas nociones de reductantes basta demostrar que

$$\mathcal{I}(@(\mathcal{B}_1, \ldots, \mathcal{B}_n)\theta) = \mathcal{I}(@_{sup}(\mathcal{D}_1, \ldots, \mathcal{D}_n))$$

Tengamos en cuenta que las reglas $C_i \leftarrow_i \mathcal{B}_i$, cuya cabeza unifica con A, se han renombrado y, además, el átomo A es básico. Por tanto, las sustituciones θ_i, tales que $A = C_i\theta_i$, no tienen variables en común ni en sus dominios ni en sus rangos. En consecuencia, $\theta = \theta_1\theta_2 \cdots \theta_n = \theta_1 \cup \theta_2 \cup \cdots \cup \theta_n$. Entonces:

[1] El equivalente procedural de este resultado se probará en la Sección 7.8 (véase el Teorema 7.8.3).

$$\begin{aligned}
\mathcal{I}(@(\mathcal{B}_1,\ldots,\mathcal{B}_n)\theta) &= \mathcal{I}(@(\mathcal{B}_1\theta,\ldots,\mathcal{B}_n\theta)) \\
&= \mathcal{I}(@(\mathcal{B}_1\theta_1,\ldots,\mathcal{B}_n\theta_n)) \\
&= \dot{@}(\mathcal{I}(\mathcal{B}_1\theta_1),\ldots,\mathcal{I}(\mathcal{B}_n\theta_n)) \\
&= sup\{v_1\dot{\&}_1\mathcal{I}(\mathcal{B}_1\theta_1),\ldots,v_n\dot{\&}_n\mathcal{I}(\mathcal{B}_n\theta_n)\} \\
&= sup\{\mathcal{I}(v_1\&_1\mathcal{B}_1\theta_1),\ldots,\mathcal{I}(v_n\&_n\mathcal{B}_n\theta_n)\} \\
&= sup\{\mathcal{I}(\mathcal{D}_1),\ldots,\mathcal{I}(\mathcal{D}_n)\} \\
&= \dot{@}_{sup}(\mathcal{I}(\mathcal{D}_1),\ldots,\mathcal{I}(\mathcal{D}_n)) \\
&= \mathcal{I}(@_{sup}(\mathcal{D}_1,\ldots,\mathcal{D}_n)),
\end{aligned}$$

lo que concluye la prueba. □

En la siguiente sección introducimos una medida de coste computacional para los programas lógicos multi-adjuntos que posteriormente aplicamos al cálculo de reductantes para justificar que el 1-reductante tiene la misma eficiencia que el reductante primitivo.

6.2. Medidas de coste computacional

El enfoque más común para analizar la eficiencia de un programa ante un objetivo dado consiste en medir su tiempo de ejecución y la memoria usada. Sin embargo, para analizar (teóricamente) la eficiencia de los programas que se obtienen por técnicas de transformación de programas, es conveniente definir métodos abstractos para estimar el coste computacional.

En el marco de la programación declarativa, es usual estimar el esfuerzo computacional necesario para ejecutar un objetivo en un programa contando el número de pasos de derivación requeridos para alcanzar sus soluciones. En el contexto de la programación lógica multi-adjunta mostramos que, aunque este método parece aceptable en la fase operacional, resulta inapropiado cuando abordamos la fase interpretativa. El problema surge cuando consideramos agregadores en el cuerpo de las reglas de programa cuyas definiciones invocan a otros agregadores. Obviamente, la evaluación de este último tipo de enlaces consume recursos computacionales en tiempo de ejecución que no son tenidos en cuenta al contar los pasos interpretativos.

Es decir, una medida ingenua no estima estos recursos en el coste final, toda vez que no se contabilizan explícitamente al dar pasos interpretativos. En esta sección, hemos resuelto el problema referido asignando pesos a los agregadores que resulten acordes con su complejidad.

Definiremos, en lo que sigue, una medida de coste (interpretativo) más refinada basada en contabilizar el número de conectivas y operadores primitivos que aparecen dentro de la definición de los agregadores que son evaluados en cada paso (interpretativo) de una derivación dada. Asimismo, en aplicación de este criterio de coste, contrastamos la eficiencia de dos nociones semánticamente equivalentes de reductantes (la original, de la Definición 6.1.1, introducida por Medina *et al.* [2004] y la versión refinada de 1-reductante introducida por primera vez en [Julián *et al.*, 2007b]) y recogida en la Definición 6.1.2.

En el futuro, nos proponemos aprovechar estos criterios de coste para abordar formalmente la eficiencia del desplegado difuso estudiado en el Capítulo 4 y de las técnicas de evaluación parcial que consideraremos en el Capítulo 7.

Un modo clásico y natural de estimar el coste computacional requerido para construir una derivación, consiste en contar el número de pasos computacionales ejecutados en la misma. Así, dada una derivación D, definimos su:

- *coste operacional*, $\mathcal{O}_c(D)$, como el número de pasos admisibles ejecutados en la derivación D.

- *coste interpretativo*, $\mathcal{I}_c(D)$, como el número de pasos interpretativos dados en la derivación D.

Así, los costes operacional e interpretativo de la derivación D_1 que se contempla en el Ejemplo 6.2.1 son $\mathcal{O}_c(D_1) = 5$ y $\mathcal{I}_c(D_1) = 4$, respectivamente. De manera intuitiva, $\mathcal{O}_c$ nos informa acerca del número de átomos explotados a lo largo de la derivación.

Análogamente, $\mathcal{I}_c$ estima el número de agregadores evaluados en una derivación. Sin embargo, esta última afirmación no es completamente cierta: $\mathcal{I}_c$ sólo tiene en cuenta aquellos agregadores que aparecen en los cuerpos de las reglas de programa, pero no aquéllos que aparecen recursivamente *anidados* en la definición de otros agregadores.

El ejemplo que se recoge a continuación pone de relieve este hecho.

Ejemplo 6.2.1. *Sea $\mathcal{P}$ el programa indicado, con retículo asociado $([0,1],\leq)$:*

$$\mathcal{R}_1 : p(X,Y) \leftarrow_{\mathtt{P}} @_1(\&_{\mathtt{L}}(q(X), r(X)), \vee_{\mathtt{G}}(s(Y), t(Y))) \quad \textit{with} \;\; 0.9$$

$$\mathcal{R}_2 : q(a) \leftarrow \quad \textit{with} \;\; 0.8$$

$$\mathcal{R}_3 : r(X) \leftarrow \quad \textit{with} \;\; 0.7$$

$$\mathcal{R}_4 : s(b) \leftarrow \quad \textit{with} \;\; 0.9$$

$$\mathcal{R}_5 : t(Y) \leftarrow \quad \textit{with} \;\; 0.6$$

para el que las funciones de verdad están definidas por:

$$\dot{@}_1(x,y) = (x+y)/2 \quad \dot{\&}_{\mathtt{L}}(x,y) = max\{0, x+y-1\}$$

$$\dot{\vee}_{\mathtt{G}}(x,y) = max\{x,y\} \quad \dot{\&}_{\mathtt{P}}(x,y) = x \cdot y$$

Como ya hemos considerado en otras ocasiones, las etiquetas `L`*,* `G` *y* `P` *son, respectivamente, las iniciales de* lógica de Łukasiewicz*,* lógica intuicionista de Gödel *y* lógica del producto*, y la función de verdad del agregador @ es la media aritmética.*

Podemos generar la siguiente derivación completa para el programa $\mathcal{P}$ y el objetivo $p(X,Y)$,

$$\begin{aligned}
\mathcal{D}_1 : [\, &\langle \underline{p(X,Y)};\ id\rangle && \rightarrow^{\mathcal{R}_1}_{AS1}\\
&\langle \&_{\mathtt{P}}(0.9, @_1(\&_{\mathtt{L}}(\underline{q(X_1)}, r(X_1)), \vee_{\mathtt{G}}(s(Y_1), t(Y_1))); \sigma_1\rangle && \rightarrow^{\mathcal{R}_2}_{AS2}\\
&\langle \&_{\mathtt{P}}(0.9,\ @_1(\ \&_{\mathtt{L}}(0.8, \underline{r(a)}), \vee_{\mathtt{G}}(s(Y_1), t(Y_1))); \sigma_2\rangle && \rightarrow^{\mathcal{R}_3}_{AS2}\\
&\langle \&_{\mathtt{P}}(0.9, @_1(\&_{\mathtt{L}}(0.8, 0.7), \vee_{\mathtt{G}}(\underline{s(Y_1)}, t(Y_1))); \sigma_3\rangle && \rightarrow^{\mathcal{R}_4}_{AS2}\\
&\langle \&_{\mathtt{P}}(0.9, @_1(\&_{\mathtt{L}}(0.8, 0.7), \vee_{\mathtt{G}}(0.9, \underline{t(b)})); \sigma_4\rangle && \rightarrow^{\mathcal{R}_5}_{AS2}\\
&\langle \&_{\mathtt{P}}(0.9, @_1(\underline{\&_{\mathtt{L}}(0.8, 0.7)}, \vee_{\mathtt{G}}(0.9, 0.6)); \sigma_5\rangle && \rightarrow_{IS}\\
&\langle \&_{\mathtt{P}}(0.9, @_1(0.5, \underline{\vee_{\mathtt{G}}(0.9, 0.6)}); \sigma_5\rangle && \rightarrow_{IS}\\
&\langle \&_{\mathtt{P}}(0.9, \underline{@_1(0.5, 0.9)}); \sigma_5\rangle && \rightarrow_{IS}\\
&\langle \underline{\&_{\mathtt{P}}(0.9, 0.7)}; \sigma_5\rangle && \rightarrow_{IS}\\
&\langle 0.63; \sigma_5\rangle\ ]
\end{aligned}$$

donde las sustituciones consideradas son

$$\sigma_1 = \{X/X_1, Y/Y_1\},$$
$$\sigma_2 = \{X/a, Y/Y_1, X_1/a\},$$
$$\sigma_3 = \{X/a, Y/Y_1, X_1/a, X_2/a\},$$
$$\sigma_4 = \{X/a, Y/b, X_1/a, X_2/a, Y_1/b\},$$
$$\sigma_5 = \{X/a, Y/b, X_1/a, X_2/a, Y_1/b, Y_2/b\}$$

Por otra parte, una versión simplificada de la regla $\mathcal{R}_1$ *es la nueva regla* $\mathcal{R}_1^*$*, en cuyo cuerpo sólo hay un símbolo agregador*

$$\mathcal{R}_1^* : p(X,Y) \leftarrow_{\mathtt{P}} @_1^*(q(X), r(X), s(Y), t(Y)) \quad \textit{with } 0.9$$

donde $\dot{@}_1^*(x,y,z,w) = (\dot{\&}_{\mathtt{L}}(x,y) + \dot{\vee}_{\mathtt{G}}(z,w))/2$*. Notemos que* $\mathcal{R}_1^*$ *tiene exactamente el mismo significado (la misma interpretación) que* $\mathcal{R}_1$*, aunque diferente sintaxis. De hecho, ambas reglas tienen la misma secuencia de átomos en la cabeza y en el cuerpo. Las diferencias surgen con respecto al conjunto de agregadores que aparecen explícitamente en sus cuerpos puesto que en* $\mathcal{R}_1^*$ *hemos trasladado* $\&_{\mathtt{L}}$ *y* $\vee_{\mathtt{G}}$ *desde el cuerpo (véase* $\mathcal{R}_1$*) a la definición de* $@_1^*$*. Ahora, usamos la regla* $\mathcal{R}_1^*$ *en lugar de* $\mathcal{R}_1$ *para generar la siguiente derivación* D_1^* *que devuelve la misma respuesta computada difusa que* D_1*:*

$$\begin{aligned}
D_1^* : [\, &\langle \underline{p(X,Y)}; id\rangle && \rightarrow^{\mathcal{R}_1^*}_{AS1}\\
&\langle \&_{\mathtt{P}}(0.9, @_1^*(\underline{q(X_1)}, r(X_1), s(Y_1), t(Y_1)); \sigma_1\rangle && \rightarrow^{\mathcal{R}_2}_{AS2}\\
&\langle \&_{\mathtt{P}}(0.9, @_1^*(0.8, \underline{r(a)}, s(Y_1), t(Y_1)); \sigma_2\rangle && \rightarrow^{\mathcal{R}_3}_{AS2}\\
&\langle \&_{\mathtt{P}}(0.9, @_1^*(0.8, 0.7, \underline{s(Y_1)}, t(Y_1)); \sigma_3\rangle && \rightarrow^{\mathcal{R}_4}_{AS2}\\
&\langle \&_{\mathtt{P}}(0.9, @_1^*(0.8, 0.7, 0.9, \underline{t(b)}); \sigma_4\rangle && \rightarrow^{\mathcal{R}_5}_{AS2}\\
&\langle \&_{\mathtt{P}}(0.9, \underline{@_1^*(0.8, 0.7, 0.9, 0.6)}; \sigma_5\rangle && \rightarrow_{IS}\\
&\langle \underline{\&_{\mathtt{P}}(0.9, 0.7)}; \sigma_5\rangle && \rightarrow_{IS}\\
&\langle 0.63; \sigma_5\rangle \,] &&
\end{aligned}$$

donde las sustituciones $\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5$ *son las anteriormente indicadas. Dado que*

hemos explotado los mismos átomos con las mismas reglas (excepto en el primer paso ejecutado con las reglas $\mathcal{R}_1$ y $\mathcal{R}_1^*$) en ambas derivaciones, se tiene

$$\mathcal{O}_c(D_1) = \mathcal{O}_c(D_1^*) = 5.$$

Sin embargo, aunque los agregadores $\&_L$ y $\vee_G$ se han evaluado en las dos derivaciones, en D_1^* tales evaluaciones no han sido contadas explícitamente como pasos interpretativos y, en consecuencia, no han sido añadidos para incrementar el coste interpretativo $\mathcal{I}_c$. Esta situación poco realista se muestra en el resultado atípico: $\mathcal{I}_c(D_1) = 4 > 2 = \mathcal{I}_c(D_1^*)$. Es importante observar que $\mathcal{R}_1^*$ no puede ser considerada una versión optimizada de $\mathcal{R}_1$: ambas reglas tiene un comportamiento computacional similar, incluso cuando la medida errónea $\mathcal{I}_c$ parece indicar lo contrario.

Para resolver este problema, nos proponemos dar una definición más fina de $\mathcal{I}_c$ en términos de las conectivas y *operadores primitivos*[2] que aparecen en la expresión de los agregadores que han sido evaluados en cada paso interpretativo de una derivación dada. Así, dado un agregador n-ario, @, cuya función de verdad está definida por $\dot{@}(x_1, \ldots, x_n) = E$ y donde las secuencias (quizás vacías) de operadores primitivos y agregadores (incluyendo conjunciones y disyunciones) que aparecen en E son respectivamente $op_1, \ldots, op_r$ y $@_1, \ldots, @_s$, el peso del agregador @ es definido recursivamente como $\mathcal{W}(@) = r + \mathcal{W}(@_1) + \cdots + \mathcal{W}(@_s)$. Ahora, nuestra noción mejorada de *coste interpretivo*, denotada por $\mathcal{I}_c^+(D)$, es la suma de los pesos de los agregadores evaluados en cada uno de los pasos interpretativos de la derivación D.

Ejemplo 6.2.2. *Observando las definiciones dadas anteriormente para los agregadores $\&_{\mathtt{P}}, \vee_{\mathtt{G}}, \&_{\mathtt{L}}, @_1$ y $@_1^*$, resulta sencillo comprobar que: $\mathcal{W}(\&_{\mathtt{P}}) = 1$, $\mathcal{W}(\&_{\mathtt{L}}) = 3$, $\mathcal{W}(\vee_{\mathtt{G}}) = 1$, $\mathcal{W}(@_1) = 2$ y $\mathcal{W}(@_1^*) = 2 + \mathcal{W}(\&_{\mathtt{L}}) + \mathcal{W}(\vee_{\mathtt{G}}) = 2 + 3 + 1 = 6$. En consecuencia, tenemos que*

$$\mathcal{I}_c^+(D_1) = \mathcal{W}(\&_{\mathtt{L}}) + \mathcal{W}(\vee_{\mathtt{G}}) + \mathcal{W}(@_1) + \mathcal{W}(\&_{\mathtt{P}}) = 3 + 1 + 2 + 1 = 7,$$

$$\mathcal{I}_c^+(D_1^*) = \mathcal{W}(@_1^*) + \mathcal{W}(\&_{\mathtt{P}}) = 6 + 1 = 7.$$

Coinciden, ahora, tanto el coste operacional como el interpretativo de las derivaciones D_1 y D_1^*, lo cual es completamente natural y realista si tenemos en cuenta que las reglas $\mathcal{R}_1$ y $\mathcal{R}_1^*$ tienen la misma semántica y, por consiguiente, deben tener el mismo comportamiento computacional.

[2] Es decir, operadores aritméticos tales como $+, -, *, /, max, min, raiz$, etc.

El ejemplo previo nos muestra que el modo en que los agregadores se han introducido en el cuerpo o en la definición de otros agregadores de una regla de programa, puede reflejar preferencias sintácticas solamente, sin repercusiones negativas sobre el coste computacional, como revela nuestra definición mejorada de coste interpretativo.

6.3. Completitud

En esta sección, dado un programa $\mathcal{P}$, para el que su semántica operacional no garantiza resultados de completitud, obtendremos –haciendo uso de los reductantes– otros programas equivalentes (en los términos que fijaremos oportunamente) para los que existan garantías de completitud y que denominaremos compleciones de $\mathcal{P}$. Con más precisión, llamaremos compleción de $\mathcal{P}$ a un programa, posiblemente infinito, cuyas reglas son todas reductantes. La definición formal que establecemos para este concepto es la siguiente.

Definición 6.3.1. *Dado un programa lógico multi-adjunto $\mathcal{P}$, decimos que una*[3] *compleción de $\mathcal{P}$ es otro programa, posiblemente infinito, cuyas reglas son reductantes de cada uno de los átomos básicos de la Base de Herbrand de $\mathcal{P}$.*

De manera más precisa, concretamos la noción que representa cada una de las compleciones posibles.

Definición 6.3.2. *Dado un programa lógico multi-adjunto $\mathcal{P}$, decimos que su compleción-1, que denotamos por $\mathcal{P}^1$, es otro programa posiblemente infinito construido a partir de $\mathcal{P}$ tal y como se indica a continuación. Para cada átomo básico A de la Base de Herbrand de $\mathcal{P}$ se selecciona el conjunto $S_A = \{\langle H_i \leftarrow_i \mathcal{B}_i; v_i\rangle \in \mathcal{P} : \exists \theta_i / A = H_i\theta_i\}$ de reglas de $\mathcal{P}$ cuyas cabezas unifican con A.*

i) Si S_A es vacío, entonces no hay ninguna regla que define a A en $\mathcal{P}^1$.

ii) En otro caso, A está definido en $\mathcal{P}^1$mediante una única regla (llamada 1-reductante*) definida como: $\langle A \leftarrow @_{sup}((v_1 \&_1 \mathcal{B}_1)\theta_1, \ldots, (v_n \&_n \mathcal{B}_n)\theta_n); \top\rangle$.*

Esta definición admite una variante que denotamos por compleción-0, $\mathcal{P}^0$, cuando en el último caso de la definición, en vez de utilizar la noción de 1-reductante, se considera la original de reductante de Medina *et al.* [2004]. En el próximo capítulo,

[3] En el próximo capítulo, veremos que podrán existir distintas compleciones para un mismo programa, dado que será posible elegir distintos reductantes para cada átomo básico.

tendrá sentido una nueva variante que podemos llamar compleción-k, $\mathcal{P}^k$, cuando usemos en lugar del 1-reductante el concepto extendido de PE-reductante (obtenido a partir de un árbol de desplegado τ de nivel k, como recoge la posterior Definición 7.3.5). Para un mismo programa $\mathcal{P}$, sólo existe una única compleción-0 y compleción-1, pero en general, podrán existir diferentes compleciones-k.

El siguiente ejemplo compara las respuestas computadas de un programa $\mathcal{P}$ y de su compleción $\mathcal{P}^1$.

Ejemplo 6.3.3. *Sea $\mathcal{P}$ el programa lógico multi-adjunto que tiene las siguientes reglas (hechos) y cuyo retículo asociado es $([0,1], \leq)$.*

$$\mathcal{R}_1 : \quad \langle p(X) \leftarrow \; ; \quad 0.5 \rangle$$

$$\mathcal{R}_2 : \quad \langle p(a) \leftarrow \; ; \quad 1 \rangle$$

Como la base de Herbrand es $\mathcal{B} = \{p(a)\}$, la compleción-1 es el programa $\mathcal{P}^1$ que tiene por única regla el 1-reductante para el átomo $p(a)$ en $\mathcal{P}$

$$\mathcal{R}^1 : \quad \langle p(a) \leftarrow \quad @_{sup}(0.5 \& 1, 1 \& 1); 1 \rangle$$

Análogamente, la compleción-0 es el programa $\mathcal{P}^0$ que tiene por única regla el reductante para el átomo $p(a)$ en $\mathcal{P}$, $\mathcal{R}^0$, en el que se toma $\dot{@}(x,y) = sup\{0.5 \& x, 1 \& y\}$.

$$\mathcal{R}^0 : \quad \langle p(a) \leftarrow \quad @(1,1); 1 \rangle$$

Con la definición que hemos dado de compleción-1 (análogamente de compleción-0), hay que asumir que no se mostrarán ciertas respuestas computadas del programa $\mathcal{P}$ para objetivos básicos de la base de Herbrand. Es lo que se muestra a continuación, y este hecho no supone la pérdida de corrección: sólo se pierden las respuestas "peores", una vez que es consustancial con la noción de reductante alcanzar a través de la misma la mejor respuesta computada. En efecto, para el objetivo $p(a)$ (obsérvese que $p(a) \in \mathcal{B}$) obtenemos las siguientes respuestas computadas en $\mathcal{P}$:

$$\langle p(a); id \rangle \rightarrow^{\mathcal{R}_1}_{AS2} \langle 0.5; id \rangle$$

$$\langle p(a); id \rangle \rightarrow^{\mathcal{R}_2}_{AS2} \langle 1; id \rangle$$

En cambio, se obtiene la única respuesta computada en $\mathcal{P}^1$:

$$\langle p(a); id \rangle \rightarrow^{\mathcal{R}^1}_{AS1} \langle @_{sup}(0.5, 1); id \rangle \rightarrow_{IS} \langle 1; id \rangle$$

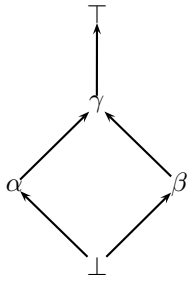
Obtenemos resultados análogos en el ejemplo posterior en el que consideramos ahora un programa cuyo retículo asociado no está totalmente ordenado. Calcularemos la compleción-1 del citado programa y discutiremos la corrección y completitud, es decir, compararemos las respuestas correctas y computadas que se obtienen en ambos programas para objetivos convenientes.

Ejemplo 6.3.4. *Sea $\mathcal{P}$ el programa lógico multi-adjunto, con retículo asociado dado por el grafo adjunto, que tiene las reglas*

$$\mathcal{R}_1: \quad \langle p(a) \leftarrow_{\mathtt{G}} ; \quad \alpha \rangle$$

$$\mathcal{R}_2: \quad \langle p(b) \leftarrow_{\mathtt{G}} ; \quad \beta \rangle$$

$$\mathcal{R}_3: \quad \langle p(X_1) \leftarrow_{\mathtt{G}} ; \quad \beta \rangle$$



Como la base de Herbrand de $\mathcal{P}$ es el conjunto $\mathcal{B} = \{p(a), p(b)\}$, la compleción-1 es el programa $\mathcal{P}^1$ que tiene por reglas el 1-reductante para el átomo $p(a)$ en $\mathcal{P}$ y el 1-reductante para el átomo $p(b)$ en $\mathcal{P}$, es decir:

$$\mathcal{R}^1_{13}: \quad \langle p(a) \leftarrow_{\mathtt{G}} @_{sup}(\alpha, \beta); \quad \top \rangle$$

$$\mathcal{R}^1_{23}: \quad \langle p(b) \leftarrow_{\mathtt{G}} @_{sup}(\beta, \beta); \quad \top \rangle$$

Para este programa $\mathcal{P}$ y su compleción-1 $\mathcal{P}^1$ nos proponemos obtener el modelo mínimo difuso, así como las respuestas correctas y computadas para diferentes objetivos, que nos permitan observar la corrección y completitud de tales programas.

1) *Modelo mínimo de Herbrand difuso de $\mathcal{P}$:*

   Una interpretación $\mathcal{I}_j$ es un modelo de $\mathcal{P}$ si, y sólo si, satisface todas las reglas de $\mathcal{P}$ (veáse el Capítulo 3). Además, en el Teorema 3.3.2 hemos caracterizado el modelo mínimo difuso $\mathcal{I}$ del programa $\mathcal{P}$ como $\mathcal{I} = inf\{\mathcal{I}_j : \mathcal{I}_j$ es un modelo de $\mathcal{P}\}$, es decir, lo hemos tomado como el menor modelo del programa $\mathcal{P}$.

Entonces, para el programa $\mathcal{P}$ anterior, un modelo $\mathcal{I}$ es tal que

$$\mathcal{I} \text{ satisface la regla } \mathcal{R}_1 \iff \alpha \leq \mathcal{I}(p(a)) \quad (1)$$

$$\mathcal{I} \text{ satisface la regla } \mathcal{R}_2 \iff \beta \leq \mathcal{I}(p(b)) \quad (2)$$

$$\mathcal{I} \text{ satisface la regla } \mathcal{R}_3 \iff \beta \leq \mathcal{I}(p(X_1)) \quad (3)$$

donde, por definición,

$$\mathcal{I}(p(X_1)) = inf\{\mathcal{I}(p(X_1)\theta) : p(X_1)\theta \text{ es una instancia básica de } p(X_1)\} = [4]$$

$inf\{\mathcal{I}(p(a), \mathcal{I}(p(b)\}$ (4).

De las condiciones (1), (2), (3), (4) resulta necesariamente que el modelo mínimo $\mathcal{I}$ de $\mathcal{P}$ satisface que

$$\mathcal{I}(p(a)) = \gamma \qquad \mathcal{I}(p(b)) = \beta \qquad \mathcal{I}(p(X_1)) = \beta$$

y, en consecuencia, el modelo mínimo difuso del programa $\mathcal{P}$ es la aplicación $\mathcal{I} : \mathcal{B} \longrightarrow L$ definida por $\mathcal{I}(p(a)) = \gamma$, $\mathcal{I}(p(b)) = \beta$.

2) *Modelo mínimo de Herbrand difuso de $\mathcal{P}^1$:*

Razonando de manera análoga al apartado anterior, un modelo $\mathcal{I}$ del programa $\mathcal{P}^1$ es tal que

$$\mathcal{I} \text{ satisface la regla } \mathcal{R}^1_{13} \iff \top \leq \mathcal{I}(p(a) \leftarrow_{\mathtt{G}} @_{sup}(\alpha, \beta)) \quad (5)$$

$$\mathcal{I} \text{ satisface la regla } \mathcal{R}^1_{23} \iff \top \leq \mathcal{I}(p(b) \leftarrow_{\mathtt{G}} @_{sup}(\beta, \beta)) \quad (6)$$

donde $\mathcal{I}(p(a)) \leftarrow_{\mathtt{G}} @_{sup}(\alpha, \beta)) = \dot{\leftarrow}_{\mathtt{G}}(\mathcal{I}(p(a)), \mathcal{I}(@_{sup}(\alpha, \beta)) = \dot{\leftarrow}_{\mathtt{G}}(\mathcal{I}(p(a)), \gamma)$.

Como la función de verdad $\dot{\leftarrow}_{\mathtt{G}}(y, x) = \begin{cases} \top, & si\ x \leq y \\ y, & en\ otro\ caso \end{cases}$, *para que se verifique la condición (5) es preciso que $\gamma \leq \mathcal{I}(p(a))$. Además, razonando de forma similar, que $\mathcal{I}$ satisface la regla $\mathcal{R}^1_{23}$ obliga a que $\beta \leq \mathcal{I}(p(b))$.*

Por tanto, el modelo mínimo difuso del programa $\mathcal{P}^1$ es la aplicación $\mathcal{I} : \mathcal{B} \longrightarrow L$ definida por $\mathcal{I}(p(a)) = \gamma, \mathcal{I}(p(b)) = \beta$.

[4] Basta tener en cuenta que $\mathcal{I}$, al igual que toda interpretación, está determinada por sus valores sobre la base de Herbrand de $\mathcal{P}$.

En consecuencia, hemos visto que los programas $\mathcal{P}$ y $\mathcal{P}^1$ tienen el mismo modelo mínimo difuso[5].

