Programación Lógica Cuantitativa y su Implementación en $\mathcal{TOY}$

Carlos A. Romero Díaz

Dpto. de Sistemas Informáticos y Computación
FACULTAD DE INFORMÁTICA
Universidad Complutense de Madrid

Septiembre 2007

Dirigido por: Mario Rodríguez Artalejo

MÁSTER EN INVESTIGACIÓN INFORMÁTICA
Especialidad en Programación y Tecnología del Software

Agradecimientos

Quisiera agradecer especialmente la colaboración de *Rafael Caballero* en la elaboración en particular del capítulo 6 por sus explicaciones sobre el diseño y estructura del sistema $\mathcal{TOY}$ sin las cuales no hubiera podido realizar el prototipo de implementación del lenguaje propuesto a lo largo del trabajo.

Y agradecer también a *Mario*, mi director de proyecto, la dedicación mostrada en sus explicaciones y revisiones que me han permitido terminar, con mejor o peor fortuna, este trabajo.

Madrid, 13 de septiembre de 2007

Índice general

Capítulo 1

Introducción

Desde hace varias décadas, el tratamiento del conocimiento incierto ha ido haciéndose necesario en diferentes áreas de la ciencia. En particular, son múltiples los resultados aportados desde el campo de la Inteligencia Artificial y la Programación Lógica. Estos resultados pueden englobarse en diferentes enfoques según la manera de tratar con la incertidumbre.

La Programación Lógica Cuantitativa es uno de estos enfoques y, dentro de la Programación Lógica con Incertidumbre, constituye el más antiguo de cuantos se han empleado para introducir el conocimiento incierto y la incertidumbre en los lenguajes de lógicos de programación. Este trabajo surge a partir de la idea de realizar una actualización y revisión de los principales resultados de la Programación Lógica Cuantitativa (aquellos propuestos entre finales de los años 80 y comienzos de los 90) con especial énfasis en el artículo de *van Emden* [61] con el fin de obtener un lenguaje lógico cuantitativo de semántica rigurosa del que podamos partir para poder trasladar, posteriormente, algunas de estas ideas a otros lenguajes declarativos multiparadigma.

El presente trabajo se divide en 7 capítulos en el que cada uno de ellos presenta el siguiente contenido:

Capítulo 1 Introducción general al trabajo y a cada uno de los capítulos que forman este trabajo.

Capítulo 2 Repaso a los fundamentos de la Lógica de Primer Orden y a los Conjuntos Ordenados, CPOs y Retículos con el fin de que sirva de recordatorio de todos aquellos conocimientos necesarios para el correcto entendimiento del resto de los capítulos.

Capítulo 3 Aporta una visión general de los fundamentos de la programación lógica clásica de forma que la extensión con incertidumbre realizada en este trabajo pueda ser comparada y puedan observarse las diferencias que surgen como resultado de la introducción de la incertidumbre.

Capítulo 4 Breve panorámica de la incertidumbre en la programación lógica hasta nuestros días dando en cada momento las referencias necesarias para seguir el desarrollo que ha tenido lugar a lo largo de la historia.

Capítulo 5 Este capítulo constituye, junto con el siguiente, el núcleo central del trabajo.

Aquí se aporta la sintaxis y semántica, declarativa y operacional, de un lenguaje lógico cuantitativo (que llamaremos QLP) y se aporta un método de resolución de objetivos que se comprobará correcto y completo.

Capítulo 6 Partiendo del lenguaje construido en el capítulo anterior, se construye un traductor de dicho lenguaje a $\mathcal{TOY}$ y se extiende el intérprete de $\mathcal{TOY}$ mediante una serie de comandos que facilitan la carga de programas cuantitativos y la resolución de objetivos cuantitativos.

Capítulo 7 Resumen de los objetivos conseguidos con este trabajo y breve aproximación a las posibles líneas de trabajo futuro.

Además de los capítulos citados anteriormente, el trabajo incluye un apéndice con el código fuente tanto de los ejemplos empleados en el capítulo 5 como del traductor implementado en el capítulo 6 para que pueda servir como orientación a las explicaciones referentes al código que aparecen especialmente en el capítulo 6.

Capítulo 2

Preliminares

Para poder entender los capítulos sucesivos, se hace necesario conocer ciertos detalles tanto de la lógica de primer orden como de conjuntos ordenados, CPOs y retículos que serán empleados a lo largo del trabajo. Es por ello objetivo de este segundo capítulo introducir ambos temas con la suficiente profundidad, aunque sin excesivo detenimiento, para permitir el desarrollo posterior.

2.1. Lógica de Primer Orden

La programación lógica tiene su base fundamental en la Lógica de Primer Orden (LPO en adelante). A continuación se realiza, a modo de resumen, una introducción a la LPO indicando todos aquellos aspectos que es necesario conocer de antemano para abordar la definición de la sintaxis y la semántica de lenguajes de programación lógica con incertidumbre como los considerados en este trabajo. Para una referencia más completa sobre la LPO se puede acudir a libros generales sobre la lógica matemática como [18] u otros más específicos para el campo de la informática como [9, 25, 53] o incluso a la primera sección de [1].

2.1.1. Sintaxis

La definición de un lenguaje lógico requiere de la existencia de un alfabeto que contenga todos los símbolos necesarios para formalizar enunciados.

Lo primero es construir una signatura $\Sigma = \langle FS_\Sigma, PS_\Sigma \rangle$ como la unión disjunta de un conjunto de símbolos de función (FS_Σ) y un conjunto de símbolos de predicado (PS_Σ). Llamaremos FS^n_Σ al conjunto de todos los símbolos de función de aridad $n \geq 0$, y definiremos FS_Σ como $FS_\Sigma = \biguplus_{n\in\mathbb{N}} FS^n_\Sigma$ siendo $\uplus$ la unión disjunta. Análogamente, definiremos PS_Σ como $PS_\Sigma = \biguplus_{n\in\mathbb{N}} PS^n_\Sigma$, donde PS^n_Σ es el conjunto de todos los símbolos de predicado de aridad $n \geq 0$. Asociaremos además, FS^0_Σ con el conjunto de las constantes y PS^0_Σ con el conjunto de los símbolos de proposición.

Necesitaremos también un conjunto de símbolos lógicos que nos permita realmente formalizar nuestros enunciados. Estos símbolos serán las conectivas proposicionales $\bot$ (*false*), $\top$ (*true*), $\neg$ (*not*), $\wedge$ (*and*), $\vee$ (*or*), $\rightarrow$ (*if-then*) y $\leftrightarrow$ (*if and only if*), los cuantificadores $\forall$ (*uni-*

versal) y $\exists$ (*existencial*) y el símbolo lógico para la igualdad ($\approx$). Además, emplearemos otros símbolos auxiliares como los paréntesis y llamaremos $\mathcal{V}$ al conjunto de todas las variables.

Para poder saber a qué tipo de símbolo nos referimos en cada momento sin necesidad de decirlo expresamente, utilizaremos c para referirnos a las constantes; f, g, etc. para referirnos a los símbolos de función; p, q, etc. para los símbolos de proposición y de predicado (que distinguiremos según sea su aridad); y x, y, etc. para referirnos a las variables.

Términos

Los términos, son aquellas combinaciones de símbolos de función, constantes y variables (junto con algunos otros símbolos del alfabeto) con sentido. Más exactamente, un término se define de manera recursiva de la siguiente forma:

1. Una variable $x \in \mathcal{V}$ es un término.
2. Una constante $c \in FS^0_\Sigma$ es un término.
3. Si $f \in FS^n_\Sigma$ es una función de aridad $n > 0$ y $t_1, \ldots, t_n$ son términos, entonces $f(t_1, \ldots, t_n)$ es un término.

Llamaremos $Term_\Sigma$ al conjunto de todos los términos.

Fórmulas

Las fórmulas se definen también de modo recursivo de la siguiente forma:

1. $\bot$ y $\top$ son fórmulas.
2. Un símbolo de proposición $p \in PS^0_\Sigma$ es una fórmula.
3. Si $p \in PS^n_\Sigma$ es un predicado de aridad $n > 0$ y $t_1, \ldots, t_n$ son términos, entonces $p(t_1, \ldots, t_n)$ es una fórmula.
4. Si $t_1, t_2 \in Term_\Sigma$, entonces $(t_1 \approx t_2)$ es una fórmula.
5. Si F es una fórmula, entonces $\neg F$ es una fórmula.
6. Si F y G son fórmulas, entonces lo son también $(F \wedge G)$, $(F \vee G)$, $(F \to G)$ y $(F \leftrightarrow G)$.
7. Si F es una fórmula y $x \in \mathcal{V}$ es una variable entonces $\exists x F$ y $\forall x F$ son fórmulas.

Escribiremos F, G, ... para referirnos a una fórmula en general, y A, B, ... para referirnos a las fórmulas a las que hacen referencia los puntos 1, 2, 3 y 4 que llamaremos *fórmulas atómicas* o *átomos*.