3) *Respuestas correctas para el átomo $p(a)$ en $\mathcal{P}, \mathcal{P}^1$:*

Por definición, el par $\langle \lambda; \theta \rangle$ es una respuesta correcta para $\mathcal{P}$ y $p(a)$ si, y sólo si, $\lambda \leq \mathcal{I}_j(p(a)\theta)$, para todo modelo $\mathcal{I}_j$ de $\mathcal{P}$. Además, en el Teorema 3.4.5, hemos caracterizado las respuestas correctas a partir del modelo mínimo difuso de $\mathcal{P}$, de forma que $\langle \lambda; \theta \rangle$ es una respuesta correcta para $\mathcal{P}$ y $p(a)$ si, y sólo si, $\lambda \leq \mathcal{I}(p(a))$, donde $\mathcal{I}$ es el modelo mínimo de Herbrand difuso de $\mathcal{P}$. En consecuencia, una vez que hemos justificado que el modelo mínimo de $\mathcal{P}$ satisface que $\mathcal{I}(p(a)) = \gamma$, de la definición se obtienen las siguientes respuestas correctas para $p(a)$ en $\mathcal{P}$:

$$\langle \gamma; id \rangle \qquad \langle \alpha; id \rangle \qquad \langle \beta; id \rangle \qquad \langle \bot; id \rangle$$

Si ahora buscamos las respuestas correctas de $p(a)$ en $\mathcal{P}^1$, basta tener en cuenta que los programas $\mathcal{P}, \mathcal{P}^1$ tienen el mismo modelo mínimo difuso, tal como hemos comprobado en los apartados $1), 2)$ anteriores. Por tanto, se obtienen las mismas respuestas correctas para el objetivo $p(a)$ en $\mathcal{P}^1$.

4) *Respuestas correctas para el átomo $p(b)$ en $\mathcal{P}, \mathcal{P}^1$:*

Con una discusión análoga, usando ahora que $\mathcal{I}(p(b)) = \beta$, se tienen las respuestas correctas para $p(b)$ en $\mathcal{P}$ y en $\mathcal{P}^1$

$$\langle \beta; id \rangle \qquad \langle \bot; id \rangle$$

5) *Respuestas correctas para el átomo $p(X)$ en $\mathcal{P}, \mathcal{P}^1$:*

Dado que el modelo mínimo difuso $\mathcal{I}$ de los programas $\mathcal{P}, \mathcal{P}^1$ satisface que $\mathcal{I}(p(x)) = \beta$, se obtienen las respuestas correctas que siguen para $p(x)$ en $\mathcal{P}$ y en $\mathcal{P}^1$, en donde tenemos en cuenta que X_1 es una variable.

$$\langle \gamma; \{X/a\} \rangle \qquad \langle \beta; \{X/b\} \rangle \qquad \langle \beta; \{X/X_1\} \rangle$$

$$\langle \alpha; \{X/a\} \rangle \qquad \langle \bot; \{X/b\} \rangle \qquad \langle \bot; \{X/X_1\} \rangle$$

$$\langle \bot; \{X/a\} \rangle$$

[5] La comprobación de que $\mathcal{P}$ y $\mathcal{P}^0$ tienen el mismo modelo mínimo de Herbrand difuso requiere el cálculo del modelo mínimo de $\mathcal{P}^0$ que se obtiene de manera similar.

6) *Respuestas computadas para el átomo* $p(X)$ *en* $\mathcal{P}$*:*

Al explotar el objetivo con variables $p(X)$ *obtenemos las derivaciones en* $\mathcal{P}$

$$\langle p(X); id\rangle \rightarrow^{\mathcal{R}_1}_{AS2} \langle \alpha; \{X/a\}\rangle$$

$$\langle p(X); id\rangle \rightarrow^{\mathcal{R}_2}_{AS2} \langle \beta; \{X/b\}\rangle$$

$$\langle p(X); id\rangle \rightarrow^{\mathcal{R}_3}_{AS2} \langle \beta; \{X/X_1\}\rangle$$

cuyas respuestas computadas difusas son, respectivamente,

$$\langle \alpha; \{X/a\}\rangle \qquad \langle \beta; \{X/b\}\rangle \qquad \langle \beta; \{X/X_1\}\rangle$$

Véase que no se computa la respuesta $\langle \gamma; \{X/a\}\rangle$ *(mostramos en el apartado* 5*) que efectivamente es respuesta correcta para el objetivo* $p(X)$*). Entonces, el programa* $\mathcal{P}$ *no es completo y, en consecuencia, tiene interés el uso de reductantes para garantizar su completitud (por cierto, los reductantes sólo se han usado en la literatura PLMA para objetivos básicos, aunque en [Morcillo y Moreno, 2009c] puede verse una aproximación al caso de objetivos con variables). Por otra parte, hay corrección ya que las respuestas anteriores son todas correctas.*

7) *Respuestas computadas para el átomo* $p(X)$ *en* $\mathcal{P}^1$*:*

Si, ahora, explotamos el objetivo $p(X)$ *respecto del programa* $\mathcal{P}$*, obtenemos las derivaciones*

$$\langle p(X); id\rangle \rightarrow^{\mathcal{R}^1_{13}}_{AS1} \langle @_{sup}(\alpha, \beta); \{X/a\}\rangle \rightarrow_{IS} \langle \gamma; \{X/a\}\rangle$$

$$\langle p(X); id\rangle \rightarrow^{\mathcal{R}^1_{23}}_{AS1} \langle @_{sup}(\beta, \beta); \{X/b\}\rangle \rightarrow_{IS} \langle \beta; \{X/b\}\rangle$$

que originan las respuestas computadas difusas

$$\langle \gamma; \{X/a\}\rangle \qquad \langle \beta; \{X/b\}\rangle$$

Observamos que, para el objetivo considerado, se obtiene la respuesta computada difusa $\langle \alpha; \{X/a\}\rangle$ *en* $\mathcal{P}$ *que falta en* $\mathcal{P}^1$*; ahora bien, en este este programa se obtiene una más general:* $\langle \gamma; \{X/a\}\rangle$*.*

Sin embargo, lo más significativo es que las respuestas correctas $\langle \beta; \{X/X_1\}\rangle$*,* $\langle \bot; \{X/X_1\}\rangle$ *para el objetivo* $p(X)$ *y el programa* $\mathcal{P}^1$*, donde la componente sustitución es del tipo* $\{X/X_1\}$ *y* X_1 *es una variable, no se computan en dicho*

programa. Este hecho tiene como consecuencia que la compleción-1 considerada no es completa si permitimos explotar átomos con variables. Por tanto, resultará imprescindible limitarnos a átomos básicos (para los únicos átomos que disponemos de la noción de reductante).

Como venimos diciendo, en programación lógica multi-adjunta es bien conocido el problema de incompletitud que se presenta al hacer uso directo de la semántica procedural estándar definida en la Sección 3.1 ante programas basados en retículos no totalmente ordenados. Existen al menos dos formas prácticamente antagónicas de hacer frente al problema:

- Modificar el programa original y mantener la semántica procedural. Esta vía se basa en el uso de reductantes (y sus variantes) donde el programa original $\mathcal{P}$ es sustituido por $\mathcal{P}^0$, $\mathcal{P}^1$o $\mathcal{P}^k$. Las ventajas se obtienen a nivel teórico aún cuando en la práctica no sea factible trabajar con programas infinitos.
- Modificar la semántica procedural sin cambiar el programa. En esta versión es necesario enriquecer la semántica procedural para simular los efectos del uso de reductantes. Su principal ventaja es que además de ser practicable, heredará todos los resultados teóricos demostrados (cómodamente) en la versión anterior, una vez establecidos los correspondientes teoremas de equivalencia entre ambas aproximaciones.

Si optamos por esta segunda vía, la definición que sigue permite formalizar la nueva semántica procedural y el Teorema 6.3.6 garantiza la equivalencia entre ambas alternativas.

Definición 6.3.5 (Paso Admisible Extendido)**.** *Sea $\mathcal{Q}$ un objetivo básico y $\mathcal{P}$ un programa multi-adjunto. Una* computación admisible extendida *se formaliza como un sistema de transición de estados (donde cada estado es un posible objetivo), cuya relación de transición $\rightarrow_{EAS}$ es la menor relación que satisface las siguientes* reglas extendidas admisibles *donde consideramos en todo caso que A es el átomo seleccionado en $\mathcal{Q}$ y que $S_A = \{\langle H_i \leftarrow_i \mathcal{B}_i; v_i\rangle \in \mathcal{P} : \exists \theta_i / A = H_i\theta_i\}$ es el conjunto de reglas de $\mathcal{P}$ cuyas cabezas unifican con A:*

1) $\mathcal{Q}[A] \rightarrow_{EAS1} (\mathcal{Q}[A/@_{sup}((v_1 \&_1 \mathcal{B}_1)\theta_1, \ldots, (v_n \&_n \mathcal{B}_n)\theta_n)])$ *si el cardinal de S_A es mayor o igual que 1.*

2) $Q[A] \rightarrow_{EAS3} (\mathcal{Q}[A/\bot])$ *si S_A es vacío.*

Teorema 6.3.6 (Equivalencia entre vía 1 y vía 2)**.** *Sea $\mathcal{P}$ un programa multi-adjunto y A un átomo básico. Entonces, existe una única derivación basada en pasos admisibles extendidos $D : [A \rightarrow^{+}_{EAS/IS} r]$ sobre el programa original $\mathcal{P}$ si, y sólo si, existe una única derivación basada en pasos admisibles ordinarios $D_1 : [A \rightarrow^{+}_{AS/IS} r]$ sobre el programa compleción-1 $\mathcal{P}^1$.*

En lo que sigue, nos centraremos en la primera vía para mostrar que las distintas compleciones de un programa $\mathcal{P}$ dan las mismas respuestas computadas difusas. Nos ocuparemos, asimismo, de estudiar la eficiencia con que se obtienen estas respuestas.

El resultado posterior garantiza que la compleción-0 y la compleción-1 aportan las mismas respuestas computadas para un átomo básico y, además, con la misma eficiencia.

Como consecuencia del mismo, resulta la equivalencia procedural entre el reductante debido a Medina *et al.* [2004] y el 1-reductante: ambos aportan las mismas respuestas computadas para un átomo básico en un programa $\mathcal{P}$ y, además, con la misma eficiencia.

Teorema 6.3.7. *Sea $\mathcal{P}$ un programa multi-adjunto y A un átomo básico. Entonces existe una única derivación $D_0 : [\langle A; id\rangle \rightarrow^{+}_{AS/IS} \langle r; \sigma\rangle]$ en $\mathcal{P}^0$ si, y sólo si, existe una única derivación $D_1 : [\langle A; id\rangle \rightarrow^{+}_{AS/IS} \langle r; \sigma\rangle]$ en $\mathcal{P}^1$ y, además, se verifica que $\mathcal{O}_c(D_0) = \mathcal{O}_c(D_1)$ y $\mathcal{I}^{+}_c(D_0) = \mathcal{I}^{+}_c(D_1)$.*

Demostración. Abordemos las dos implicaciones que comportan la equivalencia enunciada.

"$\Longrightarrow$"

Sea $[D_0 : \langle A; id\rangle \rightarrow^{+}_{AS/IS} \langle r; \theta\rangle]$ en $\mathcal{P}^0$ una derivación de longitud n. Veamos, por inducción en n, que existe una derivación D_1 en $\mathcal{P}^1$verificando el resultado.

Si $n = 1$, la derivación D_0 comprende un único paso de derivación que es, necesariamente de tipo $AS3$.

Si el paso de derivación es de tipo $AS3$, debido a que el conjunto S_A es vacío, tenemos

$$D_0 : [\langle A; id\rangle \rightarrow_{AS3} \langle \bot; id\rangle]$$

y, por definición de compleción-1, no hay ninguna regla que defina A en $\mathcal{P}^1$; en consecuencia, podemos construir la derivación D_1 en $\mathcal{P}^1$

$$D_1 : [\langle A; id\rangle \rightarrow_{AS3} \langle \bot; id\rangle]$$

Supongamos, ahora, que el resultado se satisface para derivaciones de $\mathcal{P}^0$ de longitud $n = k$ y demostremos que también se cumple para derivaciones de $\mathcal{P}^0$ de longitud $n = k + 1$. Supongamos, por tanto, que se cumple el apartado ii) de la Definición 6.3.2, esto es, A está definido en $\mathcal{P}^0$ por una regla que es el reductante $\mathcal{R}^0 : \langle A \leftarrow @(\mathcal{B}_1\theta_1, \ldots, \mathcal{B}_n\theta_n); \top\rangle \in \mathcal{P}^0$. Entonces, la derivación D_0 tiene la forma siguiente,

$$D_0 : [\langle A; id\rangle \rightarrow^{\mathcal{R}^0}_{AS1} \langle @(\mathcal{B}_1\theta_1, \ldots, \mathcal{B}_n\theta_n); \top\rangle \rightarrow^{*}_{AS/IS} \langle r; \sigma\rangle]$$

es decir, el primer paso de D_0 ha sido ejecutado con el reductante debido a Medina *et al.* [2004] $\mathcal{R}^0 \in \mathcal{P}^0$ por ser la regla que define A en $\mathcal{P}^0$. Supongamos que en $\mathcal{P}^0$ existen las siguientes derivaciones

$$D^0_1 : [\langle \mathcal{B}_1\theta_1; id\rangle \rightarrow^{m_1}_{AS} \langle \mathcal{E}_1; \sigma_1\rangle \rightarrow^{m'_1}_{IS} \langle b_1; \sigma_1\rangle]$$

$$\vdots$$

$$D^0_n : [\langle \mathcal{B}_n\theta_n; id\rangle \rightarrow^{m_n}_{AS} \langle \mathcal{E}_n; \sigma_n\rangle \rightarrow^{m'_n}_{IS} \langle b_n; \sigma_n\rangle]$$

donde $\mathcal{E}_1, \ldots, \mathcal{E}_n$ son expresiones que no contienen átomos y $b_1, \ldots, b_n \in L$.

Por tanto, la derivación D_0 comprende los siguientes pasos de derivación:

$$\begin{aligned}
&[\langle A; id\rangle && \rightarrow^{\mathcal{R}^0}_{AS}\\
&\langle @(\mathcal{B}_1\theta_1, \ldots, \mathcal{B}_n\theta_n); id\rangle && \rightarrow^{m_1}_{AS}\\
&\langle @(\mathcal{E}_1, \ldots, \mathcal{B}_n\theta_n); \sigma_1\rangle && \rightarrow^{m_n}_{AS}\\
&\langle @(\mathcal{E}_1, \ldots, \mathcal{E}_n); \sigma_1 \cdots \sigma_n\rangle && \rightarrow^{m'_1}_{IS}\\
&\langle @(b_1, \ldots, \mathcal{E}_n); \sigma_1 \cdots \sigma_n\rangle && \rightarrow^{m'_n}_{IS}\\
&\langle @(b_1, \ldots, b_n); \sigma_1 \cdots \sigma_n\rangle && \rightarrow_{IS}\\
&\langle r; \sigma\rangle]
\end{aligned}$$

donde, si denotamos $b'_i = \dot{\&}_i(v_i, b_i)$ entonces $r = sup\{b'_1, \ldots, b'_n\}, i = 1, \ldots, n$ y $\sigma = \sigma_1 \circ \cdots \circ \sigma_n$.

Entonces, en el programa $\mathcal{P}^1$ la única regla que define el átomo A es el 1-reductante $\mathcal{R}^1 : \langle A \leftarrow @_{sup}((v_1 \&_1 \mathcal{B}_1)\theta_1, \ldots, (v_n \&_n \mathcal{B}_n)\theta_n); \top\rangle$ y, en consecuencia, podemos generar la siguiente derivación D_1 en $\mathcal{P}^1$:

$$
\begin{array}{ll}
[\langle A; id\rangle & \rightarrow_{AS}^{\mathcal{R}^1} \\
\langle @_{sup}(v_1 \&_1 \mathcal{B}_1\theta_1, \ldots, v_n \&_n \mathcal{B}_n\theta_n); id\rangle & \rightarrow_{AS}^{m_1} \\
\langle @_{sup}(v_1 \&_1 \mathcal{E}_1, \ldots, v_n \&_n \mathcal{B}_n\theta_n); \sigma_1\rangle & \rightarrow_{AS}^{m_2+\cdots+m_n} \\
\langle @_{sup}(v_1 \&_1 \mathcal{E}_1, \ldots, v_n \&_n \mathcal{E}_n); \sigma\rangle & \rightarrow_{IS}^{m'_1} \\
\langle @_{sup}(\&_1(v_1, b_1), \ldots, v_n \&_n \mathcal{E}_n); \sigma\rangle & \rightarrow_{IS}^{m'_2+\cdots+m'_n} \\
\langle @_{sup}(\&_1(v_1, b_1), \ldots, \&_n(v_n, b_n)); \sigma\rangle & \rightarrow_{IS} \\
\langle @_{sup}(b'_1, \ldots, \&_n(v_n, b_n)); \sigma\rangle & \rightarrow_{IS}^{n-1} \\
\langle @_{sup}(b'_1, \ldots, b'_n); \sigma\rangle & \rightarrow_{IS} \\
\langle r; \sigma\rangle] &
\end{array}
$$

donde, si llamamos $b'_i = \dot{\&}_i(v_i, b_i)$ entonces $r = sup\{b'_1, \ldots, b'_n\}, i = 1, \ldots, n$ y $\sigma = \sigma_1 \circ \cdots \circ \sigma_n$.
Obsérvese que ambas derivaciones conducen a la misma respuesta computada difusa, como deseábamos. Además, veamos que sus costes operacional/interpretativo son el mismo.

$$\mathcal{O}_c(D_0) = 1 + m_1 + \cdots + m_n = \mathcal{O}_c(D_1)$$

Además, si denotamos por w el coste interpretativo asociado a los primeros $m'_1 + \cdots + m'_n$ pasos interpretativos de D_0, resulta que $\mathcal{I}_c^+(D_0) = w + \mathcal{W}(@) = w + 1 + \mathcal{W}(\&_1) + \cdots + \mathcal{W}(\&_n)$.
Y para la derivación D_1, $\mathcal{I}_c^+(D_1) = w + \mathcal{W}(\&_1) + \cdots + \mathcal{W}(\&_n) + \mathcal{W}(@_{sup}) = w + \mathcal{W}(\&_1) + \cdots + \mathcal{W}(\&_n) + 1$, es decir,

$$\mathcal{I}_c^+(D_0) = \mathcal{I}_c^+(D_1)$$

con lo que termina la prueba.

"$\Longleftarrow$"

Este recíproco es análogo al directo ya probado anteriormente.

En cuanto a la unicidad, observemos que los programas $\mathcal{P}^0$, $\mathcal{P}^1$, satisfacen que para cada átomo A de la base de Herbrand existe, a lo sumo, una regla cuya cabeza unifica

(y coincide) con A: el reductante correspondiente para el átomo A. Por tanto, si queremos garantizar la unicidad en ambas derivaciones[6] es preciso que cada átomo del objetivo y de los subobjetivos de las derivaciones anteriores D_0, D_1, sólo pueda ser explotado usando, a lo sumo, una única regla de estos programas.

□

6.4. Reductantes y medidas de coste

En la Definición 6.1.2 hemos introducido el concepto de 1-reductante, cuya expresión sintáctica presenta una ligera variación del reductante de Medina *et al.* [2004] pero con la ventaja de que podremos obtenerlo y generalizarlo, en el próximo capítulo, usando técnicas de evaluación parcial, y en el Teorema 6.1.4 hemos probado que es semánticamente equivalente al reductante de Medina *et al.* [2004].

Antes de abordar la generalización mencionada, aplicamos las medidas de coste introducidas, para comparar la eficiencia de las dos nociones de reductantes. Partimos con un ejemplo en el que mostramos la idoneidad de los criterios de coste que hemos formulado para los programas multi-adjuntos y que están recogidos en la Sección 6.2.

Ejemplo 6.4.1. *Sea $\mathcal{P}$ el programa lógico multi-adjunto que sigue con retículo asociado el retículo $([0,1], \leq)$, donde "$\leq$" es el orden usual.*

$$\begin{aligned}
&\mathcal{R}_1 : \langle p(a) \leftarrow_{\mathtt{L}} q(a,b); && 0.7\rangle\\
&\mathcal{R}_2 : \langle p(a) \leftarrow ; && 0.5\rangle\\
&\mathcal{R}_3 : \langle p(X) \leftarrow_{\mathtt{G}} q(X,b); && 0.4\rangle\\
&\mathcal{R}_4 : \langle q(a,b) \leftarrow ; && 0.9\rangle
\end{aligned}$$

El reductante para $\mathcal{P}$ y $p(a)$ es:

$$\mathcal{R} \equiv \langle p(a) \leftarrow @(q(a,b), 1, q(X_1,b));\ 1\rangle$$

siendo la función de verdad de @ la dada por

$$\dot{@}(x,y,z) = sup\{\dot{\&}_{\mathtt{L}}(0.7,x), \dot{\&}(0.5,y), \dot{\&}_{\mathtt{G}}(0.4,z)\}$$

[6] Téngase en cuenta que el enunciado afirma que D_0 es única si, y sólo si, D_1 es única.

Por otra parte, el 1*-reductante para el programa* $\mathcal{P}$ *y el átomo* $p(a)$ *es:*

$$\mathcal{R}' \equiv \langle p(a) \leftarrow @_{sup}(0.7 \&_{\text{L}} q(a,b), 0.5 \& 1, 0.4 \&_{\text{G}} q(X_1,b)); 1\rangle,$$

donde la función de verdad de $@_{sup}$ *está definida por:* $\dot{@}_{sup}(x,y,z) = sup\{x,y,z\}$. *Con el reductante* $\mathcal{R}$ *podemos construir la derivación* D*:*

$$\begin{array}{ll}
\langle p(a); id\rangle & \rightarrow^{\mathcal{R}}_{AS} \\
\langle @(q(a,b), 1, q(X_1,b)); id\rangle & \rightarrow^{\mathcal{R}_4}_{AS} \\
\langle @(0.9, 1, q(X_1,b)); id\rangle & \rightarrow^{\mathcal{R}_4}_{AS} \\
\langle @(0.9, 1, 0.9); \{X_1/a\}\rangle & \rightarrow_{IS} \\
\langle sup\{0.7 \&_{\text{L}} 0.9, 0.5 \& 1, 0.4 \&_{\text{G}} 0.9\}; \{X_1/a\}\rangle & \rightarrow_{IS} \\
\langle sup\{0.6, 0.5 \& 1, 0.4 \&_{\text{G}} 0.9\}; \{X_1/a\}\rangle & \rightarrow_{IS} \\
\langle sup\{0.6, 0.5, 0.4 \&_{\text{G}} 0.9\}; \{X_1/a\}\rangle & \rightarrow_{IS} \\
\langle sup\{0.6, 0.5, 0.4\}; \{X_1/a\}\rangle & \rightarrow_{IS} \\
\langle 0.6; \{X_1/a\}\rangle &
\end{array}$$

Por otra parte, si usamos el reductante $\mathcal{R}'$*, podemos generar la derivación* D'*:*

$$\begin{array}{ll}
\langle p(a); id\rangle & \rightarrow^{\mathcal{R}}_{AS} \\
\langle @_{sup}(0.7 \&_{\text{L}} q(a,b), 0.5 \& 1, 0.4 \&_{\text{G}} q(X_1,b)); id\rangle & \rightarrow^{\mathcal{R}_4}_{AS} \\
\langle @(0.7 \&_{\text{L}} 0.9, 0.5 \& 1, 0.4 \&_{\text{G}}\ q(X_1,b)); id\rangle & \rightarrow^{\mathcal{R}_4}_{AS} \\
\langle @_{sup}(0.7 \&_{\text{L}} 0.9, 0.5 \& 1, 0.4 \&_{\text{G}} 0.9); \{X_1/a\}\rangle & \rightarrow_{IS} \\
\langle @_{sup}(0.6, 0.5 \& 1, 0.4 \&_{\text{G}} 0.9); \{X_1/a\}\rangle & \rightarrow_{IS} \\
\langle @_{sup}(0.6, 0.5, 0.4 \&_{\text{G}} 0.9); \{X_1/a\}\rangle & \rightarrow_{IS} \\
\langle @_{sup}(0.6, 0.5, 0.4); \{X_1/a\}\rangle & \rightarrow_{IS} \\
\langle 0.6; \{X_1/a\}\rangle &
\end{array}$$

Ambas derivaciones conducen a la misma respuesta computada difusa y vamos a comprobar que los costes operacional e interpretativo coinciden. En efecto:

$$\mathcal{O}_c(D) = 3 = \mathcal{O}_c(D').$$

Por otra parte

$$\mathcal{I}_c^+(D) = \mathcal{W}(@) = 5 = 3 + 1 + 1 = \mathcal{W}(\&_L) + \mathcal{W}(\&_G) + \mathcal{W}(@_{sup}) = \mathcal{I}_c^+(D').$$

Por tanto, el 1*-reductante* $\mathcal{R}'$ *tiene la misma eficiencia que el reductante* $\mathcal{R}$ *cuando cuando contamos todos los operadores (conectivas y operadores primitivos) evaluados durante la fase interpretativa. Además, si hubiésemos estimado (en el coste interpretativo) sólo el número de pasos interpretativos, habríamos obtenido erróneamente que una derivación ejecutada usando* $\mathcal{R}$ *(es decir, la noción original de reductante) era más eficiente. Como ya dijimos, esto justifica la adopción de nuestro criterio de coste más refinado* $\mathcal{I}_c^+$*.*

Para finalizar, recordamos que en el Teorema 6.3.7 hemos justificado que el 1-reductante aporta las mismas respuestas computadas difusas que el reductante original (es decir, son proceduralmente equivalentes) y, además, con la misma eficiencia.

6.5. Conclusiones

Las aportaciones del capítulo se pueden sintetizar del siguiente modo.

- Hemos introducido una reformulación del concepto de reductante, justificando que aporta las mismas respuestas computadas difusas que el original y, además, con la misma eficiencia.
- Hemos probado, igualmente, la equivalencia semántica del mismo con el reductante debido a Medina *et al.* [2004].
- Este nuevo reductante tiene la ventaja de que podrá obtenerse, como detallaremos en el próximo capítulo, usando técnicas de evaluación parcial. Dichas técnicas nos permitirán, también, generalizar este concepto para obtener reductantes más generales que pueden ser más fáciles de construir vía PE (pueden tener una expresión notablemente más reducida si usamos técnicas de umbralización) y que sean equivalentes desde un punto de vista semántico y procedural.
- Hemos puesto de relieve cómo los métodos tradicionales que estiman el número de pasos de computación de una derivación produce resultados inapropiados cuando estimamos el esfuerzo computacional de la fase interpretativa en el marco de la programación lógica multi-adjunta.
- Hemos definido una medida de coste computacional para los programas lógicos multi-adjuntos, que estima adecuadamente el coste de la fase interpretativa de estos programas. De nuevo, se vuelve imprescindible la definición, que hemos contemplado en la Sección 3.1.2, de esta fase interpretativa como un sistema de transición de estados.
- Aplicamos esta medida de coste al cálculo de reductantes para justificar que el 1-reductante aportado tiene la misma eficiencia que el primitivo, es decir, coinciden el coste operacional e interpretativo de las derivaciones que aportan la misma respuesta computada difusa usando ambos tipos de reductante.
- Además, las medidas de coste formuladas son adecuadas para estimar, en el futuro, en el marco de la programación lógica multi-adjunta, la eficiencia de las técnicas de plegado/desplegado y de evaluación parcial que nuestro grupo de trabajo ha adaptado a dicho marco.

- Hemos introducido el concepto de compleción de un programa, que es un nuevo programa lógico multiadjunto formado por reductantes y que tiene la ventaja de que presenta propiedades de completitud frente al original.
- Hemos formulado una nueva semántica operacional para la programación lógica multi-adjunta que evita, en la práctica, el inconveniente de trabajar con programas infinitos.

Capítulo 7

Evaluación parcial y PE-reductantes

7.1. Introducción

Como su propio nombre indica, la *Evaluación Parcial* (EP) de un objetivo con respecto a un programa, consiste en evaluar parcialmente dicho objetivo mediante la generación de un árbol de desplegado (normalmente incompleto) para el mismo. La EP ofrece una amplia gama de aplicaciones como la transformación, optimización, depuración, corrección, inducción, análisis y, muy especialmente, la *especialización* automática de programas declarativos con respecto a parte de sus datos de entrada mediante el cálculo de *resultantes*.

Entre las múltiples técnicas de transformación que se han propuesto en la bibliografía para mejorar el código de los programas, la *Evaluación Parcial* es una de las más estudiadas y mejor conocidas, que además ofrece un entorno unificado para el estudio de compiladores e intérpretes [Jones *et al.*, 1993].

Informalmente, la EP persigue, entre sus principales objetivos, la optimización/especialización de un programa con el fin de dotarlo de un mejor comportamiento computacional.

Más concretamente, se espera que el programa especializado (también llamado programa *residual* o *parcialmente evaluado*) pueda ejecutarse más eficientemente que el programa original. Esto es posible gracias a que el programa residual evita rea-

lizar, en tiempo de ejecución, algunos cálculos que ya fueron efectuadas sólo una vez en tiempo de EP. Para alcanzar este objetivo, la EP usa computación simbólica así como algunas técnicas que proporciona el campo de la transformación de programas [Burstall y Darlington, 1977], especialmente la llamada *transformación de desplegado*.

Tal como hemos descrito en el Capítulo 4, desplegar consiste esencialmente en reemplazar una llamada por su definición, usando sustituciones apropiadas. En general, las técnicas de EP incluyen criterios de parada para garantizar la terminación del proceso.

Por consiguiente, se trata de un tipo de trasformación automática (esto es, el proceso de EP puede ser completado sin intervención humana), lo que constituye un importante rasgo diferenciador de estas técnicas con respecto a otros métodos de transformación de programas [Burstall y Darlington, 1977; Pettorossi y Proietti, 1994].

La EP ha sido ampliamente utilizada en diferentes paradigmas de programación declarativa, como es el caso de la programación funcional [Consel y Danvy, 1993; Jones *et al.*, 1993; Turchin, 1986], la programación lógica [Gallagher, 1993; Komorowski, 1982; Lloyd y Shepherdson, 1991; Pettorossi y Proietti, 1994], donde se denomina habitualmente *deducción parcial* y la programación lógico–funcional [Alpuente *et al.*, 1998; Albert *et al.*, 1998, 1999]. También ha sido aplicada al área de lenguajes imperativos (por ejemplo, el lenguaje C [Consel *et al.*, 1996]). Aunque los objetivos son similares, las técnicas son a menudo diferentes debido a los distintos modelos computacionales subyacentes.

Las técnicas convencionales de deducción parcial de programas lógicos se apoyan frecuentemente en la propagación de parámetros vía unificación [Glück y Sørensen, 1994], lo que forma parte del principio mismo de resolución. En este contexto, los datos de entrada son proporcionados al sistema como argumentos parciales de un objetivo (en la mayoría de los casos, una fórmula atómica). Dado un programa $\mathcal{P}$ y un conjunto de átomos S, para todo $A \in S$, se construye para $\mathcal{P} \cup \{\leftarrow A\}$ un árbol de búsqueda parcial finito, $\tau(A)$. En el proceso de construcción del árbol, conforme a cierto criterio, los átomos son desplegados o no desplegados. Entonces, el programa residual $\mathcal{P}'$ se obtiene recogiendo el conjunto de *resultantes*, que se construyen como sigue: *i*) considerando las hojas de ramas de no fallo de $\tau(A)$, digamos $\mathcal{Q}_1, \ldots, \mathcal{Q}_r$, y las sustituciones computadas a lo largo de estas ramas, digamos $\theta_1, \ldots, \theta_r$, y finalmente *ii*) construyendo las cláusulas: $A\theta_1 \leftarrow \mathcal{Q}_1, \ldots, A\theta_r \leftarrow \mathcal{Q}_r$, que son los

resultantes. La restricción para especializar fórmulas atómicas está motivada porque los resultantes deben ser cláusulas de Horn.

Se garantiza la corrección fuerte de la transformación cuando $\mathcal{P}' \cup \{\mathcal{G}\}$ es *S-cerrado*, esto es, todo átomo de $\mathcal{P}' \cup \{\mathcal{G}\}$ es una instancia de un átomo de S. Es necesaria una condición de *independencia*, que se cumple si ningún par de átomos de S tiene una instancia común, para garantizar que el programa residual $\mathcal{P}'$ no genera respuestas adicionales [Lloyd y Shepherdson, 1991].

Además, la EP se ha aplicado extensivamente a una gran variedad de problemas concretos entre los que pueden citarse [Jones *et al.*, 1993]: la especialización de consultas en bases de datos; la demostración automática de teoremas; la optimización del procedimiento de trazado de rayos en el área de la generación de gráficos por computador; el mantenimiento del software y la comprensión de programas; el entrenamiento de una red neuronal; y la especialización de simuladores de circuitos.

En este capítulo adaptamos las nociones clásicas de EP al contexto de la programación lógica multi-adjunta y mostramos su idoneidad para obtener programas más eficientes. Además, observamos ciertas relaciones entre la obtención de reductantes descritos en el capítulo anterior y las técnicas clásicas de la evaluación parcial. Estas semejanzas nos sugieren un método para calcular *reductantes* usando técnicas de EP en el contexto difuso, lo que supone una aplicación completamente novedosa en dicho contexto.

La importancia de relacionar el concepto de EP de un átomo en un programa lógico multi-adjunto con la construcción de un reductante para dicho átomo, estriba en el hecho de que un cálculo eficiente para este último concepto resulta vital si realmente estamos interesados en disponer de sistemas con garantías de completitud para este tipo de lenguajes.

La idea consiste en disminuir el impacto negativo de la incorporación de reductantes a un programa lógico multi-adjunto, a través de un preproceso en el que los reductantes son parcialmente evaluados antes de incluirlos en el programa final: puesto que la fase de EP produce un conjunto refinado de reductantes, el esfuerzo computacional realizado (sólo una vez) en tiempo de generación es evitado (muchas veces) en tiempo de ejecución.

Partiendo de las técnicas clásicas de evaluación parcial para programas lógico-funcionales ensayadas en nuestro grupo [Alpuente *et al.*, 1997a; Albert *et al.*, 1998], y de la experiencia en adaptar la transformación de desplegado en la programación lógica difusa (como concretamos en los Capítulos 4 y 5), en este capítulo nos

proponemos abordar entre otras las siguientes tareas:

- Definir el concepto de EP para programas y objetivos lógicos multi-adjuntos, adaptando las técnicas que surgieron en el campo de la dedución parcial de programas lógicos puros [Gallagher, 1993; Komorowski, 1982; Lloyd y Shepherdson, 1991; Pettorossi y Proietti, 1994], pero haciendo uso ahora de la regla de desplegado que desarrollamos en el Capítulo 4 para esta clase de programas lógicos borrosos. Obedeciendo a esta idea, trataremos de desplegar objetivos dando pasos admisibles, tanto como sea posible, a fin de obtener una versión optimizada (especializada) del programa original.

- Como aplicación immediata del nuevo marco de EP, ilustrar sus beneficios mediante algunos ejemplos sencillos de especialización, donde se obtienen programas que se ejecutan más eficientemente que los originales.

- Proponer un procedimiento para obtener *reductantes* usando técnicas de EP, lo que supone una aplicación completamente novedosa en el nuevo contexto.

- Definir el concepto de *PE-reductante* que generaliza el reductante de Medina *et al.* [2004] y 1-reductante que introducimos en la Definición 6.1.2.

- Obtener los *PE-reductantes* haciendo uso de técnicas de evaluación parcial con umbralización, lo que nos permitirá reducir su expresión y acortar o disminuir asimismo aquellas secuencias de derivación en las que intervengan.

- Diseñar, además, un algoritmo para computar los llamados *PE-reductantes* mediante técnicas de evaluación parcial basadas en desplegado con un conjunto de umbrales dinámico.

- Discutir en ejemplos representativos los beneficios de nuestras técnicas, refiriendo la ganancia en eficiencia no sólo cuando construimos el propio *PE-reductante*, sino también cuando lo usamos en la ejecución de objetivos.

- Abordar finalmente, en la Sección 7.8, propiedades formales de los *PE-reductantes* y los beneficios que aporta su incorporación a los programas multi-adjuntos.

Parte del material que figura en este capítulo se encuentra publicado en [Julián *et al.*, 2006b], [Julián *et al.*, 2007c], [Julián *et al.*, 2007b] y [Julián *et al.*, 2009].

7.2. Conceptos básicos

En lo que sigue construiremos, por primera vez, un marco de evaluación parcial para programas lógicos multi-adjuntos, mostrando su aptitud para especializar programas tal y como se ha hecho tradicionalmente en otros contextos declarativos. Empezamos por formalizar las nociones básicas involucradas en la evaluación parcial de programas lógicos multi-adjuntos.

Definición 7.2.1 (Resultante)**.** *Sea $\mathcal{P}$ un programa multi-adjunto y sea $\mathcal{Q}$ un objetivo. Dada una secuencia de pasos admisibles e interpretativos $\langle \mathcal{Q} \,;id\rangle \rightarrow^{+} \langle \mathcal{Q}' \,;\sigma\rangle$, cuya longitud es estrictamente mayor que cero, definimos el* resultante *de esta derivación (posiblemente incompleta, si $\mathcal{Q}'$ es una agregación de fórmulas atómicas y valores del retículo) como: $\langle Q\sigma \leftarrow \mathcal{Q}' \,;\top\rangle$, donde "$\leftarrow$" es cualquier implicación con un conjuntor adjunto.*

Observemos que, en contraste con lo que consideramos en la semántica procedural descrita en la Sección 3.1, los pasos admisibles e interpretativos pueden intercalarse en el cálculo de un resultante. Además, para la construcción de un resultante, pretendemos beneficiarnos de esta ventaja aplicando tantos pasos interpretativos como sea posible antes de ejecutar un paso admisible. Por tanto, en la práctica, concederemos preferencia a los pasos interpretativos sobre los pasos admisibles cuando acometamos la EP.

Este proceso tiene semejanzas con la técnica de *normalización*[1] introducida en el contexto de la programación lógico funcional para reducir el indeterminismo de una computación [Fay, 1979]. En consecuencia, también llamaremos *normalización* a la secuencia de pasos interpretativos ejecutados antes de un paso de desplegado operacional.

Asimismo, observemos que la componente "regla" de un resultante no es en general una regla, porque el objetivo $\mathcal{Q}$ puede ser una agregación de fórmulas atómicas. Un resultante es particularmente significativo cuando el objetivo original $\mathcal{Q}$ es un átomo A, ya que en ese caso el resultante $\langle A\sigma \leftarrow \mathcal{Q} \,;\top\rangle$ es un par $\langle \mathit{regla}; \mathit{grado\ de\ verdad}\rangle$ que forma parte del programa transformado. El siguiente ejemplo ilustra la noción de resultante.

[1] En una estrategia de *estrechamiento normalizante* un término se reescribe a su forma normal antes de que se aplique un paso de estrechamiento. Este proceso se aplica también a estrechamiento basado en EP y estrechamiento basado en técnicas de transformación de plegado/desplegado.

Ejemplo 7.2.2. *Para el programa $\mathcal{P}$ dado a continuación, para el que se considera de nuevo el intervalo $([0,1],\leq)$ como retículo asociado, y el objetivo $p(X)$,*

$$\mathcal{R}_1 : \langle p(a) \leftarrow_{\mathtt{L}} q(X,a); \quad 0.7\rangle \qquad \mathcal{R}_5 : \langle s(a) \leftarrow_{\mathtt{G}} t(a); \quad 0.5\rangle$$

$$\mathcal{R}_2 : \langle p(a) \leftarrow_{\mathtt{G}} s(Y); \quad 0.5\rangle \qquad \mathcal{R}_6 : \langle s(b) \leftarrow_{\mathtt{L}} t(b); \quad 0.8\rangle$$

$$\mathcal{R}_3 : \langle p(Y) \leftarrow_{\mathtt{G}} q(b,Y) \&_{\mathtt{L}}\ t(Y); \quad 0.8\rangle \qquad \mathcal{R}_7 : \langle t(a) \leftarrow_{\mathtt{L}} p(X); \quad 0.9\rangle$$

$$\mathcal{R}_4 : \langle q(b,a) \leftarrow\ ; \quad 0.9\rangle \qquad \mathcal{R}_8 : \langle t(b) \leftarrow_{\mathtt{G}} q(X,a); \quad 0.9\rangle$$

se obtiene el resultante $\langle p(a) \leftarrow\ 0.7 \&_{\mathtt{L}} 0.9; 1\rangle$ *a partir de la derivación:*

$$\langle p(X); id\rangle \rightarrow_{AS1}{}^{\mathcal{R}_1} \langle 0.7 \&_{\mathtt{L}} q(X_1,a); \{X/a\}\rangle \rightarrow_{AS1}{}^{\mathcal{R}_4} \langle 0.7 \&_{\mathtt{L}} 0.9; \{X/a, X_1/b\}\rangle$$

Adviértase que un paso admisible dado con el resultante y el objetivo $p(X)$ simula los efectos de los dos pasos admisibles dados en la anterior derivación, lo que evidencia la mejora introducida por el proceso de EP en el programa transformado final.

Por tanto, en general, el resultante condensa en un sólo paso toda la información de la derivación original correspondiente al objetivo A (de ahí su nombre). También compila la información del grado de verdad en la primera componente del par. Así, para reproducir el efecto de la derivación original, es suficiente tomar como grado de verdad del resultante el supremo del retículo L.

Tradicionalmente, la evaluación parcial de un objetivo atómico se define construyendo árboles de búsqueda incompletos para el objetivo y extrayendo la definición especializada –los resultantes– a partir de las ramas (desde la raíz hasta la hoja) que no son de fallo. Para el lenguaje de programación lógica multi-adjunta no existen hojas de fallo, dado que si ninguna regla del programa unifica con el objetivo, se da un paso de computación con la regla $AS3$ de la Definición 3.1.1.

En la definición posterior, adaptamos al nuevo contexto la noción clásica de árbol de desplegado.

Definición 7.2.3 (Árbol de desplegado)**.** *Sea $\mathcal{P}$ un programa multi-adjutno y $\mathcal{Q}$ un objetivo. Un* árbol de desplegado *τ_φ para $\mathcal{P}$ y $\mathcal{Q}$ (usando la regla de computación φ) es un conjunto de pares de nodos $\langle objetivo; sustitución\rangle$ verificando las siguientes condiciones:*

1. *El nodo raíz de τ_φ es $\langle \mathcal{Q}\ ; id\rangle$, donde id es la sustitución identidad.*

2. *Si* $\mathcal{N}_i \equiv \langle \mathcal{Q}[A]; \sigma \rangle$ *es un nodo de* τ_φ *y suponemos que* $\varphi(\mathcal{Q}) = A$ *es el átomo seleccionado, entonces para cada regla* $\mathcal{R} : \langle H \leftarrow \mathcal{B}; v \rangle$ *en* $\mathcal{P}$*, con* $\theta = mgu(\{H = A\})$*,* $\mathcal{N}_{ij} \equiv \langle (\mathcal{Q}[A/v\&\mathcal{B}])\theta; \sigma\theta \rangle$ *es un nodo de* τ_φ*.*

3. *Si* $\mathcal{N}_i \equiv \langle \mathcal{Q}[@(r, r')]; \sigma \rangle$ *es un nodo de* τ_φ*,* $\mathcal{N}_{ij} \equiv \langle \mathcal{Q}[@(r, r')/\dot{@}(r, r')]); \sigma \rangle$ *es un nodo de* τ_φ*.*

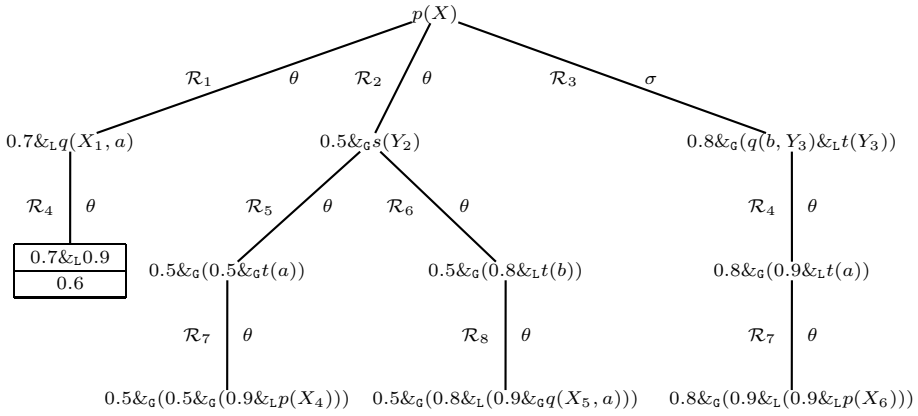
Observemos que el segundo y tercer caso están asociados, respectivamente, a la ejecución de un paso admisible del tipo $AS1$ contemplado en la Definición 3.1.1 y de un paso interpretativo conforme a la Definición 3.1.6.

Un árbol de desplegado *incompleto* es un árbol de desplegado que puede contener también hojas donde ningún átomo (o expresión interpretable) ha sido seleccionada para un posterior paso de desplegado. Es decir, permitiremos terminar una derivación en cualquier punto adecuado.

Definición 7.2.4 (Evaluación parcial de un átomo)**.** *Sea* $\mathcal{P}$ *un programa multi-adjunto,* A *un objetivo atómico, y* τ *un árbol de desplegado finito (posiblemente incompleto) para* $\mathcal{P}$ *y* A*, conteniendo al menos un nodo además de la raíz.*

Sea $\{\mathcal{Q}_i \mid i = 1, \ldots, k\}$ *el conjunto de hojas de* τ*, y* $P' = \{\langle A\sigma_i \leftarrow \mathcal{Q}_i \,; \top \rangle \mid i = 1, \ldots, k\}$ *el conjunto de resultantes asociado con el conjunto de derivaciones* $\{\langle A \,; id \rangle \rightarrow^+ \langle \mathcal{Q}_i \,; \sigma_i \rangle \mid i = 1, \ldots, k\}$*. Entonces, el conjunto* P' *se llama* evaluación parcial *de* A *en* $\mathcal{P}$ *(usando* τ*).*

Ejemplo 7.2.5. *Dado el programa* $\mathcal{P}$ *del Ejemplo 7.2.2, podemos construir el siguiente árbol de desplegado de nivel* 3 *(esto es, todas sus ramas se han desplegado no más de* 3 *pasos) para el programa* $\mathcal{P}$ *y el átomo* $p(X)$*:*



en el que las sustituciones de cada estado se anotan en la rama que lleva al mismo, salvo para el nodo raíz que tiene asociada la sustitución id; por simplicidad, las restringimos a las variables del objetivo, de modo que $\theta = \{X/a\}$ *y* $\sigma = \{X/Y_3\}$. *Cada vez que se usa una cláusula se renombran sus variables y, además, los nodos donde los pasos de normalización han originado nodos adicionales, se encierran en cajas. A partir de este árbol obtenemos el conjunto de resultantes*

$$\mathcal{R}'_1 : \langle p(a) \leftarrow 0.6; 1 \rangle$$

$$\mathcal{R}'_2 : \langle p(a) \leftarrow 0.5 \,\&_{\mathtt{G}}(0.5 \,\&_{\mathtt{G}}(0.9 \,\&_{\mathtt{L}}\, p(X_4))); 1 \rangle$$

$$\mathcal{R}'_3 : \langle p(a) \leftarrow 0.5 \,\&_{\mathtt{G}}(0.8 \,\&_{\mathtt{L}}(0.9 \,\&_{\mathtt{G}} q(X_5, a))); 1 \rangle$$

$$\mathcal{R}'_4 : \langle p(a) \leftarrow 0.8 \,\&_{\mathtt{G}}(0.9 \,\&_{\mathtt{L}}(0.9 \,\&_{\mathtt{L}} p(X_6))); 1 \rangle$$

Es sencillo extender la Definición 7.2.4 a un conjunto de fórmulas atómicas. Si S es un conjunto finito de átomos, entonces una evaluación parcial de S en $\mathcal{P}$ (también llamada una *evaluación parcial de* $\mathcal{P}$ *con respecto a* S) es la unión de las evaluaciones parciales de los elementos de S en $\mathcal{P}$.

La restricción de especializar objetivos atómicos no es una limitación severa que impida la especialización de objetivos más complejos, si convenimos en realizar la especialización de sus componentes por separado[2]. Dado un programa $\mathcal{P}$ y un objetivo compuesto $\mathcal{Q}$, suponiendo que $S = \{B_1, \ldots, B_n\}$ es el conjunto de constituyentes atómicos de $\mathcal{Q}$, la evaluación parcial de $\mathcal{Q}$ en $\mathcal{P}$ es la evaluación parcial de $\mathcal{P}$ con respecto a S.

7.3. Reductantes frente a PE-reductantes

En esta sección, volvemos sobre el concepto de reductante de Medina *et al.* [2004] y el concepto de 1-reductante y, después de mostrar la relación entre la construcción de reductantes y las técnicas propias del campo de la evaluación parcial, superamos

[2]Sin embargo, es necesario admitir que se pierden algunas posibilidades de alcanzar una buena especialización. Esto puede evitarse introduciendo técnicas adecuadas para la reordenación y partición de objetivos complejos en porciones más pequeñas antes de pasar a su especialización, como se ha propuesto en el campo de la deducción parcial conjuntiva [Glück *et al.*, 1996; Leuschel *et al.*, 1996].

estas definiciones originales proponiedo una noción mejorada de reductante al que denominaremos PE-reductante.

Tal como consideramos al comienzo de la Sección 6.1, un programa lógico multi-adjunto, interpretado en un retículo parcialmente ordenado, debe contener todos sus reductantes a fin de asegurar la completitud. Obviamente, esto aumenta tanto del tamaño (normalmente infinito) como el tiempo de ejecución del programa *"completado"* final. Sin embargo, este efecto negativo puede ser considerablemente atenuado si los reductantes propuestos han sido parcialmente evaluados antes de incluirlos en programa final.

La Definición 7.3.5 posterior contemplará una noción de reductante más refinada y general que la de 1-reductante (que a su vez caracterizaremos en la Definición 7.3.2) y la de reductante de Medina *et al.* [2004]. Nos permitirá disponer de una aproximación más flexible de este concepto incorporando el uso de un árbol de desplegado arbitrario. En efecto, si τ es un árbol de desplegado cualquiera para un programa $\mathcal{P}$ y un átomo básico A, es posible construir la versión refinada que nosotros llamaremos *PE-reductante* para A en $\mathcal{P}$. Mostraremos cómo este nuevo concepto, que usa técnicas de evaluación parcial y extiende a los primitivos, resuelve los problemas referidos anteriormente.

En el capítulo anterior hemos contemplado un concepto de 1-reductante que es una reformulación del debido a Medina *et al.* [2004]. Volvemos a traer aquí su definición para ligarla a nociones de EP de programas lógicos multi-adjuntos.

Definición 7.3.1. *Sea $\mathcal{P}$ un programa multi-adjunto y A un átomo básico. Si $\{\langle H_i \leftarrow_i \mathcal{B}_i; v_i\rangle \in \mathcal{P} : \exists \theta_i / A = H_i\theta_i\}$ es el conjunto (no vacío) de reglas de $\mathcal{P}$ cuya cabeza unifica con A, el* 1-reductante *de A en $\mathcal{P}$ es la regla*

$$\langle A \leftarrow @_{sup}((v_1 \&_1 \mathcal{B}_1)\theta_1, \ldots, (v_n \&_n \mathcal{B}_n)\theta_n); \top\rangle$$

Nos interesa destacar ahora que este 1-reductante puede construirse usando técnicas propias del campo de la evaluación parcial, tal y como vemos en la siguiente definición.

Definición 7.3.2 (Construcción de 1-reductantes)**.** *Dado un programa $\mathcal{P}$ y un objetivo atómico básico A, enumeramos los siguientes pasos en la construcción de un* 1*-reductante para A en $\mathcal{P}$:*

1. *Construir el árbol de desplegado* de nivel 1*, τ, para $\mathcal{P}$ y A, esto es, el árbol obtenido ejecutando un sólo paso de desplegado sobre el átomo A en $\mathcal{P}$.*

2. *Recolectar el conjunto de hojas de* τ*:*
   $S = \{\langle (v_1 \&_1 \mathcal{B}_1)\theta_1; \theta_1\rangle, \ldots, \langle (v_n \&_n \mathcal{B}_n)\theta_n; \theta_n\rangle\}$
3. *Construir la regla* $\langle A \leftarrow @_{sup}((v_1 \&_1 \mathcal{B}_1)\theta_1, \ldots, (v_n \&_n \mathcal{B}_n)\theta_n); \top\rangle$.

Ejemplo 7.3.3. *Dado el retículo* $([0,1], \leq)$*, donde "*$\leq$*" es el orden usual, consideremos el siguiente programa lógico multi-adjunto* $\mathcal{P}$*:*

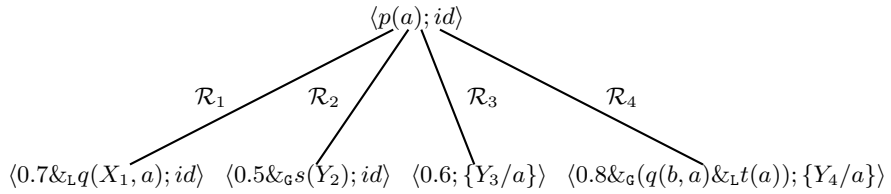
$\mathcal{R}_1 : \langle p(a) \leftarrow_{\mathtt{L}} q(X,a); \quad 0.7\rangle$ $\quad$ $\mathcal{R}_5 : \langle q(b,a) \leftarrow ; \quad 0.9\rangle$

$\mathcal{R}_2 : \langle p(a) \leftarrow_{\mathtt{G}} s(Y); \quad 0.5\rangle$ $\quad$ $\mathcal{R}_6 : \langle s(a) \leftarrow_{\mathtt{G}} t(a); \quad 0.5\rangle$

$\mathcal{R}_3 : \langle p(Y) \leftarrow ; \quad 0.6\rangle$ $\quad$ $\mathcal{R}_7 : \langle s(b) \leftarrow ; \quad 0.8\rangle$

$\mathcal{R}_4 : \langle p(Y) \leftarrow_{\mathtt{G}} q(b,Y) \&_{\mathtt{L}} t(Y); \quad 0.8\rangle$ $\quad$ $\mathcal{R}_8 : \langle t(a) \leftarrow_{\mathtt{L}} p(X); \quad 0.9\rangle$

El árbol de desplegado de nivel 1 *para el programa* $\mathcal{P}$ *y el átomo* $p(a)$ *es:*



a partir del que obtenemos el PE-reductante:

$$\langle p(a) \leftarrow @_{sup}(0.7 \&_{\mathtt{L}} q(X_1,a), 0.5 \&_{\mathtt{G}} s(Y_2), 0.6, 0.8 \&_{\mathtt{G}} (q(b,a) \&_{\mathtt{L}} t(a))); \ 1\rangle$$

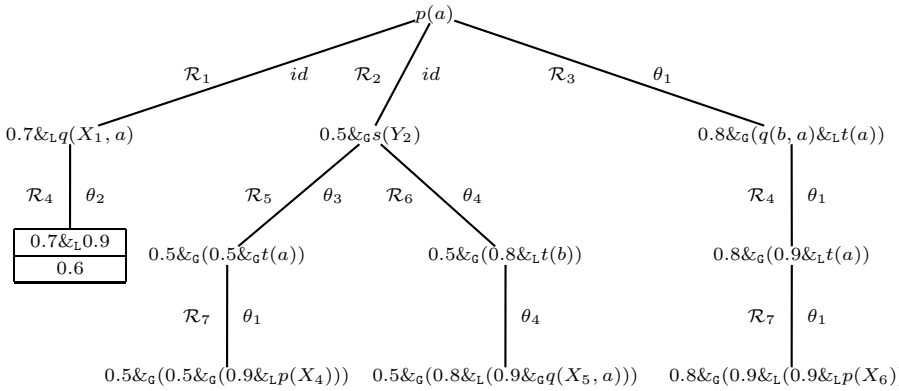
Por otra parte, volvemos a contrastar las semejanzas/diferencias entre esta regla y el reductante aportado por la Definición 6.1.1:

$$\langle p(a) \leftarrow @(q(X_1,a), s(Y_2), 0.6, q(b,a) \&_{\mathtt{L}} t(a)); \ 1\rangle$$

donde $\dot{@}(b_1,b_2,b_3,b_4) = sup\{0.7 \dot{\&}_{\mathtt{L}} b_1, 0.5 \dot{\&}_{\mathtt{G}} b_2, b_3, 0.8 \dot{\&}_{\mathtt{G}} b_4\}$.

Tal como vamos a considerar en la próxima subsección, la Definición 7.3.2 de 1-reductante puede generalizarse, conduciendo a una aproximación más flexible de este concepto, si permitimos desplegar árboles de cualquier nivel. En efecto, usando un árbol de desplegado arbitrario, τ, para un programa $\mathcal{P}$ y un átomo básico A, es posible construir una versión más refinada de la noción de reductante como sugiere el ejemplo inmediato.

Ejemplo 7.3.4. *La figura muestra un árbol de desplegado para el programa $\mathcal{P}$ del Ejemplo 7.2.2 y el átomo básico $p(a)$ de nivel 3 (esto es, todas sus ramas han sido desplegadas no más de 3 pasos).*



con $\theta_1 = \{Y_3/a\}, \theta_2 = \{X_1/b\}, \theta_3 = \{Y_2/a\}, \theta_4 = \{Y_2/b\}$. Recogiendo las hojas del mismo, obtenemos el reductante $\langle p(a) \leftarrow @_{sup}(0.6; 0.5\&_G(0.5\&_G(0.9\&_L p(X_4))) ; 0.5\&_G(0.8\&_L(0.9\&_G q(X_5, a))); 0.8\&_G(0.9\&_L(0.9\&_L p(X_6)))); \ 1\rangle$.