Del conjunto de variables que aparecen en una fórmula, diremos que están *ligadas* aquellas que aparecen cuantificadas en la fórmula, y *libres* las que aparezcan sin cuantificar y llamaremos $bv(F)$ y $fv(F)$ a los conjuntos de variables ligadas y libres, respectivamente, de F. Si F es una fórmula y las variables $x_1, \ldots, x_n$ se encuentran libres en F, podemos escribir $\exists \overline{x} F$ en

vez de $\exists x_1, \ldots, \exists x_n F$, y $\forall \overline{x} F$ en vez de $\forall x_1, \ldots, \forall x_n F$. Cuando $\{x_1, \ldots, x_n\}$ coincida con el conjunto de variables libres de F, pondremos $F^{\exists}$ (que llamaremos *cierre existencial* de F) en vez de $\exists \overline{x} F$ y $F^{\forall}$ (*cierre universal* de F) en vez de $\forall \overline{x} F$. Un término o fórmula sin variables se dice que es *cerrado*.

Cláusulas

Si convenimos, como es costumbre en lógica, en llamar *literal* a cualquier fórmula A o $\neg A$, tal que A sea una fórmula atómica, llamaremos *cláusula* a cualquier fórmula de la forma $(L_1 \vee \ldots \vee L_m)^{\forall}$ siendo $L_1, \ldots, L_n$ literales. Emplearemos una forma especial para escribir estas cláusulas que llamaremos *forma clausal*; y la cláusula anterior escrita en forma clausal será

$$A_1, \ldots, A_k \leftarrow B_1, \ldots, B_n$$

donde $A_1, \ldots, A_k$ es a lista de todos los literales positivos de entre $L_1, \ldots, L_m$ que llamaremos conclusiones, y $B_1, \ldots, B_m$ es la lista de todos los demás literales una vez eliminado el símbolo de negación, que llamaremos premisas. De modo informal, podremos entender la cláusula como (A_1 o ... o A_k) si (B_1 y ... y B_n).

En particular, para los programas lógicos considerados en este trabajo, nos interesarán sólo aquellas cláusulas para las que $k = 1$, denominadas *cláusulas definidas* o *de Horn*, pero dado que únicamente emplearemos este tipo de cláusulas, nos referiremos a ellas de manera general como cláusula o regla de programa. Por esto, las cláusulas serán de la forma

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

siendo $A, B_1, \ldots, B_n$ fórmulas atómicas que no sean de la forma $\bot, \top$ ni $(t_1 \approx t_2)$. Para representar una cláusula definida escribiremos también

$$A \leftarrow \overline{B}$$

que se identifica con la fórmula $(B_1 \wedge \ldots \wedge B_n \rightarrow A)^{\forall}$ o también $(A \vee \neg B_1 \vee \ldots \vee \neg B_n)^{\forall}$. De forma general, usaremos la letra C para denotar una cláusula.

2.1.2. Semántica

Hasta ahora, tenemos términos y fórmulas sin ningún significado especial asociado a ellas. Es objeto de la semántica darles significado para que podamos saber qué expresan exactamente y podamos emplearlos en un contexto dado. Para poder dar significado a estos términos y fórmulas, se utilizan estructuras e interpretaciones.

Estructuras

Una Σ-estructura es de la forma $\mathcal{A} = (A, \{f^{\mathcal{A}}\}_{f \in FS_\Sigma}, \{p^{\mathcal{A}}\}_{p \in PS_\Sigma})$ siendo:

- A, un conjunto no vacío llamado *universo* o *dominio* de $\mathcal{A}$.
- $c^{\mathcal{A}} \in A$, la interpretación de la constante $c \in FS^0_\Sigma$ en $\mathcal{A}$.

- $f^{\mathcal{A}} : A^n \to A$, una función que asigne un elemento del universo a cada función de $f \in FS^n_\Sigma$ con $n > 0$.
- $p^{\mathcal{A}} \in \{0, 1\}$, la interpretación del símbolo proposicional $p \in PS^0_\Sigma$.
- $p^{\mathcal{A}} \subseteq A^n$, una relación para cada símbolo de predicado $p \in PS^n_\Sigma$ con $n > 0$.

En general, dada una Σ-estructura $\mathcal{A}$, escribiremos

$f^{\mathcal{A}}(a_1, \ldots, a_n)$ o simplemente $f^{\mathcal{A}}(\overline{a})$ para indicar la imagen de $f^{\mathcal{A}} : A^n \to A$ correspondiente a la n-tupla de argumentos $\overline{a} \in A^n$.

$p^{\mathcal{A}}(a_1, \ldots, a_n)$ o simplemente $p^{\mathcal{A}}(\overline{a})$ para indicar que $\overline{a} \in p^{\mathcal{A}}$, es decir, que la n-tupla $\overline{a} \in A^n$ verifica la relación $p^{\mathcal{A}}$ que $\mathcal{A}$ nos ofrece como significado del símbolo de predicado p.