La principal novedad de la siguiente noción de PE-reductante (que generaliza tanto el reductante de Medina *et al.* [2004] como nuestra noción previa de 1-reductante que ha sido introducida por primera vez en [Julián *et al.*, 2006b]), es el hecho de que está basada directamente en el conjunto de hojas de un árbol de desplegado dado.

En lo sucesivo suponemos que $\leftarrow$ es la implicación de cualquier par adjunto $\langle \leftarrow, \& \rangle$.

Definición 7.3.5 (PE-reductante)**.** *Sea $\mathcal{P}$ un programa multi-adjunto, A un átomo básico, y τ un árbol de desplegado para A en $\mathcal{P}$. Un PE-reductante para A en $\mathcal{P}$ con respecto a τ, es una regla $\langle A \leftarrow @_{sup}(\mathcal{D}_1, \ldots, \mathcal{D}_n); \top \rangle$, donde la función de verdad del agregador $@_{sup}$ se define como $\dot{@}_{sup}(d_1, \ldots, d_n) = sup\{d_1, \ldots, d_n\}$, y $\mathcal{D}_1, \ldots, \mathcal{D}_n$ son, respectivamente, las hojas de τ.*

Es obvio que si τ es un árbol de desplegado de nivel 1 para A en $\mathcal{P}$, el PE-reductante para A en $\mathcal{P}$ coincide con el 1-reductante. Además, téngase en cuenta que nuestra definición de PE-reductante respeta la sintaxis del lenguaje multi-adjunto (véase la Sección 3.1), donde los grados de verdad y las conjunciones adjuntas están efectivamente admitidas en el cuerpo de las reglas de programa.

Como en el caso de los resultantes, los PE-reductantes incorporan información acerca de todos los aspectos relevantes de las reglas $\langle H_i \leftarrow_i \mathcal{B}_i; v_i \rangle$ usadas en la

evaluación del átomo A: el grado de verdad v_i, la implicación adjunta y los operadores conjunción, las sustituciones computadas y las instancias de los cuerpos $\mathcal{B}_i$.

Por otra parte, fijado el programa $\mathcal{P}$ y el átomo básico A, si $\{\langle H_i \leftarrow_i \mathcal{B}_i; v_i\rangle \in \mathcal{P} : \exists\theta_i / A = H_i\theta_i\}$ es el conjunto –no vacío– de reglas de $\mathcal{P}$ cuya cabeza unifica con A y suponiento que el árbol de desplegado elegido es un árbol de nivel 1, entonces el PE-reductante es la regla $\langle A \leftarrow @_{sup}((v_1 \&_1 \mathcal{B}_1)\theta_1, \ldots, (v_n \&_n \mathcal{B}_n)\theta_n); \top\rangle$, que coincide con el 1-reductante y cuya forma se asemeja notablemente al que aporta la Definición 6.1.1.

Conviene observar que un PE-reductante puede construirse, a partir de la noción de árbol de desplegado, a través del siguiente procedimiento.

Definición 7.3.6 (Construcción de PE-reductantes)**.** *Sea $\mathcal{P}$ un programa multi-adjunto y A un objetivo atómico básico. Enumeramos los siguientes pasos en la construcción*[3] *de un PE-reductante de A en $\mathcal{P}$:*

1. *Construir un árbol de desplegado, τ, para $\mathcal{P}$ y A, esto es, el árbol obtenido al desplegar el átomo A en el programa.*

2. *Recolectar el conjunto de hojas $S = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$ de τ.*

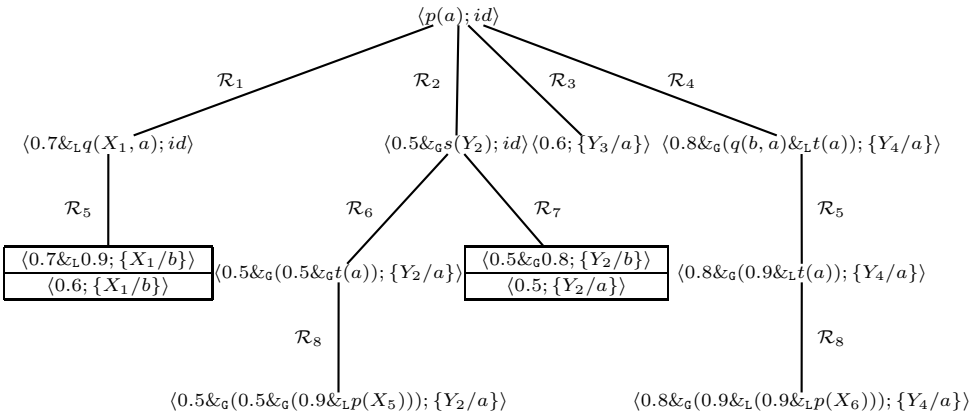
3. *Construir la regla $\langle A \leftarrow @_{sup}(\mathcal{D}_1, \ldots, \mathcal{D}_n); \top\rangle$, que es el PE-reductante de A en $\mathcal{P}$ con respecto a τ.*

El ejemplo que sigue contempla un PE-reductante obtenido a partir de un árbol de desplegado de nivel 3 (esto es, todas sus ramas han sido desplegadas no más que 3 pasos).

Ejemplo 7.3.7. *Sea $\mathcal{P}$ el programa del Ejemplo 7.3.3 y consideremos el átomo $p(a)$. En la figura siguiente, se encierran en cajas los nodos en los que se aplican pasos de normalización originando nodos adicionales. Recolectando las hojas del posterior árbol de desplegado, obtenemos el PE-reductante:*

$\mathcal{R} \equiv \langle p(a) \longleftarrow @sup(0.6, 0.5\&_{\mathtt{G}}(0.5\&_{\mathtt{G}}(0.9\&_{\mathtt{L}} p(X_5))), 0.5, 0.6, 0.8\&_{\mathtt{G}}(0.9\&_{\mathtt{L}}(0.9\&_{\mathtt{L}} p(X_6)))); 1\rangle$.

[3] Este proceso extiende al que contempla la Definición 7.3.2.



Dado que esta formulación está basada en técnicas de evaluación parcial, puede verse como un método que produce una especialización de un programa con respecto a un objetivo atómico que, en particular, permite computar la mayor respuesta correcta para dicho objetivo.

Aunque para el mismo programa $\mathcal{P}$ y átomo básico A es posible obtener distintos reductantes, dependiendo de la precisión del árbol de desplegado generado, justificaremos oportunamente que todos ellos permiten computar la misma mayor respuesta correcta para el objetivo A.

7.4. Construcción umbralizada de PE-reductantes

Si en el proceso de construcción de un PE-reductante, partimos de un árbol de desplegado umbralizado en lugar de uno arbitrario (como habíamos contemplado hasta el momento), podremos usarlo para obtener respuestas computadas para un objetivo dado con mucho menor esfuerzo computacional.

Esta construcción de reductantes usando técnicas de evaluación parcial con umbralización pretende reducir drásticamente el tamaño de los árboles de desplegado a partir de los que se obtiene el reductante.

La idea es combinar el propio proceso de evaluación parcial con técnicas de umbralización, para lograr los siguientes beneficios extra:

• La construcción del árbol de desplegado subyacente consume menos recursos computacionales (tanto memoria como CPU) al podar ramas innecesarias del árbol y, por tanto, reduciendo drásticamente su tamaño.

• Como consecuencia directa de lo anterior, se simplifica de manera notable la forma del reductante obtenido.
• Por último, aquellas secuencias de derivación realizadas en tiempo de ejecución, requieren menos pasos de computación cuando usamos esta noción refinada de *PE*-reductante, no sólo por los pasos ahorrados en tiempo de PE, sino también por los que se han eliminado definitivamente al umbralizar.

En lo que sigue aportamos un algoritmo eficiente para la construcción de un *PE*-reductante basado en desplegado con un conjunto de umbrales dinámicos. Previamente, introducimos algunos resultados de carácter básico en los que se fundamenta el algoritmo. Posteriormente, mostraremos los beneficios de esta técnica sobre un ejemplo comparativo.

7.5. Cota superior de una computación y umbrales

En el contexto de una computación difusa parece lógico omitir una derivación si hay evidencias de que conduce a una respuesta computada difusa cuyo grado de verdad caerá por debajo de un cierto umbral $\mathcal{V}$. En nuestro marco, esta situación puede ser detectada "por anticipado", es decir, antes de que la computación difusa haya sido completada. El resultado posterior facilita la base teórica que nos permite sostener esta afirmación.

Proposición 7.5.1. *Sea* $(L, \leq, \leftarrow_1, \&_1, \ldots, \leftarrow_n, \&_n)$ *un retículo multi-adjunto. Entonces, para todo* $x, y \in L$*, se verifica:* (1) $x\&_i y \leq x$, (2) $x\&_i y \leq y$.

Demostración. El apartado (1) es una consecuencia sencilla de la definición de retículo multi-adjunto (véase la Definición 2.5.1). En primer lugar, $x\&_i y \leq x\&_i \top$ puesto que el operador $\&_i$ es, por definición, creciente en ambos argumentos –es decir, si $x_1, x_2, x_3 \in L$ y $x_1 \leq x_2$ entonces $x_1\&_i x_3 \leq x_2\&_i x_3$ y $x_3\&_i x_1 \leq x_3\&_i x_2$– y $\top$ es el supremo de L –esto es, $y \leq \top$ para todo $y \in L$–. Además, el operador adjunto $\&_i$ satisface también, por definición de retículo multi-adjunto, que $x\&_i \top = x$ para todo $x \in L$, lo que concluye la demostración. La prueba de (2) es totalmente análoga. □

El resultado posterior es un corolario sencillo de la Proposición 7.5.1 y muestra que $inf\{x, y\}$ es una cota superior de $x\&_i y$.

Proposición 7.5.2. *Sea* $(L, \leq, \leftarrow_1, \&_1, \ldots, \leftarrow_n, \&_n)$ *un retículo multi-adjunto. Entonces, para todo* $x, y \in L$ *y toda conjunción adjunta* $\&_i$[4]*, se tiene:* $x\&_i y \leq inf\{x, y\}$.

Como consecuencia de la Proposición 7.5.1, merece la pena destacar que, en un paso admisible $\langle \mathcal{Q}[A]; \sigma\rangle \rightarrow_{AS} \langle (\mathcal{Q}[A/v\&_i\mathcal{B}])\theta; \sigma\theta\rangle$, (el grado de verdad de) la componente $v\&_i\mathcal{B}$, introducida por la regla, es menor o igual que su grado de verdad v. Y esto es independiente del grado de verdad eventualmente computado para el subojetivo $\mathcal{B}$. Por tanto, si el objetivo $\mathcal{Q}$ está compuesto por conectivas que satisfacen las condiciones de la Proposición 7.5.1 (notemos que esta restricción es trivialmente cierta para un objetivo atómico), v es una cota superior del grado de verdad computable para $\mathcal{Q}$.

La consideración anterior nos encamina, de manera natural, a la noción de desplegado umbralizado, donde sólo se permite ejecutar pasos de desplegado que conduzcan a nodos con un grado de verdad previsiblemente mayor que (o no comparable con) un umbral $\mathcal{V}$. En otros términos, cuando la cota superior del grado de verdad de un nodo cae por debajo del umbral $\mathcal{V}$, se evita el desplegado del mismo. En la próxima sección se precisa este concepto.

7.6. Un algoritmo concreto

Suponemos en esta sección que todas las conectivas considerados satisfacen la Proposición 7.5.1. En la práctica, esto no conlleva una severa pérdida de generalidad.

Durante la construcción de un PE-reductante muchos pasos de desplegado son inútiles, puesto que generan nodos hoja que no aportan información significativa a la computación final del supremo. En efecto, en el Ejemplo 7.3.7, el nodo $\langle 0.5; \{Y_2/a\}\rangle$ no es significativo, ya que $0.5 \leq 0.6$ –el grado de verdad de un nodo hoja totalmente evaluado– ni el nodo $\langle 0.5\&_{\mathtt{G}}(0.5\&_{\mathtt{G}}(0.9\&_{\mathtt{L}}p(X_5))); \{Y_2/a\}\rangle$, ya que, por la Proposición 7.5.1, incluso en el caso en que la evaluación completa subsiguiente del subobjetivo $p(X_5)$ alcanzara el supremo $\top$ de L, tenemos $0.5\dot{\&}_{\mathtt{G}}(0.5\dot{\&}_{\mathtt{G}}(0.9\dot{\&}_{\mathtt{L}}\top)) \leq 0.5 \leq 0.6$. Así, el PE-reductante para $p(a)$ en el programa del Ejemplo 7.3.7 puede escribirse en una forma más precisa/simplificada como:

$$\langle p(a) \leftarrow @_{sup}\{0.6, 0.8\&_{\mathtt{G}}(0.9\&_{\mathtt{L}}(0.9\&_{\mathtt{L}}p(X_6)))\}; 1\rangle$$

[4]Los resultados de la Proposición 7.5.1 y la Proposición 7.5.2 que hemos obtenido para el operador adjunto $\&_i$ se extiende sin dificultad para una t-norma cualquiera sin más que tener en cuenta la Definición 1.3.1.

Podemos optimizar la construcción de PE-reductantes si usamos una adaptación de la noción de árbol de desplegado (Definición 7.2.3) en la que:

i) los nodos contienen información acerca de una cota superior del grado de verdad asociado a la componente objetivo; y

ii) un conjunto de umbrales se ajusta dinámicamente para limitar la generación de nodos inútiles. Este último aspecto facilita grandes oportunidades para reducir el tamaño del árbol de desplegado, deteniendo el desplegado de aquellos nodos en los que la componente (cota superior del) grado de verdad quede por debajo de un cierto umbral $\mathcal{V}$.

Teniendo en cuenta estos hechos, proponemos un procedimiento de construcción en dos fases. En la primera fase, construimos (recorremos) un árbol de desplegado umbralizado incompleto, para un programa $\mathcal{P}$ y un objetivo A, intentando limitar la generación de nodos inútiles. En la construcción de este árbol, almacenamos los nodos hoja en una lista. En la segunda fase, en la que construimos el PE-reductante, recorremos la lista anterior y eliminamos los nodos hoja que no pueden contribuir a la computación del supremo.

Como en un procedimiento de prueba clásico, tres puntos son importantes: la regla de computación (es decir, la función de selección usada para decidir qué átomo debe ser explotado en el siguiente paso de computación[5]); el orden de exploración de las reglas del programa (esto es, el orden en que se ensayan las reglas para el desplegado) y la estrategia de búsqueda (en anchura o en profundidad).

Asumimos que el algoritmo que presentamos a continuación es paramétrico con respecto a todos estos puntos, así como también respecto a un criterio de parada que asegure la terminación de los sucesivos pasos de desplegado[6].

Algoritmo 7.6.1 (Desplegado con un conjunto de umbrales dinámicos)**.**

[5] Hemos probado en el Teorema 3.1.11 un resultado de independencia de esta función de selección, como es también usual en otros paradigmas lógicos puros. De manera similar a PROLOG, en los ejemplos, siempre explotamos el átomo que se encuentra en la parte más a la izquierda de un objetivo dado.

[6] El *problema de terminación local* puede resolverse de una forma simple, imponiendo un nivel de profundidad máximo para el desplegado, o usando aproximaciones más refinadas como los métodos basados en órdenes bien fundados o similares (véase, [Arts y Giesl, 2000a; Lee *et al.*, 2001]).

>>>>> **[ENTRADA]:** Un programa $\mathcal{P}$ y un átomo básico A.

1. Inicializar $HOJAS = []$ (lista vacía), y $UMBRALES = [\bot]$;
2. Construir el nodo raíz $\langle A; id; \top \rangle$ e inicializar $ABIERTO = [\langle A; id; \top \rangle]$;
3. Mientras $ABIERTO \neq []$ hacer:
   a) Elegir un nodo, digamos $\mathcal{N}_i$, de la lista $ABIERTO$ (siguiendo la estrategia de búsqueda);
   b) Si $\mathcal{N}_i$ satisface el criterio de parada, entonces añadir el nodo $\mathcal{N}_i$ a la lista $HOJAS$;
   c) En otro caso, supongamos que $\mathcal{N}_i \equiv \langle \mathcal{Q}[E]; \sigma; u \rangle$, donde E es el átomo seleccionado en $\mathcal{Q}$ (siguiendo la regla de computación);
   Para cada regla $\mathcal{R}_j : \langle H \leftarrow \mathcal{B}; v \rangle \in \mathcal{P}$ (siguiendo el orden de exploración de reglas), con $\theta = mgu(\{E = H\})$ y NO EXISTE ningún $\mathcal{V} \in UMBRALES$ tal que $v < \mathcal{V}$ hacer:
   - Generar el nodo hijo $\mathcal{N}_{ij} \equiv \langle (\mathcal{Q}[E/v\&\mathcal{B}])\theta; \sigma\theta; inf\{u, v\} \rangle$;
   - Normalizar la primera componente del nuevo nodo $\mathcal{N}_{ij}$. Esto es, aplicar una secuencia (maximal) de pasos interpretativos: $\langle ((\mathcal{Q}[E/v\&\mathcal{B}])\theta; \sigma\theta \rangle \rightarrow_{IS}^{*} \langle (\mathcal{Q}'; \sigma\theta \rangle$. De este modo, obtenemos un nuevo nodo $\mathcal{N}'_{ij} \equiv \langle \mathcal{Q}'; \sigma\theta; inf\{u, v\} \rangle$.
   - Si $\mathcal{Q}' = r \in L$, entonces
     - Si NO EXISTE ningún $\mathcal{V} \in UMBRALES$ tal que $r < \mathcal{V}$:
       - Sea $\mathcal{W} \subset UMBRALES$ el mayor subconjunto (posiblemente vacío) de valores comparables con r tal que $r > \mathcal{V}$ para cada $\mathcal{V} \in \mathcal{W}$;
       - Reemplazar el conjunto $\mathcal{W}$ por $\{r\}$ en $UMBRALES$.
   - En otro caso ($\mathcal{Q}' \neq r \in L$, es decir, el nodo no está completamente evaluado), añadir el nodo $\mathcal{N}'_{ij}$ a la lista $ABIERTO$;
4. Eliminar los nodos $\langle @(r_1, \ldots, r_n, \mathcal{B}_1, \ldots, \mathcal{B}_m); \phi; w \rangle$ en $HOJAS$ verificando que, existe $\mathcal{V} \in UMBRALES$, tal que $w < \mathcal{V}$ ó $\dot{@}(r_1, \ldots, r_n, \top, \ldots, \top) < \mathcal{V}$.

>>>>> **[SALIDA]:** Listas $UMBRALES$ y $HOJAS$.

Como hemos visto, el algoritmo trabaja con cuatro listas:

- $ABIERTO$, que contiene los nodos a desplegar;
- $HOJAS$, que contiene los nodos que satisfacen algún criterio de terminación;
- $UMBRALES$, que almacena un conjunto de nodos completamente evaluados (no comparable entre ellos) que son usados como umbrales.

En líneas generales, sólo permitimos desplegar un nodo (por medio de un paso admisible) cuando se usa una regla con grado de verdad v tal que v no es comparable con

ningún $\mathcal{V} \in UMBRALES$, o $v > \mathcal{V}$ para algún $\mathcal{V} \in UMBRALES$. En otro caso, como consecuencia directa de la Proposición 7.5.1, alcanzaremos un nodo (objetivo) cuya posterior evaluación nunca originaría un grado de verdad mayor o igual $\mathcal{V}$.

La inclusión de un paso de normalización (es decir, una secuencia de pasos de desplegado interpretativo) después de cada paso de desplegado operacional mejora la probabilidad de obtener nodos totalmente evaluados y, por tanto, la posibilidad de refinar dinámica y eficientemente el conjunto de umbrales. De este modo, podemos prescindir de muchos nodos inútiles.

Obsérvese que la lista $HOJAS$ puede gestionarse como una estructura LIFO (pila) o una estructura FIFO (cola), lo que se corresponde (respectivamente) con la generación/recorrido en profundidad o en anchura del árbol de desplegado.

La experiencia nos muestra que, en general, no existen ventajas (con respecto a la eliminación de nodos inútiles) cuando elegimos una estrategia en anchura o una en profundidad. Disponemos de ejemplos en los que la estrategia en anchura tiene mejor comportamiento en comparación con la estrategia en profundidad y viceversa. Además, no existe evidencia que indique si una regla de computación concreta puede puede mejorar la eliminación de nodos inútiles.

Sin embargo, el orden de exploración de las reglas tiene mayor impacto en la supresión de nodos innecesarios. Entendemos que un orden de exploración que seleccione las reglas del programa en función del número de átomos de sus cuerpos, concediendo preferencia a los hechos sobre el resto de reglas, tiene (posiblemente) el mejor comportamiento ya que se generan más rápidamente nodos completamente evaluados. Finalmente, si los conjuntos

$$UMBRALES = \{r_1, \ldots, r_m\} \qquad HOJAS = \{\langle \mathcal{Q}_1; \phi_1; w_1 \rangle, \ldots, \langle \mathcal{Q}_n; \phi_n; w_n \rangle\}$$

dan las listas de umbrales y hojas que devuelve el Algoritmo 7.6.1, el PE-reductante de A en $\mathcal{P}$ es:

$$\langle A \leftarrow @_{sup}\{r_1, \ldots, r_m, \mathcal{Q}_1, \ldots, \mathcal{Q}_n\}; \top \rangle$$

Ejemplo 7.6.2. *Sea $\mathcal{P}$ el programa del Ejemplo 7.3.3 y tómese el (mismo) objetivo $p(a)$. Supongamos ahora un orden de exploración de las reglas en el que las reglas de $\mathcal{P}$ se ensayan en el orden que se relacionan a continuación para el desplegado y un criterio de parada que sólo permita desplegar hasta el nivel* 3.

$\mathcal{R}_1 : \langle p(Y) \leftarrow\ ; \quad 0.6\rangle$

$\mathcal{R}_2 : \langle q(b,a) \leftarrow\ ; \quad 0.9\rangle$

$\mathcal{R}_3 : \langle s(b) \leftarrow\ ; \quad 0.8\rangle$

$\mathcal{R}_4 : \langle p(a) \leftarrow_{\mathtt{L}} q(X,a); \quad 0.7\rangle$

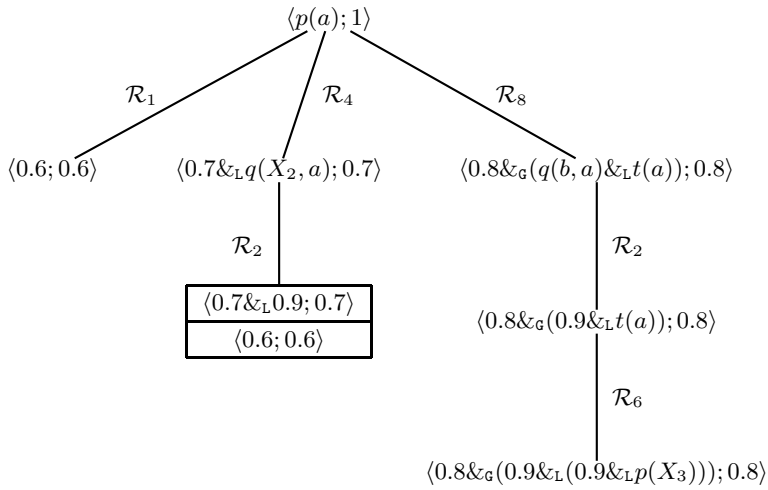
$\mathcal{R}_5 : \langle p(a) \leftarrow_{\mathtt{G}} s(Y); \quad 0.5\rangle$

$\mathcal{R}_6 : \langle t(a) \leftarrow_{\mathtt{L}} p(X); \quad 0.9\rangle$

$\mathcal{R}_7 : \langle s(a) \leftarrow_{\mathtt{G}} t(a); \quad 0.5\rangle$

$\mathcal{R}_8 : \langle p(Y) \leftarrow_{\mathtt{G}} q(b,Y) \&_{\mathtt{L}}\ t(Y); \quad 0.8\rangle$

Después de fijar $\mathcal{V} = 0$ *y constuir el nodo raíz* $\langle p(a); 1\rangle$[7]*, aplicando la secuencia de pasos del Algoritmo 7.6.1, obtenemos el siguiente árbol de desplegado umbralizado de nivel* 3 *para el programa* $\mathcal{P}$ *y el átomo básico* $p(a)$ *(que, para este ejemplo, es independiente de la estrategia de búsqueda usada en su construcción):*



Observemos que, originariamente, el paso de desplegado ejecutado con la regla $\mathcal{R}_1$ *conduce al nodo hoja totalmente evaluado* $\langle 0.6; 0.6\rangle$*. Por tanto, el umbral* $\mathcal{V}$ *se actualiza a* 0.6 *y se evita el paso de desplegado con la regla* $\mathcal{R}_5$*. En el nivel* 2*, el nodo hoja normalizado* $\langle 0.6; 0.6\rangle$ *no altera el umbral* $\mathcal{V}$ *y dado que el grado de verdad* 0.6 *computado no es mayor que* $\mathcal{V}$*, no se añade este nodo al conjunto* $HOJAS$*. En consecuencia, obtenemos un árbol de desplegado mucho más reducido que el del Ejemplo 7.3.7. Finalmente, el Algoritmo 7.6.1 devuelve el conjunto de* $HOJAS$

[7] En rigor, el algoritmo construye el nodo $\langle p(a); id; 1\rangle$, pero, en principio, la segunda componente no será relevante en la construcción de PE-reductantes de átomos básicos, por lo que puede omitirse.

$\{\langle 0.6; 0.6\rangle, \langle 0.8 \&_{\mathtt{G}} (0.9 \&_{\mathtt{L}} (0.9 \&_{\mathtt{L}} p(X_3))); 0.8\rangle\}$*, que nos permite generar un PE-reductante más simple:*

$$\langle p(a) \leftarrow @_{sup}\{0.6, 0.8 \&_{\mathtt{G}} (0.9 \&_{\mathtt{L}} (0.9 \&_{\mathtt{L}} p(X_3)))\};\ 1\rangle.$$

7.7. Un ejemplo comparativo

A continuación mostramos sobre un ejemplo los beneficios alcanzados por nuestras técnicas basadas en umbralización para calcular PE-reductantes y las ventajas de estos con respecto al reductante de Medina *et al.* [2004].

Primeramente, estamos interesados en evidenciar que el programa original no permite computar una respuesta correcta determinada que sí se obtiene mediante el uso de reductantes. En segundo lugar, centramos nuestra atención en comparar el esfuerzo computacional necesario para computar y ejecutar diferentes PE-reductantes, así como sus propios formatos, lo que pone de relieve las principales ventajas de nuestro algoritmo.

Antes de acometer este ejemplo, deseamos introducir un breve comentario acerca de los ejemplos previos con que ilustramos hasta ahora el concepto de reductante. Hemos visto que las técnicas de EP son útiles, en general, en diferentes tareas asociadas a la de ingeniería del software, independientemente de que, en el contexto difuso, los retículos asociados a los programas sean totalmente ordenados o (sólo) parcialmente ordenados.

Por comodidad, en todos los ejemplos previos, especialmente en aquéllos que ilustran la generación de los árboles de desplegado en las Secciones 7.2 y 7.3, hemos usado el retículo totalmente ordenado $([0,1], \leq)$. Sin embargo, como ya hemos adelantado en la introducción de este concepto, los reductantes son realmente interesantes sólo cuando el retículo no es totalmente ordenado. Por este motivo, los ejemplos que siguen a partir de ahora contemplarán tales retículos.

Sea $\mathcal{P}$ el siguiente programa, donde la conectiva & usado en todas las reglas tiene función de verdad definida por $\dot{\&}(x, y) = inf\{x, y\}$, y el retículo $(L, \leq)$ asociado está determinado por el diagrama de Hasse que se indica.

$\mathcal{R}_1 : \langle p(a) \leftarrow q(X,a); \quad \alpha \rangle$

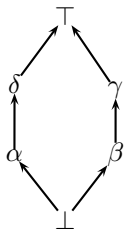
$\mathcal{R}_2 : \langle p(Y) \leftarrow q(X;Y) \& s(X) \& t(Y); \quad \alpha \rangle$

$\mathcal{R}_3 : \langle p(a) \leftarrow s(b) \& \ t(a); \quad \top \rangle$

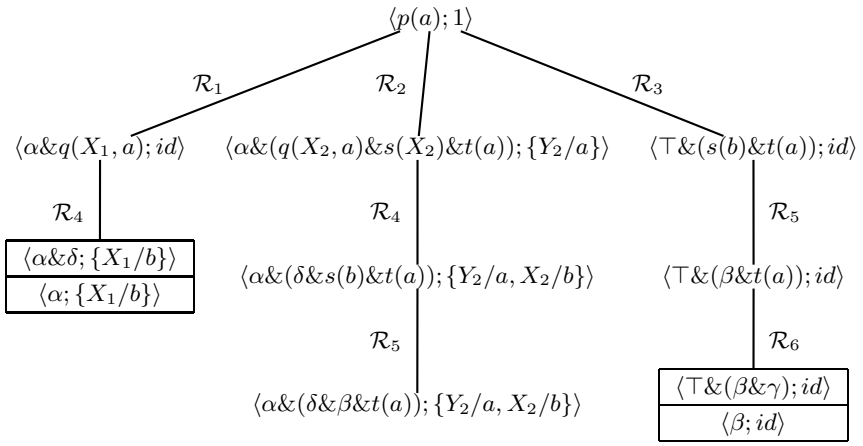
$\mathcal{R}_4 : \langle q(b,a) \leftarrow ; \quad \delta \rangle$

$\mathcal{R}_5 : \langle s(b) \leftarrow ; \quad \beta \rangle$

$\mathcal{R}_6 : \langle t(a) \leftarrow ; \quad \gamma \rangle$



Un árbol de desplegado de nivel 3 para el programa $\mathcal{P}$ y el átomo básico $p(a)$ (para el que evitamos concretar en cada nodo la componente grado de verdad que se obtiene sin más que anotar el correspondiente $inf\{u,v\}$ del algoritmo anterior) es:



A partir de este grafo podemos construir los siguientes PE-reductantes explotando diferentes árboles de nivel 1, nivel 3, o nivel 3 con umbralización (que evita la generación de la rama central mostrada en la figura), respectivamente:

$\mathcal{R} \equiv \ \langle p(a) \leftarrow @_{sup}(\alpha \& (q(X_1,a), \alpha \& q(X_2,a) \& s(X_2) \& t(a)), \top \& (s(b) \& t(a))); \top \rangle$

$\mathcal{R}' \equiv \ \langle p(a) \leftarrow @_{sup}(\alpha, \alpha \& (\delta \& (\beta \& t(a))), \beta); \top \rangle$

$\mathcal{R}'' \equiv \ \langle p(a) \leftarrow @_{sup}(\alpha, \beta); \top \rangle$

Entonces, para el objetivo considerado $p(a)$, podemos concluir las siguientes observaciones:

1. Por la propiedad de corrección de la programación lógica multi-adjunta, dado que $\langle \alpha; id \rangle$ y $\langle \beta; id \rangle$ son respuestas computadas difusas para $\mathcal{P}$ y $p(a)$, también son respuestas correctas. Además, $\langle sup\{\alpha, \beta\}; id \rangle = \langle \top; id \rangle$ es también una respuesta correcta. Sin embargo, $\langle \top; id \rangle$ no puede ser computada en $\mathcal{P}$.

2. Por fortuna, el PE-reductante $\mathcal{R}$ nos permite obtener la respuesta computada difusa $\langle \top; id \rangle$ después de aplicar 10 pasos de computación como sigue: $\langle p(a); id \rangle \rightarrow_{AS}^{R} \langle @_{sup}(\alpha \& q(X_1, a), \alpha \& (q(X_2, a) \& s(X_2) \& t(a)), \top \& (s(b) \& t(a))); id \rangle \rightarrow_{AS/IS}^{*(9)} \langle \top; id \rangle$. Por otra parte, aproximadamente la mitad del esfuerzo computacional es necesario cuando usamos el PE-reductante más simple $\mathcal{R}'$.

3. Finalmente, no sólo $\mathcal{R}''$ es la regla más sencilla, sino que aporta el mejor comportamiento computacional, requiriendo solamente los dos pasos de computación siguientes: $\langle p(a); id \rangle \rightarrow_{AS}^{R''} \langle @_{sup}(\alpha, \beta); id \rangle \rightarrow_{IS} \langle \top; id \rangle$.

7.8. Propiedades formales de los PE-reductantes

Establecemos aquí las propiedades formales de corrección procedural de los PE-reductantes. Como ya hemos visto en el capítulo anterior, es importante remarcar que la noción original de reductante al igual que nuestra definición mejorada de PE-reductante, están referidas a átomos básicos. Por tanto, en cualquier respuesta computada difusa para un átomo básico dado A (incluyendo también aquéllas que se han obtenido mediante cualquier tipo de estos reductantes), la componente sustitución es irrelevante. En esta sección, nos aprovecharemos de este aspecto simplificador cuando definamos y obtengamos formalmente los resultados principales.

Necesitamos hacer uso de los términos PE^k-reductante para referirnos a aquellos PE-reductantes obtenidos a partir del arbol de desplegado de nivel $k, k \geq 1$. Como ya dijimos, el PE^1-reductante coincide con el 1-reductante introducido en la Definición 6.1.2.

En síntesis, demostraremos que el reductante de Medina *et al.* [2004], el PE^1-reductante y el PE^k-reductante son proceduralmente equivalentes y, si $k > 1$, el PE^k-reductante es más eficiente que sus primitivos.

Todos los resultados pueden encontrarse en [Julián *et al.*, 2009].

7.8.1. Corrección procedural y eficiencia

Justificaremos en esta sección la equivalencia procedural entre el reductante clásico, el PE^1-reductante y el PE^k-reductante. Veremos también que el PE^k-reductante, con $k > 1$, es más eficiente que los anteriores. Con este propósito, formalizamos primeramente algunas nociones y resultados previos.

En primer lugar, denotaremos por $\mathcal{FCA}(E)$ el conjunto de respuestas computadas difusas o f.c.a.'s de una expresion dada (objetivo) E. Formalmente, $\mathcal{FCA}(E) = \{\langle r;\theta\rangle \mid \langle E;id\rangle \rightarrow^*_{AS/IS} \langle r;\sigma\rangle, r \in L, \theta = \sigma[\mathcal{V}ar(E)]\}$ que puede generalizarse al conjunto de expresiones $E_1,\ldots,E_n$ del modo $\mathcal{FCA}(E_1,\ldots,E_n) = \mathcal{FCA}(E_1) \cup \cdots \cup \mathcal{FCA}(E_n)$. Además, a fin de dar una medida del esfuerzo computacional necesario para computar este conjunto de f.c.a.'s para una expresión dada E, denotamos por $[\![\mathcal{FCA}]\!](E)$ el número de pasos admisibles/interpretativos diferentes que son necesarios para generar todo el conjunto $\mathcal{FCA}(E)$.

Finalmente, puesto que trabajamos con reductantes para átomos básicos, no será necesario prestar atención a la componente sustitución de una respuesta computada difusa $\langle r;\theta\rangle$ y nos referiremos al valor r como la respuesta computada difusa. En particular, podemos usar sin riesgo una versión simplificada de las nociones de $\mathcal{FCA}$ y $[\![\mathcal{FCA}]\!]$ anteriores. Es decir, podemos considerar que $\mathcal{FCA}(E) = \{r \in L \mid \langle E;id\rangle \rightarrow^*_{AS/IS} \langle r;\sigma\rangle\}$, y $[\![\mathcal{FCA}]\!](E)$ es el número de pasos admisibles/interpretativos diferentes necesarios para generar el conjunto $\mathcal{FCA}(E)$ referido.

Lema 7.8.1. *Dado un árbol de desplegado τ para un programa multi-adjunto $\mathcal{P}$ y un átomo A, sean E_1 y E_2 expresiones (nodos) de τ, tales que existe un paso de derivación de la forma $E_1 \rightarrow_{AS/IS} E_2$. Entonces,*

(1) $\mathcal{FCA}(E_2) \subset \mathcal{FCA}(E_1)$, *y*

(2) $[\![\mathcal{FCA}]\!](E_2) < [\![\mathcal{FCA}]\!](E_1)$.

Demostración. Probamos cada uno de los resultados por separado:

(1) Es suficiente mostrar que, para cada valor $r \in \mathcal{FCA}(E_2)$, se tiene $r \in \mathcal{FCA}(E_1)$. Por hipótesis, existe un paso de derivación $D : [\langle E_1;id\rangle \rightarrow_{AS/IS} \langle E_2;\sigma\rangle]$ y además, como $r \in \mathcal{FCA}(E_2)$, también existe una derivación $D' : [\langle E_2;id\rangle \rightarrow^*_{AS/IS} \langle r;\sigma'\rangle]$. Ahora, componiendo las derivaciones D y D', tenemos la nueva derivación $D'' : [\langle E_1;id\rangle \rightarrow_{AS/IS} \langle E_2;\sigma\rangle \rightarrow^*_{AS/IS} \langle r;\sigma\sigma'\rangle]$, lo que justifica que $r \in \mathcal{FCA}(E_1)$, como deseábamos.

(2) El resultado se obtiene trivialmente sin más que considerar que la derivación previa D'' tiene exactamente un paso más (el primero, asociado a D) que D', esto es $longitud(D') < longitud(D) + longitud(D') = 1 + longitud(D') = longitud(D'')$, de lo que se deduce $[\![\mathcal{FCA}]\!](E_2) < [\![\mathcal{FCA}]\!](E_1)$.

□

La proposición siguiente afirma que, dado un árbol de desplegado de un átomo A (quizás básico) y un programa $\mathcal{P}$, el conjunto de respuestas computadas difusas de un nodo (estado) E es la unión de los conjuntos de respuestas computadas de sus estados sucesores (es decir, de aquellos nodos obtenidos de E después de ejecutar un paso admisible o interpretativo).

Proposición 7.8.2. *Dado un árbol de desplegado τ para un programa $\mathcal{P}$ y un átomo A, conteniendo al menos un nodo no raíz, sea E una expresión (nodo) de τ y sea $\mathcal{U}(E) = \{E' \mid E \rightarrow_{AS/IS} E'\}$ el conjunto de sucesores de E. Entonces, $\mathcal{FCA}(E) = \bigcup_{E_i \in \mathcal{U}(E)} \mathcal{FCA}(E_i)$.*

Demostración. Si E no es un valor de L y para un cierto índice i existe un paso $E \rightarrow_{AS/IS} E_i$, por el resultado (1) del Lema 7.8.1, se tiene que $\mathcal{FCA}(E_i) \subset \mathcal{FCA}(E)$ y, por la definición de unión, concluimos

$$\bigcup_{E_i \in \mathcal{U}(E)} \mathcal{FCA}(E_i) \subset \mathcal{FCA}(E)$$

Recíprocamente, si $r \in \mathcal{FCA}(E)$, demostraremos que r es una f.c.a. del conjunto $\bigcup_{E_i \in \mathcal{U}(E)} \mathcal{FCA}(E_i)$. En efecto, dado $r \in \mathcal{FCA}(E)$, existe una derivación $E \rightarrow^n_{AS/IS} r$, para la que distinguimos los dos casos siguientes:

- si $n = 0$, entonces E es un elemento de L y se satisface trivialmente el resultado.
- si $n > 0$, sea E_j la expresión verificando $\langle E; id\rangle \rightarrow_{AS/IS} \langle E_j; \sigma\rangle \rightarrow^{n-1}_{AS/IS} \langle r; \theta\rangle$. Entonces, $r \in \mathcal{FCA}(E_j) \subset \bigcup_{E_i \in \mathcal{U}(E)} \mathcal{FCA}(E_i)$, como deseábamos probar.

□

Llegados aquí, estamos en condiciones de probar el primer resultado importante de esta sección, que es totalmente análogo al Teorema 6.1.4, pero ocupándonos ahora del carácter procedural en vez del semántico referido. Más exactamente, veamos que el reductante debido a Medina *et al.* [2004] y el PE^1-reductante son proceduralmente equivalentes.

Teorema 7.8.3. *Sea $\mathcal{P}$ un programa multi-adjunto, A un átomo básico y $\mathcal{R} \equiv \langle A \leftarrow @(\mathcal{B}_1, \ldots, \mathcal{B}_n)\theta; \top\rangle$ el reductante para A en $\mathcal{P}$, donde $\theta = \theta_1 \cdots \theta_n$ y cada sustitución θ_i es unificador de A y la cabeza de una regla $\langle H_i \leftarrow_i \mathcal{B}_i; v_i\rangle$. El PE^1-reductante $\mathcal{R}' \equiv \langle A \leftarrow @_{sup}(\mathcal{D}_1, \ldots, \mathcal{D}_n); \top\rangle$ (donde $\mathcal{D}_i \equiv v_i \&_i \mathcal{B}_i\theta$, $1 \leq i \leq n$), obtenido a partir de un árbol de desplegado de nivel uno para $\mathcal{P}$ y A, es proceduralmente equivalente al reductante $\mathcal{R}$.*

Demostración. Para lograr la equivalencia procedural entre ambas nociones de reductantes es suficiente demostrar la igualdad

$$\mathcal{FCA}(@(\mathcal{B}_1,\ldots,\mathcal{B}_n)\theta) = \mathcal{FCA}(@_{sup}(\mathcal{D}_1,\ldots,\mathcal{D}_n))$$

Por la Proposición 7.8.2, si ejecutamos en primer lugar todos los pasos admisibles con el PE^1-reductante antes de ejecutar los interpretativos, obtenemos las mismas respuestas computadas admisibles con ambos reductantes (para el átomo A). Además, por la corrección total fuerte de la transformación de desplegado interpretativo (veáse el Teorema 5.5.1), la fase interpretativa considerada por Medina *et al.* [2004] conduce a las mismas respuestas computadas difusas que la ejecución de todos los pasos interpretativos definidos en la Sección 3.1.2. □

Hasta el momento, hemos relacionado la noción de reductante clásico con la de PE^1-reductante, estableciendo resultados equivalentes a nivel semántico y procedural (Teoremas 6.1.4 y 7.8.3, respectivamente). A partir de ahora, nos interesa abordar la equivalencia entre el PE^1-reductante y el PE^k-reductante (Teorema 7.8.4). Además, mostraremos que a medida que el PE-reductante es parcialmente evaluado, las ganancias en eficiencia son más relevantes en tiempo de ejecución (Teorema 7.8.5).

El resultado que sigue muestra, en una formulación sencilla, que las respuestas computadas difusas o f.c.a.'s de las expresiones presentes en las hojas de un árbol de desplegado asociado al PE^1-reductante, para un átomo básico A en un programa $\mathcal{P}$, coincide con las f.c.a.'s de las expresiones de las hojas de un árbol de desplegado asociado al PE^k-reductante. Por tanto, el PE^1-reductante y el PE^k-reductante son proceduralmente equivalentes.

Teorema 7.8.4 (Corrección Procedural)**.** *Si $\mathcal{R}^1 \equiv \langle A\leftarrow @_{sup}(\mathcal{D}_1,\ldots,\mathcal{D}_n);\ \top\rangle$ y $\mathcal{R}^k \equiv \langle A\leftarrow @_{sup}(\mathcal{D}'_1,\ldots,\mathcal{D}'_m);\top\rangle$ son, respectivamente, el PE^1-reductante y el PE^k-reductante para un átomo básico A en un programa $\mathcal{P}$, entonces, $\mathcal{R}^1$ y $\mathcal{R}^k$ son proceduralmente equivalentes, es decir, $\mathcal{FCA}(\mathcal{D}_1,\ldots,\mathcal{D}_n) = \mathcal{FCA}(\mathcal{D}'_1,\ldots,\mathcal{D}'_m)$.*

Demostración. Por la Proposición 7.8.2 se tiene $\mathcal{FCA}(A) = \mathcal{FCA}(\mathcal{D}_1,\ldots,\mathcal{D}_n) = \mathcal{FCA}(\mathcal{D}_1)\cup\cdots\cup\mathcal{FCA}(\mathcal{D}_n)$ y, además, cada $\mathcal{FCA}(\mathcal{D}_i) = \mathcal{FCA}(\mathcal{B}_{1i},\ldots,\mathcal{B}_{ri})$, donde $\mathcal{B}_{1i}\ ,\ldots,\mathcal{B}_{ri}$ son todos los sucesores de $\mathcal{D}_i$, que son fórmulas del cuerpo del PE^2-reductante. Reiterando el proceso hasta alcanzar los nodos de nivel k, obtenemos $\mathcal{FCA}(A) = \mathcal{FCA}(\mathcal{D}'_1,\ldots,\mathcal{D}'_m)$ y se verifica la tesis. □

Estamos en condiciones de introducir el siguiente resultado que complementa el

anterior garantizando la ganancia en eficiencia alcanzada con los PE^k-reductantes en comparación con los PE^1-reductantes.

Teorema 7.8.5 (Eficiencia Procedural)**.** *Si* $\mathcal{R}^1 \equiv \langle A \leftarrow @_{sup}(\mathcal{D}_1, \ldots, \mathcal{D}_n); \top \rangle$ *y* $\mathcal{R}^k \equiv \langle A \leftarrow @_{sup}(\mathcal{D}'_1, \ldots, \mathcal{D}'_m); \top \rangle$ *son, respectivamente, el* PE^1*-reductante y el* PE^k*-reductante de un átomo* A *en un programa* $\mathcal{P}$*, entonces,* $\mathcal{R}^k$ *es más eficiente que* $\mathcal{R}^1$ *cuando* $k{>}1$*, es decir,* $[\![\mathcal{FCA}]\!](\mathcal{D}_1, \ldots, \mathcal{D}_n) > [\![\mathcal{FCA}]\!](\mathcal{D}'_1, \ldots, \mathcal{D}'_m)$.

Demostración. Esta prueba es totalmente análoga a la del Teorema 7.8.4, pero explotando ahora también la tesis (2) del Lema 7.8.1. □

Para terminar esta sección, en el siguiente corolario relacionamos a nivel procedural (como consecuencia directa de los Teoremas 7.8.3 y 7.8.4) la noción de reductante clásico y nuestra definición mejorada de PE^k-reductante.

Corolario 7.8.6. *Sean* $\mathcal{R} \equiv \langle A \leftarrow @(\mathcal{B}_1, \ldots, \mathcal{B}_n); \top \rangle$ *y* $\mathcal{R}^k \equiv \langle A \leftarrow @_{sup}(\mathcal{D}_1, \ldots, \mathcal{D}_m); \top \rangle$*, respectivamente, un reductante conforme a la Definición 6.1.1 y un* PE^k*-reductante de un átomo básico* A *en* $\mathcal{P}$*. Entonces* $\mathcal{R}$ *y* $\mathcal{R}^k$ *son proceduralmente equivalentes, es decir,* $\mathcal{FCA}(@(\mathcal{B}_1, \ldots, \mathcal{B}_n)) = \mathcal{FCA}(@_{sup}(\mathcal{D}_1, \ldots, \mathcal{D}_m))$.

7.9. Conclusiones

Las aportaciones de este capítulo pueden resumirse del siguiente modo.

- Hemos definido por primera vez un marco de evaluación parcial para programas lógicos multi-adjuntos.
- Hemos mostrado la habilidad de esta transformación, en el contexto multi-adjunto, para especializar programas tal y como se ha hecho tradicionalmente en otros contextos declarativos.
- Hemos mostrado asimismo, sobre algunos ejemplos, cómo esta especialización aporta programas más eficientes.
- Hemos propuesto un método novedoso para el cálculo de PE-reductantes mediante técnicas de evaluación parcial.
- Hemos propuesto, asimismo, obtener el PE-reductante a partir de un árbol de desplegado umbralizado a fin de reducir la expresión de éste y beneficiarnos de otras ventajas añadidas en la computación.
- Hemos diseñado un algoritmo eficiente para el cálculo de PE-reductantes, que reducirá drásticamente el tamaño de los árboles de desplegado que aportan dichos reductantes usando técnicas de evaluación parcial con umbralización.
- Hemos mostrado que la construcción (mediante este algoritmo) de tales árboles de desplegado exige menos recursos computacionales y simplifica la expresión del PE-reductante que aportan.
- Hemos contrastado que los PE-reductantes que produce el algoritmo reducen pasos de derivación en las secuencias de derivación en que intervienen.
- Hemos probado la equivalencia semántica y procedural entre el PE-reductante, el 1-reductante y el reductante debido a Medina *et al.* [2004], a la vez que contrastamos la mayor eficiencia del PE-reductante.
- Las equivalencias anteriores, junto con la comodidad (ya referida anteriormente) que supone obtener el PE-reductante a partir de técnicas de evaluación parcial umbralizada, reflejan las ventajas del concepto de PE-reductante respecto del original.

- El uso de estos PE-reductantes permite obtener programas residuales más eficientes que los originales, en los que se rebaja la longitud de las derivaciones admisibles.

Capítulo 8

Conclusiones

El objetivo central que nos hemos propuesto en esta tesis es la introducción de un conjunto de transformaciones (basadas en desplegado) para la optimización de programas lógicos difusos. Estas técnicas de transformación de programas, clásicas en programación declarativa (lógica, funcional y lógico funcional), se han adaptado para distintos lenguajes de la programación lógica difusa, y se han tratado con especial incidencia en el marco multi-adjunto.

La programación lógica multi-adjunta ha sido introducida en [Medina *et al.*, 2001d,c] como una generalización de la programación lógica monótona y residuada considerada en [Damásio y Moniz-Pereira, 2000; Dekhtyar y Subrahmanian, 2000; Damásio y Moniz-Pereira, 2001b, 2002, 2004] que, a su vez, extiende (entre otros) los programas lógicos probabilísticos híbridos de Dekhtyar y Subrahmanian [2000], los programas de bases de datos deductivas probabilísticas de Lakshmanan y Sadri [2001], los programas lógicos probabilísticos ordinarios de Lukasiewicz [2001b] y los programas del marco de deducción cuantitativa de van Emden [1986] (que, por su parte, extienden a los programas de Dubois *et al.* [1991b]).

Por tanto, dado que la programación lógica multi-adjunta representa un marco muy general en el que se pueden subsumir otros lenguajes lógicos difusos, resulta muy atractiva para formular reglas de transformación de programas que, una vez caracterizadas en este ámbito, podrían ser trasladadas a estos lenguajes (instancias del lenguaje multi-adjunto). Este hecho garantizará la relevancia y generalidad de nuestros resultados, y facilitará la difusión de los mismos.

Además, este lenguaje posee un alto nivel de expresividad y dispone de una

semántica procedural clara, lo que resulta crucial para definir las transformaciones estudiadas (en particular, para una definición formal de reglas de desplegado y de técnicas de evaluación parcial, la semántica procedural debe estar formalizada en términos de un sistema de transición de estados), que es la tarea central de esta tesis.

Nuestras principales aportaciones han sido las siguientes:

1. **Lenguajes:**

   Entre la variedad de lenguajes de programación lógica difusa existentes en la literatura, el descrito en [Vojtáš y Paulík, 1996], al que hemos llamado f-Prolog, es una de las referencias clásicas y lo hemos visto adecuado para definir el concepto de desplegado de programas lógicos difusos [Julián *et al.*, 2004b,c].

   El lenguaje f-Prolog ha sido mejorado en esta tesis con marcas etiquetadas para obtener lf-Prolog, sobre el que hemos podido formular cómodamente la transformación de desplegado, al permitir la codificación de los programas desplegados.

   Además, extendimos ambos lenguajes permitiendo una interpretación más flexible de las conectivas y admitiendo diferentes lógicas en un mismo programa; de este modo, es posible interpretar cada cláusula con una lógica diferente, lo que enriquece notablemente su potencia expresiva. Los lenguajes resultantes son ef-Prolog y lef-Prolog [Julián *et al.*, 2004a, 2005c]. Y merece ser destacado el hecho de que estos lenguajes extendidos conservan las misma semántica operacional y declarativa que los primitivos.

   Asimismo, hemos adaptado el concepto de *regla de computación* al marco difuso, y hemos demostrado la independencia de la regla de computación tanto para programas y objetivos lf-Prolog (véase [Julián *et al.*, 2004b,c]) como para los correspondientes del lenguaje lef-Prolog (como puede verse en [Julián *et al.*, 2004a, 2005c]) que extiende a nuestro entorno el resultado obtenido por Lloyd [1987].

   Centrándonos en el lenguaje multi-adjunto, hemos implementado, en el grupo DEC-TAU, un prototipo de intérprete/compilador para traducir programas lógicos multi-adjuntos a código Prolog, que pretende ser una plataforma de gran utilidad para la optimización de programas difusos. Esta herramienta permite, ya en la actualidad, compilar, ejecutar y manipular programas lógicos difusos, así como generar y visualizar árboles de desplegado.

2. **Semánticas:**

   Hemos revisado todas las semánticas conocidas al día de hoy para la programación multi-adjunta, con aportaciones significativas para alguna de ellas y hemos desarrollado los resultados que garantizan buenas relaciones entre ellas.

   Clarificamos la semántica procedural, añadiendo a la fase operacional (debida a los autores del lenguaje), el diseño de la fase interpretativa, tal como se recoge en [Julián *et al.*, 2006c], entendida como un sistema de transición de estados, tarea que resultó imprescindible para la formalización del desplegado interpretativo en el marco multi-adjunto. Esta reformulación de la fase interpretativa permite también abordar un análisis del coste computacional de estos programas multi-adjuntos, como precisaremos a continuación.

   Demostramos la independencia de la regla de computación, extendiendo para programas multi-adjuntos (al igual que para programas lef-Prolog, según se ha observado en el apartado anterior) el resultado bien conocido de la programación lógica clásica por el que, tal como se justifica en [Julián *et al.*, 2005a], a la hora de ejecutar pasos de computación admisible, no importa la función se selección de subexpresiones que se emplee.

   Posteriormente, obtenemos por primera vez la noción semántica de modelo mínimo difuso concebido éste como el ínfimo del conjunto de interpretaciones que son modelo del programa. Este concepto reproduce, como puede verse en [Julián *et al.*, 2009], la concepción clásica de modelo mínimo de la programación lógica pura, es equivalente a la semántica de punto fijo y presenta también relaciones con la semántica procedural entendida como el conjunto de respuestas computadas difusas. Además, este modelo mínimo permite caracterizar las respuestas correctas mejorando así el caso de la programación lógica pura.

   Además, incluimos una demostración original de la corrección de la semántica procedural de la programación multi-adjunta. Y si el resultado es una réplica del caso clásico (de la programación lógica pura), el esquema de demostración sí contempla novedades muy significativas con respecto a la primitiva.

   Hemos considerado todas las relaciones entre las semánticas conocidas para la programación lógica multi-adjunta, probando en particular la equivalencia de nuestra semántica declarativa de modelo mínimo difuso y la semántica de punto fijo debida a Medina *et al.* [2004].

   Por otra parte, hemos definido medidas de coste y hemos aportado resultados

acerca del coste de ejecución de estos programas. Mostramos cómo la opción de estimar el esfuerzo computacional que se requiere para ejecutar un objetivo contanto el número de pasos de derivación, habitual en la programación declarativa, no es apropiado en la programación lógica multi-adjunta cuando interviene la fase interpretativa.

La dificultad aparece cuando aparecen agregadores en el cuerpo de las reglas de programa cuyas definiciones invocan a otros agregadores. Es evidente que la evaluación de este tipo de enlaces consume recursos computacionales en tiempo de ejecución que no son estimados al contar los pasos interpretativos.

Por tanto, una medida ingenua que cuente los pasos interpretativos no evalúa estos recursos en el coste final porque no se contabilizan explícitamente dichos agregadores al dar pasos interpretativos. Por nuestra parte, hemos resuelto esta dificultad asignando pesos a los agregadores que son acordes con su complejidad.

Hemos definido, como puede verse en [Julián *et al.*, 2007c], [Julián *et al.*, 2007a], una medida de coste computacional para los programas lógicos multi-adjuntos, basada en contabilizar el número de conectivas y operadores primitivos que aparecen dentro de la definición de los agregadores que son evaluados en cada paso (interpretativo) de una derivación dada.

Asimismo, en aplicación de este criterio de coste, hemos contrastado (véanse de nuevo [Julián *et al.*, 2007c], [Julián *et al.*, 2007a]) la eficiencia de dos nociones semánticamente equivalentes de reductantes (la original introducida por Medina *et al.* [2004] y nuestra versión refinada de 1-reductante.)

Por último, observamos que también en este ámbito ha sido imprescindible el diseño, contemplado en la Sección 3.1.2, de la fase interpretativa como un sistema de transición de estados.

3. **Transformaciones:**

   Hemos adaptado al contexto difuso (para varios lenguajes), por primera vez en la literatura, la noción clásica de desplegado, heredando la simplicidad y la potencia computacional de experiencias precedentes en entornos no difusos.

   Primeramente, hemos definido el desplegado difuso para el lenguaje f-Prolog de Vojtáš y Paulík [1996] (tal como se recoge en [Julián *et al.*, 2004b,c]) y para

el lenguaje ef-Prolog, así como el reemplazamiento de t-norma de programas ef-Prolog [Julián *et al.*, 2004a, 2005c].

Asimismo, hemos introducido para el lenguaje lef-Prolog las reglas de reemplazamiento de t-norma, que aportan nuevas ventajas a esta tranformación a la hora de computar los grados de verdad. Esta regla, introducida en [Julián *et al.*, 2005c], puede verse como un precedente de la de desplegado interpretativo que referimos posteriormente para el lenguaje multi-adjunto.

Además, hemos abordado el estudio de la transformación de desplegado en el marco de la programación lógica multi-adjunta. En concreto, hemos contemplado para el lenguaje multi-adjunto, dos tipos de desplegado muy interrelacionados: el operacional (primeramente introducido en [Julián *et al.*, 2005a]) y el interpretativo (introducido por vez primera en [Julián *et al.*, 2006c]).

Este último, acelerará la evaluación de los grados de verdad en la fase interpretativa, tal como hemos justificado en [Julián *et al.*, 2006c]. Para su formulación ha sido imprescindible la formalización de esta fase como un sistema de transición de estados que, además, evita el uso de conceptos semánticos (los que originariamente definen esta fase en [Medina *et al.*, 2004] y que nosotros también respetamos en [Julián *et al.*, 2005a]).

Por último, hemos destacado el papel fundamental que juega la transformación de desplegado (para la programación lógica multi-adjunta) tanto para producir evaluadores parciales y mejorar el cálculo de reductantes, tal como se trató en los Capítulos 6 y 7, como para construir semánticas por desplegado para este lenguaje (lo que se contempla posteriormente como trabajo futuro).

4. **Propiedades de corrección del desplegado difuso:**

   Hemos obtenido los mejores resultados de corrección y eficiencia, que pueden esperarse de las transformaciones de desplegado difuso planteadas anteriormente para las distintas variaciones difusas de Prolog, con especial énfasis en el marco multi-adjunto.

   Estos resultados se obtienen por primera vez en el contexto difuso y los correspondientes al lenguaje lef-Prolog generalizan a los que obtuvimos también para lenguajes más primitivos (f-Prolog y lf-Prolog, de Vojtáš).

   En esencia, hemos justificado que, desde el punto de vista teórico, los programas transformados obtienen las mismas respuestas computadas difusas que el

original; y, desde el punto de vista práctico, se obtiene mejora en eficiencia cuando ejecutamos los programas residuales, puesto que se reduce el número de pasos de computación necesarios para resolver un objetivo.

Debemos precisar que los resultados obtenidos para el desplegado de programas multi-adjuntos no extienden automáticamente a los correspondientes del lenguaje lef-Prolog dado que estos lenguajes poseen distinta semántica procedural, aparte de otros rasgos que los diferencia a nivel expresivo y procedural.

En cuanto al lenguaje lef-Prolog, hemos justificado la corrección total fuerte (véase [Julián *et al.*, 2004a], [Julián *et al.*, 2004b] y [Julián *et al.*, 2005c]). Por lo que se refiere al lenguaje lenguaje multi-adjunto, hemos demostrado las propiedades de corrección del desplegado operacional, del desplegado interpretativo y, finalmente, los resultados de corrección del sistema de transformación determinado por la combinación de ambos tipos de desplegado. Estas propiedades del desplegado operacional se recogen en [Julián *et al.*, 2005a], y los relativos al desplegado interpretativo y al sistema de transformación en [Julián *et al.*, 2005b] y en [Julián *et al.*, 2006c].

En cuanto a la eficiencia, hemos probado que las secuencias de transformación pueden ser dirigidas de manera arbitraria[1], puesto que cualquier paso de transformación basada en desplegado difuso o reemplazamiento de t-norma siempre produce una mejora en los programas transformados.

Esto constrasta con otras reglas de transformación, como la introducción de definición o el plegado, que pueden degradar la eficiencia de los programas si no se usan "estrategias de transformación" apropiadas para orientar la secuencia de transformación.

Finalmente, es importante destacar que los resultados obtenidos pueden tomarse como el punto de partida para la optimización de programas lógicos difusos y constituyen el primer paso en la construcción de un sistema global de plegado/desplegado (incluyendo más reglas de transformación y estrategias) para la optimización de esta clase de programas.

5. **Reductantes y Evaluación Parcial:**

   Los reductantes son una herramienta introducida en el contexto de la programación lógica generalizada con anotaciones (véase [Kifer y Subrahmanian,

[1] Sin necesidad de introducir heurísticas más refinadas (composición, tupling, etc.)

1992]) para demostrar propiedades de corrección y completitud, y adaptado recientemente al marco multi-adjunto (véase [Medina *et al.*, 2001c]), para tratar el problema de incompletitud que se presenta cuando se trabaja en retículos asociados (sólo) parcialmente ordenados. Un programa lógico multi-adjunto, interpretado en un retículo completo, precisa contener estas reglas (los reductantes) para que quede garantizada su completitud (aproximada).

Por nuestra parte, hemos introducido (como puede verse en [Julián *et al.*, 2006b], [Julián *et al.*, 2009]) un nuevo concepto de reductante con pequeñas diferencias sintácticas con el reductante de Medina *et al.* [2004] –al que llamamos 1-reductante–, pero que posee la ventaja de que puede definirse y extenderse (como referimos a continuación) usando conceptos de evaluación parcial que facilitarán notablemente su cálculo.

Hemos probado que este reductante y el original aportan la misma respuesta computada difusa (y con la misma eficiencia) para un objetivo básico A en un programa $\mathcal{P}$ y, por tanto, son proceduralmente equivalentes; además, hemos justificado la equivalencia semántica de ambos reductantes (véase de nuevo [Julián *et al.*, 2007c], [Julián *et al.*, 2009]).

Hemos adaptado (en [Julián *et al.*, 2009]) el concepto clásico de EP al marco de la programación lógica multi-adjunta partiendo de las técnicas conocidas en el campo de la dedución parcial de programas lógicos puros [Gallagher, 1993; Komorowski, 1982; Lloyd y Shepherdson, 1991; Pettorossi y Proietti, 1994], pero haciendo uso de la regla de desplegado difuso que desarrollamos en [Julián *et al.*, 2005a,b] para esta clase de programas, a fin de obtener una versión optimizada (especializada) del programa original que se ejecuta más eficientemente.

Hemos propuesto (también en [Julián *et al.*, 2009]) un método para calcular reductantes usando estas técnicas de EP, lo que supone una aplicación muy original de las mismas. La idea consiste en relacionar el concepto de EP de un átomo en un programa lógico multi-adjunto con la construcción de un reductante para dicho átomo, lo que nos permite diseñar un cálculo eficiente de reductantes. De este modo, logramos rebajar el impacto negativo de la incorporación de reductantes a un programa lógico multi-adjunto, mediante un cálculo previo en el que los reductantes son parcialmente evaluados antes de añadirlos al programa final: puesto que la fase de EP produce un conjunto refinado de reductantes, el esfuerzo computacional realizado (sólo una vez) en

tiempo de generación es evitado (muchas veces) en tiempo de ejecución.

Surge así el concepto de PE-reductante que generaliza y mejora el reductante de Medina *et al.* [2004] y 1-reductante que habíamos introducido previamente.

Además, hemos obtenido estos PE-reductantes haciendo uso de técnicas de evaluación parcial con umbralización, lo que nos permite simplificar su expresión y reducir las secuencias de derivación en las que intervienen (véanse [Julián *et al.*, 2006a, 2007b], a tal efecto).

Hemos diseñado un algoritmo eficiente para computar PE-reductantes usando técnicas de evaluación parcial basadas en desplegado con un conjunto de umbrales dinámico.

Hemos considerado equivalencias procedurales entre el PE-reductante, el 1-reductante y el reductante debido a Medina *et al.* [2004] (véase [Julián *et al.*, 2007c], [Julián *et al.*, 2009]). Los resultados obtenidos, junto con la comodidad (ya referida) que supone obtener el PE-reductante a partir de técnicas de evaluación parcial umbralizada, muestran las ventajas del concepto de PE-reductante respecto del primitivo. En particular, el uso de PE-reductantes permite trabajar con programas residuales más eficientes que los originales, en los que se rebaja la longitud de las derivaciones admisibles.

Por último, introducimos el concepto de *compleción* de un programa (programa equivalente –en cierto sentido– al dado, completo y formado por reductantes), para evidenciar el uso de reductantes. Además, definimos una nueva semántica operacional para la programación lógica multi-adjunta que evita el inconveniente de trabajar en la práctica con programas infinitos.

Capítulo 9

Trabajo futuro

El desarrollo de la presente tesis doctoral ha supuesto de forma colateral la apertura de una serie de líneas de investigación en nuestro grupo que, en términos generales, pretenden aúnar esfuerzos en cuanto al diseño, implementación y estandarización de los lenguajes lógicos difusos. Centrándonos en el marco multi-adjunto, algunas de estas líneas ya están en marcha o se iniciarán de forma inminente, y su evolución posterior puede entenderse como trabajo futuro de esta tesis. A modo de resumen, estos son los principales frentes temáticos que pretenden continuar los desarrollos descritos en esta memoria:

1. **Semánticas por desplegado:**

   A un nivel teórico, el uso de la regla de desplegado como medio para caracterizar la semántica declarativa de los programas lógicos usando técnicas puramente operacionales, aparece por primera vez descrito en [Levi y Mancarella, 1988].

   En dicho texto, se evidencia el hecho de que la semántica declarativa de un programa lógico, definida de forma estándar como el modelo mínimo de Herbrand del programa, es incapaz de capturar el observable que mejor refleja la esencia operacional de una computación lógica: las respuestas computadas.

   A pesar de que se han descrito conceptos más potentes y ambiciosos de semánticas declarativas (la así llamada s-semántica de Falaschi [1988]; Bossi *et al.* [1994]) capaces de reflejar el comportamiento operacional asociado al observable de las respuestas computadas, es en [Levi y Mancarella, 1988] donde por

primera vez se presenta la idea novedosa de concebir este tipo de semántica como un programa transformado obtenido por una secuencia (posiblemente infinita) de desplegados.

Dicho programa desplegado estaría compuesto únicamente por cláusulas unitarias que se correspondería con la semántica declarativa deseada. Más aún: la semántica por desplegado así construida, puede verse como un programa donde cualquier objetivo puede ser "ejecutado" haciendo uso exclusivamente de unificación sintáctica (sin necesidad de recurrir a la resolución-SLD).

En [Bossi *et al.*, 1994] se muestra que las transformaciones basadas o *guiadas* por la semántica de un programa lógico $\mathcal{P}$, entre las que se incluye con especial relevancia el desplegado, suponen una vía alternativa para caracterizar la s-semántica asociada al observable de respuestas computadas de $\mathcal{P}$.

De forma similar a lo que ocurre con la semántica por punto fijo, esta formalización se revela a medio camino entre la semántica declarativa y la operacional de un programa lógico.

Sirve no sólo para mostrar de forma sencilla la equivalencia entre éstas, sino también para formalizar distintos tipos de semánticas composicionales que pueden aplicarse incluso a programas abiertos (donde el conjunto de cláusulas que definen un mismo predicado puede estar distribuido en módulos independientes), lo que revierte en distintos tipos de modularidad y paralelismo potencial de este tipo de programas.

Como acabamos de decir, la semántica por desplegado de un programa lógico $\mathcal{P}$ supone una forma alternativa de describir el significado del mismo, al tiempo que disfruta de dos importantes propiedades (la primera de carácter semántico y la segunda de estilo más procedural) que podemos resumir así:

a) la semántica por desplegado de $\mathcal{P}$ es capaz de reflejar el comportamiento operacional de $\mathcal{P}$ asociado al observable de las respuestas computadas, y

b) la semántica por desplegado de $\mathcal{P}$ es otro programa (transformado) $\mathcal{P}'$ compuesto únicamente por hechos (cláusulas unitarias o sin cuerpo) sobre el que cualquier objetivo puede "evaluarse" usando unificación sintáctica, obteniéndose los mismos resultados que usando la resolución-SLD sobre el programa original $\mathcal{P}$.

Existen semánticas por desplegado para programas lógicos [Levi y Mancarella,

1988] y lógico funcionales [Moreno, 2000]. Nosotros abordaremos, en el futuro, la obtención de semánticas por desplegado para programas lógicos difusos, en concreto para el lenguaje multi-adjunto de Medina *et al.* [2004]. Aunque la mayoría de los lenguajes lógicos difusos disponen de semánticas de punto fijo y basadas en el modelo mínimo de Herbrand, estas caracterizaciones suelen ser básicas (*ground*), en el sentido de que no contemplan la presencia de variables en sus modelos. Nosotros nos planteamos superar esta barrera mediante la definición de semánticas más ricas (por ejemplo, en la línea de la semántica de respuestas computadas, o s-semánticas [Falaschi *et al.*, 1989, 1993; Bossi *et al.*, 1994]): en el caso concreto del marco multi-adjunto, pensamos que es factible reutilizar buena parte de nuestra experiencia en la definición de reglas difusas de desplegado para obtener una caracterización semántica más potente que la basada en la obtención del conjunto de éxitos básicos.

En efecto, nuestro objetivo es definir una semántica por desplegado (y estudiar las propiedades formales de corrección, completitud), que extienda la aproximación clásica para dar cuenta de los grados de verdad y que nos permita simular nuestra semántica declarativa por modelo mínimo (diseñada en [Julián *et al.*, 2009] y recogida en el Capítulo 3) y la semántica de punto fijo descrita por Medina *et al.* [2004], para el lenguaje multi-adjunto. Esta experiencia nos permitirá, además, valorar la posibilidad de extender el diseño de esta semántica a otros lenguajes difusos que se encuentren suficientemente estandarizados.

2. **Especialización por evaluación parcial:**

   Ya hemos visto en el Capítulo 7, cómo es posible reutilizar técnicas de evaluación parcial (EP) para definir un refinamiento de la noción clásica de reductante, el denominado *PE-reductante*, que permite mejorar las propiedades de completitud del marco multi-adjunto sin que se resienta la eficiencia en la ejecución de los programas. Es importante destacar una vez más que nunca antes se había concebido la EP como un medio capaz de alcanzar unas metas tan novedosos como las nuestras. El algoritmo de contrucción de PE-reductantes descrito en [Julián *et al.*, 2009], es pues un método efectivo y eficiente para el cómputo de reductantes que parte de una primera adaptación al marco multi-adjunto de unos cuantos conceptos básicos de EP provenientes de la programación lógica pura. Pero poco más que la construcción de árboles de desplegado (aún introduciendo elementos originales, como la poda de ramas inútiles me-

diante el uso de un conjunto dinámico de umbrales, propuesta en [Julián *et al.*, 2006a, 2007b]) ha sido necesario incorporar en nuestros desarrollos para cubrir nuestro principal propósito de generar PE-reductantes.

Sin embargo, cuando el objetivo perseguido es el clásico de la especialización de programas, entonces las técnicas de evaluación parcial deben reforzarse en muchos otros aspectos como son nuevos análisis de los árboles de desplegado para generar resultantes, criterios de terminación local y global, etc., que pasamos a comentar a continuación. Es importante tener en cuenta que algunas de estas acciones pueden a su vez arrojar resultados reutilizables en revisiones futuras de los procesos de construcción de PE-reductantes, lo que en el mejor de los casos permitirá una interesante realimentación de las dos aplicaciones fundamentales de la EP que nosotros abordamos en el marco multiadjunto.

Así pues, si en términos clásicos la EP persigue la *especialización* automática de programas declarativos con respecto a parte de sus datos de entrada mediante el cálculo de *resultantes*, en nuestro marco difuso estamos en disposición de acometer esta tarea enlazando con la generación de los árboles de desplegado (posiblemente umbralizados) que hemos descrito en esta tesis para construir PE-reductantes. A partir de aquí, es posible avanzar en los siguientes aspectos:

a) En primer lugar, es necesario superar la restricción de evaluar separadamente átomos básicos que imponemos actualmente a la hora de generar PE-reductantes. Para conseguir mejores grados de especialización, interesa evaluar parcialmente objetivos complejos que contengan varios átomos no necesariamente básicos combinados con agregadores, tal y como se ha hecho en el marco de la deducción parcial conjuntiva [Glück *et al.*, 1996; Leuschel *et al.*, 1996] para la especialización de conjunciones de átomos. Esta misma línea será fundamental para diseñar PE-reductantes referidos a átomos no necesariamente básicos (algunos avances preliminares pueden encontrarse en [Morcillo y Moreno, 2009c]).

b) También es preciso introducir estrategias más refinadas de *control local*, como los métodos basados en (quasi-)órdenes bien-fundados [Albert *et al.*, 1998], en vez de fijar *ad hoc* un máximo arbitrario de niveles de desplegado. Además, es necesario introducir un procedimiento efectivo de evaluación parcial de un programa multi-adjunto con respecto a un con-

junto de átomos/objetivos, lo que enlaza con las estrategias de *control global*. Estos controles están relacionados con la terminación de procesos recursivos de desplegado, es decir, sobre como frenar la construcción reiterada de árboles de desplegado para átomos/objetivos iguales o similares a los ya tratados, mientras se garantiza que el potencial deseado de especialización se mantiene sin perjudicar la corrección semántica del programa resultante. Nótese que cuando un programa es parcialmente evaluado, la colección de objetivos que aparecen en el conjunto inicial (con respecto al cual se realiza la especialización) normalmente precisa ser aumentado en sucesivas etapas para obtener una especialización efectiva. Estos nuevos objetivos deben ser recursivamente tratados (generando nuevos árboles de desplegado), como también ocurrirá con las técnicas de tabulación que veremos más adelante (y que también generan bosques de árboles), para completar satisfactoriamente el proceso de especialización. Por tanto, para garantizar la terminación del proceso de evaluación parcial (y del de tabulación) es importante investigar en técnicas apropiadas capaces de mantener este conjunto finito.

c) Por último, es necesario replantear la corrección y la completitud del proceso completo de evaluación parcial en su vocación especializadora, ya que son muchos los elementos en los que difiere con respecto a la generación de PE-reductantes. En particular, nótese que en esta tesis hemos descrito cómo sintetizar un sólo reductante a partir de todas las hojas de un sólo árbol de desplegado para un sólo átomo (básico), mientras que la especialización de un programa con respecto (incluso) a un sólo átomo, puede requerir la generación de más de un árbol de desplegado, al tiempo que cada una de las hojas de cada árbol dará lugar a una sola regla de programa distinta e independiente (resultante).

3. **Optimización por plegado/desplegado:**

Este tipo de técnicas, también conocidas como la aproximación basada en "reglas + estrategias", consiste en definir un conjunto de reglas de transformación elementales (incluyendo invariablemente las operaciones básicas de plegado y desplegado, que en esencia realizan la contracción y expansión, respectivamente, de expresiones en un programa usando definiciones equivalentes del mismo programa u otros precedentes), que se aplican sobre un programa siguiendo

una determinada estrategia o heurística encargada de dirigir de forma automática el proceso de aplicación de estas reglas al construir la secuencia de transformaciones (según Pettorossi y Proietti [1994, 1996b]).

La operación de desplegado (en sus distintas formulaciones) ha demostrado ampliamente su utilidad en técnicas de análisis, depuración, compilación, síntesis, etc., de programas declarativos, y particularizando en el marco multi-adjunto, en esta tesis hemos dejado constancia de cómo la aplicación reiterada de desplegados difusos (y sus variantes) produce programas cada vez más eficientes (ver Julián *et al.* [2004b,a, 2005a,b, 2006c]). Sin embargo, el potencial optimizador de esta regla de transformación tiene sus límites ya que, por ejemplo, no es capaz de reducir la complejidad algorítmica de los programas. Para superar esta limitación, es preciso combinarla con otras reglas entre las que debe cobrar especial importancia la transformación inversa de plegado. Por tanto, puesto que el desplegado alcanza, posiblemente, sus mejores cotas de aplicación en sistemas de transformación por plegado/desplegado basados en "reglas+estrategias", surge nuestro interés por desarrollar versiones de plegado difuso potentes y efectivamente combinables con los desplegados difusos diseñados hasta ahora.

Como precedente en el marco multi-adjunto, en [Moreno, 2006] ya describimos una primera versión de plegado difuso que es reversible (es decir, sus efectos pueden deshacerse al aplicar un desplegado posterior) y por tanto con poder optimizador limitado, al no mostrarse capaz, por ejemplo, de generar definiciones recursivas de predicados difusos. El sistema de transformación se completa con otras reglas auxiliares como son la introducción de definiciones y el "facting" (que en esencia se corresponde con la cuarta variedad de los reemplazamientos de t-norma desarrollados en esta tesis). En [Moreno, 2006] se demuestra que el sistema global preserva la semántica de los programas y que siempre que el número de desplegados supere al de plegados, queda garantizado el incremento de eficiencia en el programa final, aún cuando esta mejora sea moderada.

Un avance en esta dirección lo damos en [Guerrero y Moreno, 2007, 2008], donde por primera vez se supera el carácter reversible de nuestra primera aproximación en [Moreno, 2006]. La nueva variedad de plegado difuso, esta vez no reversible, es capaz de plegar una regla de programa con otra "plegante" perteneciente a cualquier otro programa de la secuencia de transformación. Una característica muy novedosa de la adaptación de esta regla al marco difuso,

es que exige la aplicación previa de una transformación orientada a "limpiar" el cuerpo de las reglas a plegar sin que éstas pierdan su significado, retirando de ellas tantos agregadores y grados de verdad como sean necesarios para asemejar su aspecto al de la regla que se usará como plegante, y llevándolos a la definición de un nuevo agregador que será invocado desde la regla a plegar. Para ello, se realiza la introducción de nuevas definiciones de agregadores que quedan adecuadamente descritos en términos de estos elementos "que estorban" en la regla de partida, según la distribución original que tenían en la misma. El proceso global (que tiene ciertas correspondencias con la así llamada regla de "abstracción" de los sistemas de transformación funcionales y lógico-funcionales) es llevado a cabo por lo que nosotros hemos denominado "transformación de agregación", que presenta altos grados de originalidad en nuestro marco y de la que volveremos a hablar más adelante.

El poder optimizador del nuevo sistema de transformación ahora es mucho más elevado que su predecesor, permitiendo la síntesis de predicados recursivos que en muchas ocasiones derivan en programas transformados con complejidades algorítmicas mucho más ligeras que los originales. En cualquier caso, aunque todo parece indicar que la nueva transformación preserva la semántica de los programas bajo las mismas condiciones de aplicabilidad que la versión reversible primitiva, todavía queda por demostrar formalmente sus propiedades fundamentales de corrección y completitud.

Además, como ya ocurre en otros paradigmas declarativos precedentes, se sabe que el alto poder optimizador del plegado no reversible suele venir compañado, también en el marco difuso, de fuertes riesgos de producir secuencias de transformación degeneradas que arrojen programas más ineficientes incluso que los originales. Por tanto, ahora resulta inaplazable el diseño de estrategias de transformación satisfactorias capaces de conducir hábilmente estas secuencias hasta alcanzar programas optimizados. En este sentido, algunas experiencias previas adquiridas en nuestro grupo (ver [Alpuente *et al.*, 2004; Moreno, 2000]) en la definición de este tipo de heurísticas en un marco lógico-funcional, pueden ser muy valiosas para adaptar al contexto difuso las estrategias clásicas de composición y formación de tuplas, entre otras.

4. **Tabulación con umbralización:**

   Las técnicas de tabulación proporcionan un procedimiento para la contestación

de preguntas que se basa en el almacenamiento y reutilización de cómputos en tiempo de ejecución ([Swift, 1999; Tamaki y Sato, 1986]), manteniendo ciertas correspondencias con las técnicas de evaluación parcial y nuestros métodos de construcción de PE-reductantes (sobre todo en lo referentes a la construcción de bosques de árboles) aún cuando en estos últimos casos los objetivos difieran: en vez de ejecutar programas de forma eficiente, lo que se trata es de especializarlos o "completarlos" para que su posterior ejecución presente mejores propiedades de eficiencia y/o completitud.

Las primeras adaptaciones de las técnicas clásicas de tabulación provenientes de la programación lógica pura ([Tamaki y Sato, 1986; Swift, 1999]) al contexto difuso de los programas tanto residuados como multi-adjuntos, las encontramos en [Damásio *et al.*, 2004b,a]. La poda de ramas inútiles en tiempo de ejecución se hace sobre aquellas computaciones que, como en el caso clásico, se sabe que sólo conducen a soluciones repetidas. Sin embargo, en los marcos difusos, la gestión de los grados de verdad proporciona nuevas oportunidades para eliminar las ramas que, aún cuando no conduzcan a soluciones redundantes, resultan irrelevantes al llevar asociados grados de verdad no significativos (inferiores a otras salidas más representativas).

Este efecto ya se ha observado en la tesis al construir PE-reductantes, lo que nos llevó a la generación eficiente de árboles de desplegado umbralizados, donde muchas ramas irrelevantes (que conducían a hojas con grados de verdad muy degradados) no llegaban nunca a generarse, incrementando no sólo la eficiciencia del proceso global (en tiempo y en espacio) sino también garantizando una mayor calidad de los PE-reductantes obtenidos, lo que revertía en una ejecución más rápida de los programas resultantes. Con el mismo espirítu, pero está vez orientados a las técnicas de tabulación difusa, la versión umbralizada que proponemos en [Julián *et al.*, 2008a] supone una nueva semántica operacional para programas multi-adjuntos que goza de altos grados de eficiencia, en la que no sólo se soslayan computaciones redundantes (que son los efectos clásicos de la tabulación) sino que también se evitan (gracias a la umbralización dinámica) aquéllas que conducen a soluciones inútiles por llevar asociados grados de verdad degradados.

Un paso más allá en esta misma línea, lo damos en [Julián *et al.*, 2008b; Julián *et al.*, 2009], donde se mejoran los criterios de actualización dinámica de umbrales. Si en la versión anterior el filtro básico lo constituía el grado de

verdad de la regla de programa $\mathcal{R}$ que se pretendía explotar en un nuevo paso de computación, ahora se usan dos nuevos criterios de frenado de computaciones supérfluas: uno basado en una estimación segura y al alza del cuerpo de $\mathcal{R}$ (asumiendo que en el mejor de los casos los átomos que contienen serán evaluados con el mayor grado de verdad posible) y otro más que se obtiene por combinación de los dos anteriores (mediante la conjunción adjunta a la implicación de $\mathcal{R}$). Con todo ello conseguimos que este conjunto de tres filtros optimice al máximo la gestión de umbrales dinámicos, reduciendo muy significativamente la generación de ramas redundantes e inútiles.

Finalmente, en [Julián *et al.*, 2010] nos planteamos condiciones que aseguren la terminación de los procesos computacionales propuestos anteriormente. En principio, y partiendo de las evidencias obtenidas en precedentes lógicos puros, sabemos que estas condiciones suelen estar muy relacionadas (cuando no son idénticas) tanto en técnicas de evaluación parcial como de tabulación ([Verbaeten *et al.*, 2001; Vidal, 2007]). En esos marcos es frecuente que, aún cuando las ejecuciones estándar de algunos programas entren en bucle infinito, sí terminen las basadas en tabulación (y la especialización). Sin embargo, la pregunta que nos surge en nuestro marco difuso es si es posible encontrar situaciones donde la semántica operacional estándar, la EP y la tabulación difusa no terminen, mientras que sí lo hagan sus versiones umbralizadas. Afortunadamente, en [Julián *et al.*, 2010] ilustramos que estas situaciones existen, lo que da todavía mucho más sentido al uso de umbrales a la hora de construir árboles de desplegado: las técnicas de EP y tabulación ya no sólo mejoran sus rendimientos al ser umbralizadas, sino que además se hacen aplicables en casos donde las versiones no umbralizadas presentaban comportamientos no terminantes.

Para el futuro inmediato queda pendiente no sólo elevar nuestros resultados al primer orden, ya que actualmente nuestras técnicas umbralizadas de tabulación difusa sólo se centran en el caso proposicional, sino también formalizar las nuevas condiciones de terminación (que se adivinan mucho más flexibles gracias a la umbralización, especialmente cuando consideremos programas de primer orden), caracterizar la clase ahora mucho más amplia de programas sobre la que nuestras técnicas son aplicables de forma segura, y por último reforzar y realimentar con todos estos resultados la revisión de las técnicas de evaluación parcial difusa y cálculo de PE-reductantes que hemos desarrollado hasta este momento.

5. **Otras actuaciones relacionadas:**

   Todos los pasos que se están dando en nuestro grupo de investigación en cuanto al desarrollo del marco multi-adjunto, ya están materializados o esperan su inminente implementación práctica sobre el entorno FLOPER que hemos descrito en el Capítulo 2. Como ya dijimos, el prototipo en su versión inicial ya es capaz de compilar programas lógicos multi-adjuntos a código Prolog estándar (que posteriormente puede ser ejecutado sobre cualquier intérprete Prolog de manera completamente transparente para el usuario) siguiendo el método descrito en esta tesis y publicado en [Julián *et al.*, 2006c].

   En algunas actuaciones posteriores ([Morcillo y Moreno, 2008]), se analiza un nuevo método de representación más detallada de los programas difusos, que también se basa en una codificación Prolog, pero esta vez de forma más sofisticada (cada regla difusa genera un hecho Prolog donde se "desmenuzan" todos sus componentes para ser procesados a más bajo nivel por otros predicados descritos también en forma de cláusulas Prolog) lo que permite visualizar detalles internos de la ejecución de los programas multi-adjuntos. Esta última opción, que amplía el repertorio de funcionalidades de FLOPER, ha desvelado entre otras habilidades, sus capacidades de depuración declarativa mediante la generación de trazas que permiten la visualización de árboles de desplegado, por lo que de alguna manera todo parece indicar que la implantación de todas nuestras técnicas basadas en este tipo construcciones arborescentes (evaluación parcial difusa, PE-reductantes, tabulación umbralizada, etc.) tendrán una rápida implantación sobre FLOPER.

   Aunque a más largo plazo, en nuestro grupo de trabajo, nos planteamos nuevos métodos de generación de un tipo de código más eficiente que las versiones basadas en Prolog (como por ejemplo, aquellas formulaciones que puedan hacer uso de las extensiones difusas que hemos realizado sobre la Máquina Abstracta de Warren, WAM, [Aït-Kaci, 1991; Warren, 1983] y que describimos en [Julián y Rubio-Manzano, 2006, 2009]), actualmente estamos mejorando el interfaz gráfico de la herramienta para facilitar su interacción con el usuario. Más allá, también estamos implementando sobre el sistema diversos modelos de computaciones interpretativas que superan nuestra definición inicial de "paso interpretativo" propuesta en [Julián *et al.*, 2006c], como es el caso de la definición de "paso interpretativo corto" que se proporciona en [Morcillo y Moreno, 2009a].

Este último concepto enlaza con los análisis de coste computacional descritos en el Capítulo 6 de esta tesis, ante la evidencia de que si bien en un programa lógico multi-adjunto los objetivos son evaluados en dos fases computacionales separadas, operacional e interpretativa, esta segunda fase precisaba desde sus inicios de mayores esfuerzos de formalización y contabilización de su coste computacional. La medida propuesta en esta memoria (ver [Julián *et al.*, 2007c]) es pionera en diferenciar las diferentes cargas computacionales asociadas a dos pasos interpretativos que evalúan connectivas de diferente complejidad en función del número de operadores primitivos involucrados (directa o indirectamente) en la definición de cada una de ellas. Sin embargo, esta asignación de "pesos" o sobrecarga computacional asociada a los conectivos que invocan a otros conectivos para ser evaluados durante la fase interpretativa (cuyo número de llamadas es además tenido en cuenta de forma precisa en [Morcillo y Moreno, 2009b]) se resuelve de forma sencilla mediante el simple recuento de "pasos interpretativos cortos" que se dan a lo largo de una derivación. Este nuevo concepto se basa en la distinción de pasos "cortos" aplicados durante la fase interpretativa que, o bien expanden "sintácticamente" la definición de las funciones de verdad de los agregadores evaluados sobre los estados de una derivación, o bien tan sólo proceden a evaluar un único operador primitivo en cada paso (normalmente una operación aritmética del tipo $+$, $*$, $-$, min, etc.). Todos estos desarrollos están llamados a influir de forma significativa en la calibración de las distintas técnicas y herramientas con fuerte sabor difuso que venimos comentando (plegado/desplegado, evaluación parcial, tabulación umbralizada, etc.).

En cualquier caso, en [Morcillo y Moreno, 2009d] se ha iniciado una nueva línea de trabajo que, de forma colateral y en colaboración con todos los esfuerzos realizados anteriormente para mitigar y dar más precision a las medidas de coste propuestas, permitirá reducir la complejidad de las definiciones de conectivas y así reducir su impacto computacional a la hora de ser evaluadas. Ante la evidencia de que durante los procesos de transformación por plegado/desplegado que hemos descrito anteriormente, la regla de "agregación" puede producir en su aplicación reiterada definiciones de conectivas con altos niveles de anidamiento en invocaciones a otros agregadores, urge la necesidad de optimizar estas definiciones. Para conseguir este fin, en [Morcillo y Moreno, 2009d] se esbozan algunas técnicas que se inspiran a su vez en:

- por una parte, métodos clásicos de plegado/desplegado especialmente concebidos en su origen para "Sistemas de Reescritura de Términos" (o TRS's según Klop [1992]; Baader y Nipkow [1998]; Huet y Lévy [1992]; Klop y Middeldorp [1991], que sientan en buena parte las bases de la programación funcional y lógico-funcional) debido sobre todo al enorme parecido entre estos sistemas y las ecuaciones que definen los agregadores difusos, y
- por otra parte, distintos tipos de análisis de terminación de programas basados en la manupulación de grafos de dependencias y también "size-change" ([Arts y Giesl, 2000b; Lee *et al.*, 2001]), usados en múltiples contextos declarativos.

Bibliografía

Ackerman R., 1967. *Introduction to Many Valued Logics*. Dover, New York.

Aczél J., 1948. On mean values. *Bulletin of the American Mathematical Society*, 54(2):392–400.

Adams J.B., 1976. Probabilistic reasoning and certainty factors. *Mathematical Biosciences*, 32:177–186.

Aït-Kaci H., 1991. Warren's Abstract Machine: A Tutorial Reconstruction. *The MIT Press, Cambridge, Massachusetts*.

Albert E., Alpuente M., Falaschi M., Julián P. y Vidal G., 1998. Improving Control in Functional Logic Program Specialization. En G. Levi, ed., *Proc. of Static Analysis Symposium, SAS'98*, págs. 262–277. Lecture Notes in Computer Science 1503.

Albert E., Alpuente M., Hanus M. y Vidal G., 1999. A Partial Evaluation Framework for Curry Programs. En *Proc. of the 6th International Conference on Logic for Programming and Automated Reasoning, LPAR'99*, págs. 376–395. Lecture Notes in Artificial Intelligence 1705.

Alpuente M., Falaschi M., Julián P. y Vidal G., 1997a. Specialization of Lazy Functional Logic Programs. En *Proc. of the ACM SIGPLAN Conference on Partial Evaluation*, Sigplan Notices 32(12), págs. 151–162. ACM Press, New York.

Alpuente M., Falaschi M., Moreno G. y Vidal G., 1997b. Safe Folding/Unfolding with Conditional Narrowing. En M. Hanus, H. Heering y K. Meinke, eds., *Proc. of the International Conference on Algebraic and Logic Programming, ALP'97, Southampton*, págs. 1–15. Lecture Notes in Computer Science 1298.

Alpuente M., Falaschi M., Moreno G. y Vidal G., 1999. A Transformation System for Lazy Functional Logic Programs. En A. Middeldorp y T. Sato, eds., *Proc. of the 4th Fuji International Symposyum on Functional and Logic Programming, FLOPS'99, Tsukuba*, págs. 147–162. Lecture Notes in Computer Science 1722.

Alpuente M., Falaschi M., Moreno G. y Vidal G., 2004. Rules + Strategies for Transforming Lazy Functional Logic Programs. *Theoretical Computer Science*, 311:479–525.

Alpuente M., Falaschi M. y Vidal G., 1998. Partial Evaluation of Functional Logic Programs. *ACM Transactions on Programming Languages and Systems*, 20(4):768–844.

Alsina C., Castro J.L. y Trillas E., 1995. On the characterization of S and R implications. En *Proc. of the 6th International Fuzzy Systems Association World Congress, Sao Paulo*, volumen 1, págs. 317–319.

Alsinet T., Chesnevar C.I., Godo L. y Simari G.R., 2008. A logic programming framework for possibilistic argumentation: Formalization and logical properties. *Fuzzy Sets and Systems*, 159(10):1208–1228.

Alsinet T. y Godo L., 1998. Fuzzy Unification Degree. En *Proc. 2th International Workshop on Logic Programming and Soft Computing'98, in conjunction with JICSLP'98, Manchester*, pág. 18.

Alsinet T. y Godo L., 2000. A Complete Calculus for Possibilistic Logic Programming with Fuzzy Propositional Variables. En *Proc. of the 16th Conference on Uncertainty in Artificial Intelligence, UAI'00*, págs. 1–10. Morgan Kaufmann Publishers Inc., San Francisco.

Apt K.R., 1990. Introduction to Logic Programming. En J. van Leeuwen, ed., *Handbook of Theoretical Computer Science*, volumen B: Formal Models and Semantics, págs. 493–574. Elsevier, Amsterdam and The MIT Press, Cambridge, Massachusetts.

Apt K.R., 1997. *From Logic Programming to Prolog*. International Series in Computer Science, Prentice Hall.

Aranda J., Fernández J.L. y Morilla F., 1993. *Lógica Matemática*. Sanz y Torres.

Arcelli F. y Formato F., 1999. Likelog: A Logic Programming Language for Flexible Data Retrieval. En *Proc. of the ACM Symposium on Applied Computing, SAC'99, San Antonio, Texas*, págs. 260–267. ACM, Artificial Intelligence and Computational Logic.

Arcelli F. y Formato F., 2002. A similarity-based resolution rule. *International Journal of Intelligent Systems*, 17(9):853–872.

Arts T. y Giesl J., 2000a. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236(1-2):133–178.

Arts T. y Giesl J., 2000b. Termination of term rewriting using dependency pairs. *Theoretical Computer Science*, 236(1-2):133–178.

Atanassov K. y Georgiev C., 1993. Intuitionistic fuzzy Prolog. *Fuzzy Sets Systems*, 53(2):121–128.

Baader F. y Nipkow T., 1998. *Term Rewriting and All That*. Cambridge University Press.

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

Baral C., Gelfond M. y Rushton N., 2004. Probabilistic reasoning with answer sets. En *Proc. of the 7th International Conference in Logic Programming and Nonmonotonic Reasoning, LPNMR'04, Fort Lauderdale, Florida*, págs. 21–33. Lecture Notes in Artificial Intelligence 2923.

Bossi A., Gabbrielli M., Levi G. y Martelli M., 1994. The s-semantics approach: Theory and applications. *Journal of Logic Programming*, 19-20:149–197.

Burstall R.M. y Darlington J., 1977. A Transformation System for Developing Recursive Programs. *Journal of the ACM*, 24(1):44–67.

Caballero R., Rodríguez-Artalejo M. y Romero-Díaz C.A., 2008. Similarity-based reasoning in qualified logic programming. En *PPDP'08: Proc. of the 10th international ACM SIGPLAN conference on Principles and practice of declarative programming*, págs. 185–194. ACM, New York.

Calvo T., Baets B.D. y Mesiar R., 1999. Weighted sums of aggregation operators. *Mathware & soft computing*, 6(1):33–47.

Calvo T., Kolesárová A., Komorníková M. y Mesiar R., 2002. Aggregation operators: properties, classes and construction methods. En T. Calvo, G. Mayor, y R. Mesiar, eds., *Aggregation operators: new trends and applications*, págs. 3–104. Physica-Verlag GmbH, Heidelberg, Germany.

Cao T.H., 2000. Annotated fuzzy logic programs. *Fuzzy Sets and Systems*, 113(2):277–298.

Carlsson C. y Fullér R., 1995. On fuzzy screening system. En V. Mainz, ed., *Proc. of the 3th European Congress on Intelligent Techniques and Soft Computing, EUFIT'95, Aachen*, págs. 1261–1264.

Carlsson C., Fullér R. y Fullér S., 1997. Possibility and necessity in weighted aggregation. En R. Yager y J. Kacprzyk, eds., *The ordered weighted averaging operators: Theory, Methodology and Applications*, págs. 18–28. Kluwer Academic Publishers.

Castillo E., Gutiérrez J. y Hadi A., 1996. *Sistemas Expertos y Modelos de Redes Probabilísticas*. Monografías de la Academia de Ingeniería.

Chang C.C., 1958. Algebraic analysis of many valued logics. *Transactions of the American Mathematical Society*, 88:467–490.

Chesñevar C.I., Simari G.R., Alsinet T. y Godo L., 2004. A logic programming framework for possibilistic argumentation with vague knowledge. En *Proc. of the 20th conference on Uncertainty in artificial intelligence, AUAI'04*, págs. 76–84. AUAI Press, Arlington, Virginia.

Cohen P.R., 1985. *Heuristic reasoning about uncertainty: an artificial intelligence approach*. Pitman Publishing, Inc., Marshfield, Massachusetts.

Consel C. y Danvy O., 1993. Tutorial notes on Partial Evaluation. En *Proc. of 20th Annual ACM Symposium on Principles of Programming Languages*, págs. 493–501. ACM, New York.

Consel C., Hornof L., Noël F., Noyé J.. y Volanschi E., 1996. A Uniform Approach for Compile-Time and Run-Time Specialisation. En *Proc. of the 1996 Dagstuhl Seminar on Partial Evaluation*, págs. 54–72. Lecture Notes in Computer Science 1110.

Cordón O., Herrera F., Hoffmann F. y Magdalena L., 2001. Genetic Fuzzy Systems: Evolutionary Tuning and Learning of Fuzzy Knowledge Bases. En *Advances in Fuzzy Systems Applications and Theory*, volumen 19. World Scientific.

Damásio C.V., Medina J. y Ojeda-Aciego M., 2004a. Sorted multi-adjoint logic programs: termination results and applications. En *Proc. of Logics in Artificial Intelligence, JELIA'04, Lisbon*, págs. 260–273. Lecture Notes in Artificial Intelligence 3229.

Damásio C.V., Medina J. y Ojeda-Aciego M., 2004b. A tabulation proof procedure for residuated logic programming. *In Proc. of the European Conference on Artificial Intelligence, Frontiers in Artificial Intelligence and Applications*, 110:808–812.

Damasio C.V., Medina J. y Ojeda-Aciego M., 2004. Termination results for sorted multi-adjoint logic programming. En *Proc. of the 10th International Conference on Information Processing and Managment of Uncertainty in Knowledge-Based Systems, IPMU'04, Perugia*, págs. 1879–1886.

Damásio C.V., Medina J. y Ojeda-Aciego M., 2007. Termination of logic programs with imperfect information: applications and query procedure. *Journal of Appplied Logic*, 5:435–458.

Damásio C.V. y Moniz-Pereira L., 2000. Hybrid Probabilistic Logic Programs as Residuated Logic Programs. En *JELIA '00: Proc. of the European Workshop on Logics in Artificial Intelligence*, págs. 57–72. Springer-Verlag, London.

Damásio C.V. y Moniz-Pereira L., 2001a. Antitonic Logic Programs. En *Proc. of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR'01, Vienna*, págs. 379–392. Springer-Verlag.

Damásio C.V. y Moniz-Pereira L., 2001b. Monotonic and residuated logic programs. En S. Benferhat y P. Besnard, eds., *Proc. of the 6th European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, ECSQARU'01, Toulouse*, págs. 748–759. Lecture Notes in Artificial Intelligence 2143.

Damásio C.V. y Moniz-Pereira L., 2002. Hybrid Probabilistic Logic Programs as Residuated Logic Programs. *Lecture Notes in Computer Science*, 1919:57–72.

Damásio C.V. y Moniz-Pereira L., 2004. Sorted Monotonic Logic Programs and their Embeddings. En *Proc. of the 10th International Conference on Information Processing and Managment of Uncertainty in Knowledge-Based Systems, IPMU'04, Perugia*, págs. 807–814.

Dekhtyar A. y Dekhtyar M.I., 2005. Revisiting the semantics of interval probabilistic logic programs. En *8th International Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR'05*, volumen 3662, págs. 330–342. Lecture Notes in Computer Science.

Dekhtyar A. y Subrahmanian V.S., 2000. Hybrid Probabilistic Programs. *Journal of Logic Programming*, 43(3):187–250.

Dilworth R., 1939. Residuated Lattices. *Transactions of the American Mathematical Society*, 45:335–354.

Driankov D., Hellendoorn H. y Reinfrank M., 1996. *An introduction to control fuzzy*. Springer-Verlag.

Dubois D., Lang J. y Prade H., 1991a. Fuzzy sets in approximate reasoning, part 2: logical approaches. *Fuzzy Sets and Systems*, 40(1):203–244.

Dubois D., Lang J. y Prade H., 1991b. Towards possibilistic logic programming. En *Proc. of the 8th International Confenrence on Logic Programming, ICLP'91*, págs. 581–595. The MIT Press.

Dubois D. y Prade H., 1980. *Fuzzy Sets and Systems: Theory and Applications*. Academic Press.

Dubois D. y Prade H., 1984. Criteria aggregation and ranking of alternatives in the framework of fuzzy set theory. *TIMS Studies in the Management Sciences*, 20:209–240.

Dubois D. y Prade H., 1985. A review of fuzzy sets aggregation connectives. *Information Sciences*, 36:85–121.

Dubois D. y Prade H., 1986. Weighted minimun and maximum operations in fuzzy sets theory. *Information Sciences*, 39:205–210.

Dubois D. y Prade H., 1991. Fuzzy sets in approximate reasoning, part 1: inference with possibility distributions. *Fuzzy Sets and Systems*, 40(1):143–202.

Dubois D., Prade H. y Sandri S., 1998. Possibilistic logic with fuzzy constants and fuzzily restricted quantifiers. En *Logic Programming and Soft Computing*, págs. 69–90. T. P. Martin and F. Arcelli-Fontana eds., Baldock.

Duda R.O., Hart P.E. y Nilsson N.J., 1990. Subjective Bayesian methods for rule-based inference systems. En *Readings in uncertain reasoning*, págs. 274–281. Morgan Kaufmann Publishers Inc., San Francisco.

Durante F., Sempi C., Mesiar R. y Klement E.P., 2007. Conjunctors and their residual implicators: characterizations and construction methods. *Mediterranean Journal of Mathematics*, 4(3):343–356.

van Emden M.H., 1986. Quantitative Deduction and its Fixpoint Theory. *Journal of Logic Programming*, 3(1):37–53.

van Emden M.H. y Kowalski R.A., 1976. The Semantics of Predicate Logic as a Programming Language. *Journal of the ACM*, 23(4):733–742.

Falaschi M., 1988. *Semantica del non determinismo nei linguaggi logici concorrenti.* Servizio Editoriale Universitario, Università di Pisa.

Falaschi M., Levi G., Martelli M. y Palamidessi C., 1989. Declarative Modeling of the Operational Behavior of Logic Languages. *Theoretical Computer Science*, 69(3):289–318.

Falaschi M., Levi G., Martelli M. y Palamidessi C., 1993. A Model-Theoretic Reconstruction of the Operational Semantics of Logic Programs. *Information and Computation*, 103(1):86–113.

Fay M., 1979. First Order Unification in an Equational Theory. En *Proc. of 4th International Conference on Automated Deduction*, págs. 161–167.

Fitting M., 1991. Bilattices and the semantics of logic programming. *Journal of Logic Programming*, 11:91–116.

Fodor J. y Calvo T., 1998. Aggregation functions defined by t-norms and t-conorms. En B. Bouchon-Meunier, ed., *Aggregation and Fusion of Imperfect Information*, págs. 36–48. Physica Verlag.

Fodor J. y Roubens M., 1992. Aggregation and scoring procedures in multicriteria decision making methods. En *Proc. of the IEEE International Conference on Fuzzy Systems 1992*, págs. 1261–1267.

Fodor J. y Yager R.R., 1994. Fuzzy Preference Modelling and Multicriteria Decision Support. En *Theory and Decision Library, Series D, System Theory, Knowlege engineering and Problem Solving*, volumen 14. Kluwer Academic, Kluwer, Dordrecht.

Formato F., Gerla G. y Sessa M.I., 1999. Extension of Logic Programming by Similarity. En *Proc. of the Italian-Portuguese-Spanish Joint Conference on Declarative Programming, APPIA-GULP-PRODE'99, L'Aquila*, págs. 397–410.

Formato F., Gerla G. y Sessa M.I., 2000. Similarity-based Unification. *Fundamenta Informaticae*, 40(4):393–414.

Fuhr N., 2000. Probabilistic Datalog: implementing logical information retrieval for advanced applications. *Journal of the American Society for Information Science*, 51(2):95–110.

Gallagher J., 1993. Tutorial on Specialisation of Logic Programs. En *Proc. of the 1993 ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, págs. 88–98. ACM, New York.

Gerla G., 2001. Fuzzy control as a fuzzy deduction system. *Fuzzy Sets and Systems*, 121(3):409–425.

Gerla G., 2004. Representation theorems for fuzzy orders and quasi-metrics. *Soft Computing*, 8(8):571–580.

Gerla G., 2005. Fuzzy Logic Programming and fuzzy control. *Studia Logica*, 79:231–254.

Giarratano J. y Riley G., 1994. *Experts Systems: Principles and Programming*. PWS, Boston.

Ginsberg M.L., 1988. Multi-valued logics: a uniform approach to reasoning in artificial intelligence. *Computational Intelligence*, 4:265–316.

Glück R., Jørgensen J., Martens B. y Sørensen M., 1996. Controlling Conjunctive Partial Deduction of Definite Logic Programs. En *Proc. International Symposium on Programming Languages: Implementations, Logics and Programs, PLILP'96*, págs. 152–166. Lecture Notes in Computer Science 1140.

Glück R. y Sørensen M., 1994. Partial Deduction and Driving are Equivalent. En *Proc. International Symposium on Programming Language Implementation and Logic Programming, PLILP'94*, págs. 165–181. Lecture Notes in Computer Science 844.

Goguen J.A., 1969. The logic of inexact concepts. *Synthese*, 19:325–373.

Guadarrama S., Muñoz S. y Vaucheret C., 2004. Fuzzy Prolog: A New Approach Using Soft Constraints Propagation. *Fuzzy Sets and Systems, Elsevier*, 144(1):127–150.

Guerrero J. y Moreno G., 2008. Optimizing Fuzzy Logic Programs by Unfolding, Aggregation and Folding. *Electronic Notes in Theoretical Computer Science*, 219:19–34. Extended version of Guerrero y Moreno [2007].

Guerrero J.A. y Moreno G., 2007. Optimizing Fuzzy Logic Programs by Unfolding, Aggregation and Folding. En J. Visser y V. Winter, eds., *Proc. of the 8th. International Workshop on Rule-Based Programming, RULE'07, Paris, June 29*, pág. 15. University of Paris.

Gupta M.M. y Qi J., 1991. Theory of T-norms and fuzzy inference methods. *Fuzzy Sets and Systems*, 40(3):431–450.

Hájek P., 1998. *Metamathematics of Fuzzy Logic*. Kluwer Academic Publishers, Dordrecht.

Hajek P., 2006. Fuzzy Logic. En E.N. Zalta, ed., *The Stanford Encyclopedia of Philosophy (Summer 2008 Edition)*. The MRL and the CSLI, Stanford University.

Herrera F., Herrera-Viedma E. y Verdegay J.L., 1996. Direct approach processes in group decision making using linguistic OWA operators. *Fuzzy Sets and Systems*, 79(2):175–190.

Hinde C., 1986. Fuzzy prolog. *International Journal Man-Machine Studies*, 24:569–595.

Huet G. y Lévy J., 1992. Computations in Orthogonal Rewriting Systems, Part I + II. En J. Lassez y G. Plotkin, eds., *Computational Logic – Essays in Honor of Alan Robinson*, págs. 395–443. The MIT Press, Cambridge, MA.

Ishizuka M. y Kanai N., 1985. Prolog-ELF incorporating fuzzy logic. En A.K. Joshi, ed., *Proc. of the 9th International Joint Conference on Artificial Intelligence, IJCAI'85. Los Angeles, 1985.*, págs. 701–703. Morgan Kaufmann.

Jenei S., 2004. How to construct left-continuous triangular norms: state of the art. *Fuzzy Sets and Systems*, 143:27–45.

Jenei S., 2006. On the convex combination of left-continuous t-norms. *Aequationes Mathematicae*, 72:47–59.

Jenei S. y Montagna F., 2003. A general method for constructing left-continuous t-norms. *Fuzzy Sets and Systems*, 136:263–282.

Jones N., Gomard C. y Sestoft P., 1993. *Partial Evaluation and Automatic Program Generation*. Prentice-Hall, Englewood Cliffs, New York.

Julián P., 2000. *Especialización de Programas Lógico-Funcionales Perezosos*. Universidad de Valencia. Tesis Doctoral.

Julián P., 2004. *Lógica simbólica para informáticos*. RA-MA, Madrid.

Julián P. y Alpuente M., 2007. *Programación Lógica. Teoría y Práctica*. Pearson Educación, S.A., Madrid.

Julián P., Medina J., Moreno G. y Ojeda M., 2008a. Combining Tabulation and Thresholding Techniques for Executing Multi-Adjoint Logic Programs. En L. Magdalena, M. Ojeda-Aciego y J. Verdegay, eds., *In Proc. of the 12th International Conference on Information Processing and Management of Uncertainty in Knoledge-based Systems, IPMU'08, June 22-27, Málaga*, págs. 550–512. University of Málaga.

Julián P., Medina J., Moreno G. y Ojeda M., 2008b. Thresholded Tabulation in a Fuzzy Logic Setting. En J. Almendros, ed., *In Proc. of VIII Jornadas sobre Programación y Lenguajes, PROLE'08, Gijón, October, 7-10*, págs. 57–71. Fundación Universidad de Oviedo.

Julián P., Medina J., Moreno G. y Ojeda M., 2010. Efficient Thresholded Tabulation for Fuzzy Query Answering. *Studies in Fuzziness and Soft Computing (Foundations of Reasoning under Uncertainty)*, 249:125–141.

Julián P., Medina J., Moreno G. y Ojeda-Aciego M., 2009. Thresholded Tabulation in a Fuzzy Logic Setting. *Electronic Notes in Theoretical Computer Science*, 248:115–130. Improved version of Julián *et al.* [2008b].

Julián P., Moreno G. y Penabad J., 2004a. Unfolding-based Improvements on Fuzzy Logic Programs. En S. Lucas, ed., *Proc. of IV Jornadas sobre Programación y Lenguajes, PROLE'04, Málaga, November 11-12*, págs. 241–254. University of Málaga.

Julián P., Moreno G. y Penabad J., 2004b. Unfolding Fuzzy Logic Programs. En *Proc. of the 4th International Conference on Intelligent Systems Design and Applications, ISDA'04 (Sponsored by IEEE). Budapest, August 26-28*, págs. 595–600.

Julián P., Moreno G. y Penabad J., 2004c. Unfolding Fuzzy Logic Programs. Informe Técnico DIAB-04-03-2, Departamento de Informática, Universidad de Castilla-La Mancha.

Julián P., Moreno G. y Penabad J., 2005a. On Fuzzy Unfolding. A Multi-Adjoint Approach. *Fuzzy Sets and Systems, Elsevier*, 154:16–33.

Julián P., Moreno G. y Penabad J., 2005b. Operational/Interpretive Unfolding of Multi-adjoint Logic Programs. En F. López-Fraguas, ed., *Proc. of V Jornadas sobre Programación y Lenguajes, PROLE'05, Granada, September 14-16*, págs. 239–248. University of Granada.

Julián P., Moreno G. y Penabad J., 2005c. Unfolding-based Improvements on Fuzzy Logic Programs. En *Electronic Notes in Theoretical Computer Science*, volumen 137, págs. 69–103. Elsevier, Amsterdam.

Julián P., Moreno G. y Penabad J., 2006a. Efficient Reductants Calculi using Partial Evaluation Techniques with Thresholding. En P. Lucio, ed., *Proc. of VI Jornadas sobre Programación y Lenguajes, PROLE'06, Sitges, October 10-12*, págs. 275–289. Technical University of Catalonia.

Julián P., Moreno G. y Penabad J., 2006b. Evaluación Parcial de Programas Lógicos Multi-Adjuntos y Aplicaciones. En A.F. Caballero, ed., *Proc. of Campus Multidisciplinar sobre Percepción e Inteligencia, CMPI'06. Albacete, July 10-12*, págs. 712–724. University of Castilla–La Mancha.

Julián P., Moreno G. y Penabad J., 2006c. Operational/Interpretive Unfolding of Multi-adjoint Logic Programs. *Journal of Universal Computer Science*, 12(11):1679–1699.

Julián P., Moreno G. y Penabad J., 2007a. Cost Measures for Fuzzy Logic Computations. En E. Pimentel, ed., *Proc. of VII Jornadas sobre Programación y Lenguajes, PROLE'07, Zaragoza, September 12-14*, págs. 103–110. Thomson-Paraninfo.

Julián P., Moreno G. y Penabad J., 2007b. Efficient Reductants Calculi using Partial Evaluation Techniques with Thresholding. En *Electronic Notes in Theoretical Computer Science*, volumen 188C, págs. 77–90. Elsevier, Amsterdam.

Julián P., Moreno G. y Penabad J., 2007c. Measuring the Interpretive Cost in Fuzzy Logic Computations. En F. Masulli y al., eds., *Proc. of 7th. International Whorkshop on Fuzzy Logic and Applications, WILF'07, Portofino, July 07-10*, págs. 28–36. Springer Verlag, Lecture Notes in Artificial Intelligence 4578.

Julián P., Moreno G. y Penabad J., 2009. An Improved Reductant Calculus using Fuzzy Partial Evaluation Techniques. *Fuzzy Sets and Systems, Elsevier*, 160:162–181.

Julián P., Moreno G. y Penabad J., 2009. On the Declarative Semantics of Multi-Adjoint Logic Programs. En *Proc. of the 10th International Work-Conference on Artificial Neural Networks, IWANN'09*, págs. 253–260. Springer-Verlag, Lecture Notes In Computer Science, 5517, Berlín.

Julián P. y Rubio-Manzano C., 2006. A WAM Implementation for Flexible Query Answering. En A.P. del Pobil, ed., *In Proc. of the 10th IASTED International Conference on Artificial Intelligence and Soft Computing, ASC'06, August 28-30, Palma de Mallorca*, págs. 262–267. ACTA Press.

Julián P. y Rubio-Manzano C., 2009. A Similarity-Based WAM for Bousi∼Prolog. En J. Cabestany, F.S. Hernández, A. Prieto y J.M. Corchado, eds., *IWANN (1)*, págs. 245–252. Springer, Lecture Notes in Computer Science, 5517.

Julián P. y Villamizar C., 2004. Analizing Definitional Trees: Looking for Determinism. En Y. Kameyama y M.P.J. Stuckey, eds., *Proc. of the 7th Fuji International Symposyum on Functional and Logic Programming, FLOPS'04, Nara*, págs. 55–69. Springer, Lecture Notes in Computer Science, 2998.

Khamsi M.A. y Misane D., 1997. Fixed point theorems in logic programming. *Annals of Mathematics and Artificial Intelligence*, 21:231–243.

Kifer M. y Li A., 1988. On the Semantics of Rule-Based Expert Systems with Uncertainty. En *Proc. of the 2th International Conference on Database Theory, ICDT'88*, págs. 102–117. Springer-Verlag, London.

Kifer M. y Subrahmanian V., 1992. Theory of generalized annotated logic programming and its applications. *Journal of Logic Programming*, 12:335–367.

Klawonn F., 2003. Should fuzzy equality and similarity satisfy transitivity? comments on the paper by M. De Cock and E. Kerre. *Fuzzy Sets and Systems*, 133(2):175–180.

Klawonn F. y Kruse R., 1994. A Łukasiewicz logic based Prolog. *Mathware & Soft Computing*, 1(1):5–29.

Klement E.P., Mesiar R. y Pap E., 2004. Triangular norms. Position paper II: General constructions and parameterized families. *Fuzzy Sets and Systems*, 145(1):411–438.

Klir G.J. y Yuan B., 1995. *Fuzzy Sets and Fuzzy Logic*. Prentice-Hall.

Klop J.W., 1992. Term Rewriting Systems. En S. Abramsky, D. Gabbay y T. Maibaum, eds., *Handbook of Logic in Computer Science*, volumen I, págs. 1–112. Oxford University Press.

Klop J.W. y Middeldorp A., 1991. Sequentiality in Orthogonal Term Rewriting Systems. *Journal of Symbolic Computation*, págs. 161–195.

Kolesárová A. y Komorníková M., 1999. Triangular norm-based iterative compensatory operators. *Fuzzy Sets and Systems*, 104(1):109–120.

Komorowski H., 1982. Partial Evaluation as a Means for Inferencing Data Structures in an Applicative Language: A Theory and Implementation in the Case of Prolog. En *Proc. of 9th ACM Symposium on Principles of Programming Languages*, págs. 255–267.

Kott L., 1985. Unfold/fold program transformation. En M.Nivat y J. Reynolds, eds., *Algebraic methods in semantics*, capítulo 12, págs. 411–434. Cambridge University Press.

Kowalski R.A., 1974. Predicate Logic as a Programming Language. *Information Processing*, 74:569–574.

Krajci S., Lencses R., Medina J., Ojeda-Aciego M., Valverde A. y Vojtás P., 2002. Non-commutativity and Expressive Deductive Logic Databases. En *Proc. of the European Conference on Logics in Artificial Intelligence, JELIA'02*, págs. 149–160. Springer-Verlag, London.

Krajči S., Lencses R. y Vojtáš P., 2002. A data model for annotated programs. *ADBIS Research Communications*, págs. 141–154.

Krajči S., Lencses R. y Vojtáš P., 2004. A comparison of fuzzy and annotated logic programming. *Fuzzy Sets and Systems*, 144:173–192.

Lakshmanan L.V.S. y Sadri F., 1994. Probabilistic Deductive Databases. En *Symposium on Logic Programming*, págs. 254–268.

Lakshmanan L.V.S. y Sadri F., 1997. Uncertain deductive databases: a hybrid approach. *Information Systems*, 22(9):483–508.

Lakshmanan L.V.S. y Sadri F., 2001. On a theory of probabilistic deductive databases. *Theory and Practice of Logic Programming*, 1(1):5–42.

Lakshmanan L.V.S. y Shiri N., 2001. A Parametric Approach to Deductive Databases with Uncertainty. *IEEE Transactions on Knowledge and Data Engineering*, 13(4):554–570.

Lassez J.L., Maher M.J. y Marriott K., 1988. Unification Revisited. En J. Minker, ed., *Foundations of Deductive Databases and Logic Programming*, págs. 587–625. Morgan Kaufmann, Los Altos, California.

Lee C., Jones N. y Ben-Amram A., 2001. The size-change principle for program termination. *SIGPLAN Notices*, 36(3):81–92.

Lee R.C.T., 1972. Fuzzy Logic and the Resolution Principle. *Journal of the ACM*, 19(1):119–129.

Leuschel M., De Schreye D. y de Waal A., 1996. A Conceptual Embedding of Folding into Partial Deduction: Towards a Maximal Integration. En M. Maher, ed., *Proc. of the Joint International Conference and Symposium on Logic Programming, JICSLP'96*, págs. 319–332. The MIT Press, Cambridge, Massachusetts.

Levi G. y Mancarella P., 1988. The Unfolding Semantics of Logic Programs. Informe Técnico TR-13/88, Dipartimento di Informatica, Università di Pisa.

Li D. y Liu D., 1990. *A fuzzy Prolog database system*. John Wiley & Sons, Inc.

Ling C., 1965. Representation of associative functions. *Publicationes Mathematicae Debrecen*, 12:189–212.

Lloyd J., 1987. *Foundations of Logic Programming*. Springer-Verlag, Berlin. Second edition.

Lloyd J. y Shepherdson J., 1991. Partial Evaluation in Logic Programming. *Journal of Logic Programming*, 11:217–242.

Loia V., Senatore S. y Sessa M.I., 2001. Similarity-based SLD Resolution and Its Implementation in An Extended Prolog System. En *Proc. of the 10th IEEE International Conference of Fuzzy Systems FUZZ-IEEE'01, Melbourne*, págs. 650–653.

Loyer Y. y Straccia U., 2002a. Uncertainty and Partial Non-uniform Assumptions in Parametric Deductive Databases. En *Proc. of the European Conference on Logics in Artificial Intelligence*, págs. 271–282. Springer-Verlag, London.

Loyer Y. y Straccia U., 2002b. The Well-Founded Semantics in Normal Logic Programs with Uncertainty. En *Proc. of the 6th International Symposium on Functional and Logic Programming*, págs. 152–166. Springer-Verlag, London.

Loyer Y. y Straccia U., 2003. The Approximate Well-founded Semantics for Logic Programs with Uncertainty. En *Proc. of the 28th International Symposium on Mathematical Foundations of Computer Science*, volumen 2747, págs. 541–550. Lecture Notes in Computer Science.

Loyer Y. y Straccia U., 2004. Epistemic Foundation of the Well-Founded Semantics over Bilattices. En *Proc. of the 29th International Symposium on Mathematical Foundations of Computer Science, MFCS'04*, volumen 3153, págs. 513–524. Springer, Heidelberg, Berlin.

Loyer Y. y Straccia U., 2005. Approximate Well-founded Semantics, Query Answering and Generalized Normal Logic Programs over Lattices. Informe Técnico ISTI-2005-TR-xx, I.S.T.I. - C.N.R.

Loyer Y. y Straccia U., 2006. Epistemic foundation of stable model semantics. *Theory and Practice of Logic Programming*, 6(4):355–393.

Lu J.J., 1996. Logic programming with signs and annotations. *Journal of Logic and Computation*, 6(6):755–778.

Lukasiewicz T., 1999a. Many-Valued Disjunctive Logic Programs with Probabilistic Semantics. En *Proc. of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR'99*, págs. 277–289. Springer-Verlag, London.

Lukasiewicz T., 1999b. Many-Valued First-Order Logics with Probabilistic Semantics. En *Proc. of the 12th International Workshop on Computer Science Logic*, págs. 415–429. Springer-Verlag, London.

Lukasiewicz T., 2001a. Fixpoint Characterizations for Many-Valued Disjunctive Logic Programs with Probabilistic Semantics. En *Proc. of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR'01*, págs. 336–350. Springer-Verlag, London.

Lukasiewicz T., 2001b. Probabilistic logic programming with conditional constraints. *ACM Transactions on Computational Logic*, 2(3):289–339.

Mamdani E., 1993. Twenty Years of Fuzzy Control: Experiences Gained and Lessons Learnt. En *Proc. of the 2th IEEE International Conference on Fuzzy Systems*, págs. 339–344.

Mamdani E. y Assilian S., 1975. An experiment in linguistic synthesis with a fuzzy logic controller. *International Journal on Man Machine Studies*, 7:1–13.

Martin T.P., Baldwin J.F. y Pilsworth B.W., 1987. The implementation of fprolog–a fuzzy prolog interpreter. *Fuzzy Sets Systems*, 23(1):119–129.

Mateis C., 2000. Quantitative disjunctive logic programming: Semantics and computation. *AI Communications*, 13(4):225–248.

Mayor G. y Calvo T., 1997. Extended aggregation functions. En *Proc. Seventh IFSA congress, Prague*, volumen I, págs. 281–285. Academy Sciences.

Medina J. y Ojeda-Aciego M., 2002. A new approach to completeness for multi-adjoint logic programming. En *Proc. of 9th Information Processing and Management of Uncertainty in Knowledge-Based Systems Conference, IPMU'02, Annecy.*

Medina J. y Ojeda-Aciego M., 2004. Multi-adjoint logic programming. En *In Proc. of the 10th International Conference on Information Processing and Managment of Uncertainty in Knowledge-Based Systems, IPMU'04, Perugia*, págs. 823–830.

Medina J., Ojeda-Aciego M. y Ruiz-Calviño J., 2000. Multi-lattices as a basis for generalized fuzzy logic programming. *Lecture Notes in Artificial Intelligence*, 3849:61–70.

Medina J., Ojeda-Aciego M. y Ruiz-Calviño J., 2005. Fuzzy logic programming via multilattices: first results and prospects. En *Proc. of Lógica Fuzzy & Soft Computing, LFSC'05, Granada*, págs. 19–26. Thomson.

Medina J., Ojeda-Aciego M. y Ruiz-Calviño J., 2006. On the ideal semantics of multilattice-based logic programs. En *Proc. of the 11th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems, IPMU'06, Paris*, págs. 463–470.

Medina J., Ojeda-Aciego M. y Ruiz-Calviño J., 2007a. A fixed-point theorem for multivalued functions with application to multilattice-based logic programming. *Lecture Notes in Artificial Intelligence*, 4578:37–44.

Medina J., Ojeda-Aciego M. y Ruiz-Calviño J., 2007b. Fuzzy logic programming via multilattices. *Fuzzy Sets and Systems*, 158(6):674–688.

Medina J., Ojeda-Aciego M. y Ruiz-Calviño J., 2007c. On reachability of minimal models of multilattice-based logic programming. *Lecture Notes in Artificial Intelligence*, 4827:271–282.

Medina J., Ojeda-Aciego M. y Vojtas P., 2001a. A completeness theorem for multi-adjoint logic programming. En *Proc. of 10th IEEE International Conference on Fuzzy Systems, IEEE Press, Melbourne*, volumen 2, págs. 1031–1034.

Medina J., Ojeda-Aciego M. y Vojtás P., 2001b. A Multi-adjoint Logic Approach to Abductive Reasoning. En *Proc. of the 17th International Conference on Logic Programming, EPIA'01*, págs. 269–283. Springer-Verlag, London.

Medina J., Ojeda-Aciego M. y Vojtás P., 2001c. A Procedural Semantics for Multi-adjoint Logic Programming. En *Proc. of the10th Portuguese Conference on Artificial Intelligence on Progress in Artificial Intelligence, Knowledge Extraction,*

Multi-agent Systems, Logic Programming and Constraint Solving, EPIA '01, Lectures Notes in Artificial Intelligence 2258, págs. 290–297. London.

Medina J., Ojeda-Aciego M. y Vojtáš P., 2001d. Multi-adjoint logic programming with continuous semantics. En *Proc. of Logic Programming and Non-Monotonic Reasoning, LPNMR'01, Vienna, Lecture Notes in Artificial Intelligence 2173*, págs. 351–364.

Medina J., Ojeda-Aciego M. y Vojtáš P., 2004. Similarity-based Unification: a multi-adjoint approach. *Fuzzy Sets and Systems, Elsevier*, 146:43–62.

Meritt D., 1989. *Building Expert Systems in Prolog*. Springer Verlag, Berlin.

Mizumoto M., 1989a. Pictorial representations of fuzzy connectives, part I: cases of t-norms, t-conorms and averaging operators. *Fuzzy Sets and Systems*, 31(2):217–242.

Mizumoto M., 1989b. Pictorial representations of fuzzy connectives, part II: cases of compensatory operators and self-dual operators. *Fuzzy Sets and Systems*, 32(1):45–79.

Morcillo P.J. y Moreno G., 2008. Programming with Fuzzy Logic Rules by using the FLOPER Tool. En N.B. et al., ed., *Proc. of 2th International Symposium on Rule Interchange and Applications, RuleML'08. Orlando, October 30-31*, págs. 119–126. Springer Verlag, Lecture Notes in Computer Science 3521.

Morcillo P.J. y Moreno G., 2009a. Modeling Interpretive Steps in Fuzzy Logic Computations. En V.D. Gesù, S.K. Pal y A. Petrosino, eds., *Proc. of the 8th International Workshop on Fuzzy Logic and Applications, WILF'09. Palermo, June 9-12*, págs. 44–51. Springer Verlag, Lecture Notes in Artificial Intelligence 5571.

Morcillo P.J. y Moreno G., 2009b. On Cost Estimations for Executing Fuzzy Logic Programs. En H.R. Arabnia, D. de la Fuente y J.A. Olivas, eds., *Proc. of the 2009 International Conference on Artificial Intelligence, ICAI'09, July 13-16, Las Vegas Nevada*, págs. 217–223. CSREA Press.

Morcillo P.J. y Moreno G., 2009c. A Practical Approach for Ensuring Completeness of Multi-adjoint Logic Computations via General Reductants. En G.M. P. Lucio y R. Peña, eds., *Proc. of IX Jornadas sobre Programación y Lenguajes, PROLE'09, San Sebastián, September 8-11*, págs. 355–363. Universidad del País Vasco.

Morcillo P.J. y Moreno G., 2009d. Unfolding Connective Definitions in Multi-adjoint Logic Programs. En F.J. Martín, ed., *Proc. of the First Workshop on Computational Logics and Artificial Intelligence, CLAI'09 (integrated in CAEPIA'09), Sevilla, November 9*, págs. 59–70. University of Sevilla.

Moreno G., 2000. *Reglas y Estrategias de Transformación para Programas Lógico-Funcionales*. Universidad de Valencia. Tesis Doctoral.

Moreno G., 2006. Building a Fuzzy Transformation System. En J. Wiedermann, G. Tel, J. Pokorný, M. Bieliková y J. Stuller, eds., *Proc. of the 32th Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM'06. Merin, January 21-27*, págs. 409–418. Springer, Lecture Notes in Computer Science 3831.

Moser B., Tsiporkova E. y Klement E.P., 1999. Convex combination in terms of triangular norms: a characterization of idempotent, bisymmetrical and self-dual compensatory operators. *Fuzzy Sets and Systems, Special Issue: Triangular norms*, 104:97–108.

Muggleton S., 1995. Stochastic Logic Programs. En L.D. Raedt, ed., *Proc. of the 5th International Workshop on Inductive Logic Programming*, págs. 254–264. Department of Computer Science, Katholieke Universiteit Leuven.

Mukaidono M., 1982. Fuzzy inference of resolution style. *Fuzzy Sets and Possibility Theory*, págs. 224–231.

Mukaidono M., Shen Z. y Ding L., 1989. Fundamentals of Fuzzy Prolog. *International Journal Approximate Reasoning*, 3(2):179–193.

Ng R. y Subrahmanian V.S., 1991. Stable Model Semantics for Probabilistic Deductive Databases. En *Proc. of the 6th International Symposium on Methodologies for Intelligent Systems, ISMIS'91*, págs. 162–171. Springer-Verlag, Charlotte, North Carolina.

Ng R. y Subrahmanian V.S., 1992. Probabilistic logic programming. *Information and Computation*, 101(2):150–201.

Ngo L. y Haddawy P., 1997. Answering queries from context-sensitive probabilistic knowledge bases. *Theoretical Computer Science*, 171(1-2):147–177.

Nguyen H.T., 2002. *A First Course in Fuzzy and Neural Control*. CRC Press, Inc., Boca Raton, Florida.

Nguyen H.T. y Walker E., 2006. *A First Course in Fuzzy Logic.* Chapman & Hall/CRC, Boca Ratón, Florida. Third edition.

Nicolas P., Garcia L., y Stéphan I., 2005. Possibilistic stable models. En *Proc. of the 19th International Joint Conference on Artificial Intelligence, IJCAI'05*, págs. 248–253. Morgan Kaufmann, San Francisco.

Novak V., Perfilieva I. y Mockor J., 1999. *Mathematical principles of fuzzy logic.* Kluwer, Boston, Dordrecht.

Pajares G. y Santos M., 2005. *Inteligencia Artificial e Ingeniería del Conocimiento.* Ra-Ma.

Palamidessi C., 1990. Algebraic Properties of Idempotent Substitutions. En M.S. Paterson, ed., *Proc. of 17th International Colloquium on Automata, Languages and Programming*, págs. 386–399. Lecture Notes in Computer Science 443.

Palmn R., Driankov D. y Hellendoorn H., 1997. *Model-Based Fuzzy Control.* Springer-Verlag.

Pavelka J., 1979. On fuzzy logic I, II, III. *Zeitschrift fur Mathemathische Logik und Grundlagen der Mathematik*, 25:45–52, 119–134, 447–464.

Pedrycz W. y Gomide F., 1998. *Introduction to fuzzy sets.* MIT Press, Cambridge, Massachusetts.

Pettorossi A. y Proietti M., 1994. Transformation of Logic Programs: Foundations and Techniques. *Journal of Logic Programming*, 19,20:261–320.

Pettorossi A. y Proietti M., 1996a. A Comparative Revisitation of Some Program Transformation Techniques. En O. Danvy, R. Glück y P. Thiemann, eds., *Partial Evaluation, International Seminar, Dagstuhl Castle*, págs. 355–385. Lecture Notes in Computer Science 1110.

Pettorossi A. y Proietti M., 1996b. Rules and Strategies for Transforming Functional and Logic Programs. *ACM Computing Surveys*, 28(2):360–414.

Pettorossi A. y Proietti M., 1998. Transformation of Logic Programs. En *Handbook of Logic in Artificial Intelligence*, volumen 5, págs. 697–787. Oxford University Press.

Proietti M. y Pettorossi A., 1993. The Loop Absorption and the Generalization Strategies for the Development of Logic Programs and Partial Deduction. *Journal of Logic Programming*, 16(1&2):123–161.

Rios-Filho L.G. y Sandri S.A., 1995. Contextual Fuzzy Unification. En *Proc. 5th International Fuzzy Systems Association World Congress IFSA'95, Sao Paulo*, págs. 81–84.

Rodríguez-Artalejo M. y Romero-Díaz C.A., 2008. Quantitative Logic Programming Revisited. En *Proc. of the 9th International Symposium on Functional and Logic Programming, FLOPS'08*, volumen 4989, págs. 272–288. Springer-Verlag, Lecture Notes in Computer Science.

Rodríguez-Artalejo M. y Romero-Díaz C.A., 2009. Qualified Logic Programming with Bivalued Predicates. *Electronic Notes in Theoretical Computer Science*, 248:67–82.

Rounds W. y Zhang G.Q., 2001. Clausal Logic and Logic Programming in Algebraic Domains. *Information and Computation*, 171:183–200.

Sands D., 1995. Higher Order Expression Procedures. En *Proc. of the 1995 ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation*, págs. 178–189. ACM Press, New York.

Sands D., 1996. Total Correctness by Local Improvement in the Transformation of Functional Programs. *ACM Transactions on Programming Languages and Systems*, 18(2):175–234.

Scherlis W., 1981. Program Improvement by Internal Specialization. En *Proc. of 8th Annual ACM Symposium on Principles of Programming Languages*, págs. 41–49. ACM Press, New York.

Schweizer B. y Sklar A., 1983. *Probabilistic Metric Spaces*. North-Holland, New York.

Scott D.S., 1982. Domains for denotational semantics. *Lecture Notes in Computer Science*, 140:577–610.

Sessa M.I., 2000. Flexible Querying in Deductive Database. En A.D. Nola y G. Gerla, eds., *School on Soft Computing at Salerno University: Selected Lectures 1996-1999*, págs. 257–276. Springer Verlag.

Sessa M.I., 2002. Approximate reasoning by similarity-based SLD resolution. *Fuzzy Sets and Systems*, 275:389–426.

Shafer G., 1976. *A Mathematical Theory of Evidence*. Princeton University Press.

Shapiro E.Y., 1983. Logic programs with uncertainties: A tool for implementing rule-based systems. En *Proc. of the 8th International Joint Conference on Artificial Intelligence, IJCAI'83, Karlsruhe*, págs. 529–532.

Shortliffe E.H. y Buchanan B.G., 1975. A model of inexact reasoning in medicine. *Mathematical Biosciences*, 23:351–379.

Sterling L. y Shapiro E., 1986. *The Art of Prolog*. MIT Press, Cambridge.

Stouti A., 2004. A fuzzy version of Tarski's fixpoint theorem. *Archivum Mathematicum*, 40(3):273–279.

Straccia U., 2005a. Query Answering in Normal Logic Programs Under Uncertainty. En L. Godó, ed., *Proc. of 8th. European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty, ECSQARU'05, Barcelona*, págs. 687–700. Lecture Notes in Computer Science 3571.

Straccia U., 2005b. Uncertainty Management in Logic Programming: Simple and Effective Top-Down Query Answering. En R. Khosla, R.J. Howlett y L.C. Jain, eds., *Proc. 9th International Conference on Knowledge-Based and Intelligent Information and Engineering Systems, Part II*, págs. 753–760. Lecture Notes in Computer Science 3682.

Straccia U., Ojeda-Aciego M. y Damásio C.V., 2009. On Fixed-Points of Multivalued Functions on Complete Lattices and Their Application to Generalized Logic Programs. *SIAM Journal on Computing*, 38(5):1881–1911.

Swift T., 1999. Tabling for non-monotonic programming. *Annals of Mathematics and Artificial Intelligence*, 25(3-4):201–240.

Tamaki H. y Sato T., 1984. Unfold/Fold Transformations of Logic Programs. En S. Tärnlund, ed., *Proc. of 2th International Conference on Logic Programming*, págs. 127–139.

Tamaki H. y Sato T., 1986. OLD Resolution with Tabulation. En *Proc. of the 3th International Conference on Logic Programming*, págs. 84–98. Springer-Verlag, London.

Tarski A., 1955. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309.

Terricabras J.M. y Trillas E., 1989. Some Remarks on Vague Predicates. *Theoria*, 10:1–12.

Trillas E., Alsina C. y Terricabras J.M., 1995. *Introducción a la lógica borrosa*. Ariel, S.A., Barcelona.

Trillas E., del Campo C. y Cubillo S., 2000. When QM-Operators Are Implication Functions and Conditional Fuzzy Relations. *International Journal of Intelligent Systems*, 15:647–655.

Trillas E. y Valverde L., 1985. On mode and implication in approximate reasoning. En M.M. Gupta, A. Kandel, W. Bandler y J. Kiszka, eds., *Approximate reasoning in expert systems*. North Holland, Elsevier Science Publishers B. V.

Turchin V., 1986. The Concept of a Supercompiler. *ACM Transactions on Programming Languages and Systems*, 8(3):292–325.

Turksen I.B., 1992. Interval-valued fuzzy sets and "compensatory AND". *Fuzzy Sets Systems*, 51(3):295–307.

Valverde L. y Zadeh L.A., 1996. Del control analítico al control borroso. *Universitat de les Illes Balears*.

Vaucheret C., Guadarrama S. y Muñoz S., 2002. Fuzzy Prolog: A Simple General Implementation Using $CLP(R)$. En M. Baaz y A. Voronkov, eds., *Proc. of Logic for Programming, Artificial Intelligence and Reasoning, LPAR'02, Tbilisi*, págs. 450–463. Lectures Notes in Artificial Intelligence 2514.

Verbaeten S., Schreye D.D. y Sagonas K., 2001. Termination proofs for logic programs with tabling. *ACM Transactions on Computational Logic*, 2(1):57–92.

Vidal G., 2007. Quasi-Terminating Logic Programs for Ensuring the Termination of Partial Evaluation. En *Proc. of the ACM SIGPLAN 2007 Workshop on Partial Evaluation and Program Manipulation, PEPM'07*, págs. 51–60. ACM Press.

Virtanen E., 1991. Fuzzy Unification. En *Proc. of the International Conference on Information Processing and Management of Uncertainty in Knoledge-based Systems, IPMU'94, Paris*, págs. 1147–1152.

Vojtáš P., 2001. Fuzzy Logic Programming. *Fuzzy Sets and Systems, Elsevier*, 124(1):361–370.

Vojtáš P. y Paulík L., 1996. Soundness and completeness of non-classical extended SLD-resolution. En R. Dyckhoff y al, eds., *Proc. of the Workshop on Extensions of Logic Programming, ELP'96, Leipzig*, págs. 289–301. Lecture Notes in Computer Science 1050.

Wadler P., 1990. Deforestation: Transforming programs to eliminate trees. *Theoretical Computer Science*, 73:231–248.

Warren D.H.D., 1983. An Abstract Prolog Instruction Set. Technical note 309, SRI International, Menlo Park, California.

Weigert T.J., Tsai J.J.P. y Liu X., 1993. Fuzzy Operator Logic and Fuzzy Resolution. *Journal of Automated Reasoning*, 10(1):59–78.

Wüthrich B., 1995. Probabilistic Knowledge Bases. *IEEE Transactions on Knowledge and Data Engineering*, 7(5):691–698.

Yager R.R., 1988. On ordered weighted averaging aggregation operators in multicriteria decision making. *IEEE Transactions on Systems, Man and Cybernetics*, 18(1):183–190.

Yager R.R., 1993a. Families of OWA operators. *Fuzzy Sets and Systems*, 59(2):125–148.

Yager R.R., 1993b. Fuzzy Screening Systems. En R. Owen y M. Roubens, eds., *Fuzzy logic: state of the art*, págs. 251–261. Kluwer Academic Publishers, Dordrecht.

Yager R.R., 1994a. Aggregation operators and fuzzy systems modeling. *Fuzzy Sets and Systems*, 67(2):129–145.

Yager R.R., 1994b. On weighted median aggregation. *International Journal of Uncertainty*, 2:101–113.

Yasui H., Hamada Y. y Mukaidono M., 1995. Fuzzy prolog based on Łukasiewicz implication and bounded product. En *Proc. of the 4th IEEE International Conference on Fuzzy Systems, Yokohama*, volumen 2, págs. 949–954.

Ying M., 2002. Implication operators in fuzzy logic. *IEEE Transactions on Fuzzy Systems*, 10:88–91.

Zadeh L.A., 1965a. Fuzzy logic and approximate reasoning. *Synthese*, 30:407–428.

Zadeh L.A., 1965b. Fuzzy Sets. *Information and Control*, 8(3):338–353.

Zadeh L.A., 1973. Outline of a new approach to the analysis of complex systems and decision processes. *IEEE Transactions on Systems, Man and Cybernetics*, 3(1).

Zadeh L.A., 1975. The concept of a Linguistic Variable and Its Applications to Aproximate Reasoning. *Information Sciences*, págs. 199–249.

Zadeh L.A., 1996. Nacimiento y evolución de la lógica borrosa, el soft computing y la computación con palabras: un punto de vista personal. *Psicothema*, 8(2):421–429.

Zadeh L.A., 2008. Is there a need for fuzzy logic? *Information Sciences*, 178:2751–2779.

Zhu H., 1994. How powerful are folding/unfolding transformations. *Journal of Functional Programming*, 4(1):89–112.

Zimmermann H. y Zysno P., 1980. Latent Connectives in Human Decision Making. *Fuzzy Sets and Systems*, 4:37–51.