Interpretaciones

Una Σ-interpretación $\langle \mathcal{A}, \varepsilon \rangle$ está compuesta de una Σ-estructura $\mathcal{A}$ y de un estado ε para las variables. Un *estado* es una aplicación $\varepsilon : \mathcal{V} \to A$ de variables en elementos del universo de $\mathcal{A}$. Si ε es un estado, a un elemento del dominio de $\mathcal{A}$, y x una variable, $\varepsilon[x/a]$ es el estado que hace corresponder a a x y coincide con ε en todas las demás variables:

$$\varepsilon[a/x](y) =_{def} \begin{cases} \varepsilon(y) & \text{si } y \neq x \\ a & \text{si } y = x \end{cases}$$

La Σ-interpretación nos da información suficiente para poder interpretar cada término t de $Term_\Sigma$ como un individuo del universo ($[\![t]\!]^{\mathcal{A},\varepsilon} \in A$) y cada fórmula F como un valor veritativo ($[\![F]\!]^{\mathcal{A},\varepsilon} \in \{0, 1\}$). Mediante la notación anterior indicamos que la interpretación de los términos y las fórmulas dependen tanto de la estructura como del estado.

1. Interpretación de un término

   Una Σ-interpretación $\langle \mathcal{A}, \varepsilon \rangle$ hace corresponder a cada término t el valor $[\![t]\!]^{\mathcal{A},\varepsilon}$ que se define recursivamente según sea la forma de t:

   - $[\![x]\!]^{\mathcal{A},\varepsilon} = \varepsilon(x)$, siendo $x \in \mathcal{V}$.
   - $[\![c]\!]^{\mathcal{A},\varepsilon} = c^{\mathcal{A}}$, siendo $c \in FS^0_\Sigma$.
   - $[\![f(t_1, \ldots, t_n)]\!]^{\mathcal{A},\varepsilon} = f^{\mathcal{A}}([\![t_1]\!]^{\mathcal{A},\varepsilon}, \ldots, [\![t_n]\!]^{\mathcal{A},\varepsilon})$, siendo $f \in FS^n_\Sigma$ con $n > 0$ y $t_1, \ldots, t_n \in Term_\Sigma$.

2. Interpretación de una fórmula

   Y, al igual que con los términos una Σ-interpretación les hace corresponder un valor, a la fórmula F le hace corresponder un valor $[\![F]\!]^{\mathcal{A},\varepsilon}$, en este caso veritativo, que se define recursivamente según sea la forma de F:

   - $[\![\bot]\!]^{\mathcal{A},\varepsilon} = 0$.
   - $[\![\top]\!]^{\mathcal{A},\varepsilon} = 1$.

- $[\![p]\!]^{\mathcal{A},\varepsilon} = p^{\mathcal{A}} \in \{0,1\}$, siendo $p \in PS^0_\Sigma$.
- $[\![p(t_1,\ldots,t_n)]\!]^{\mathcal{A},\varepsilon} = \begin{cases} 1 & \text{si } p^{\mathcal{A}}([\![t_1]\!]^{\mathcal{A},\varepsilon},\ldots,[\![t_n]\!]^{\mathcal{A},\varepsilon}) \\ 0 & \text{si no } p^{\mathcal{A}}([\![t_1]\!]^{\mathcal{A},\varepsilon},\ldots,[\![t_n]\!]^{\mathcal{A},\varepsilon}) \end{cases}$, siendo $p \in FS^n_\Sigma$ con $n > 0$ y $t_1,\ldots,t_n \in Term_\Sigma$.
- $[\![(t_1 \approx t_2)]\!]^{\mathcal{A},\varepsilon} = \begin{cases} 1 & \text{si } [\![t_1]\!]^{\mathcal{A},\varepsilon} = [\![t_2]\!]^{\mathcal{A},\varepsilon} \\ 0 & \text{en otro caso} \end{cases}$, siendo $t_1, t_2 \in Term_\Sigma$.
- $[\![\neg F]\!]^{\mathcal{A},\varepsilon} = v_\neg([\![F]\!]^{\mathcal{A},\varepsilon})$ donde $v_\neg : \{0,1\} \to \{0,1\}$ calcula un valor veritativo, a partir del valor veritativo de $[\![F]\!]^{\mathcal{A},\varepsilon}$, definido según la siguiente tabla (que se corresponde con la tabla de verdad para la conectiva $\neg$ de la lógica matemática):

	$v_\neg$
0	1
1	0

- $[\![(F \diamond G)]\!]^{\mathcal{A},\varepsilon} = v_\diamond([\![F]\!]^{\mathcal{A},\varepsilon}, [\![G]\!]^{\mathcal{A},\varepsilon})$, donde $\diamond \in \{\wedge, \vee, \to, \leftrightarrow\}$ y $v_\diamond : \{0,1\}^2 \to \{0,1\}$ calcula un valor veritativo, según sean los valores veritativos de $[\![F]\!]^{\mathcal{A},\varepsilon}$ y $[\![G]\!]^{\mathcal{A},\varepsilon}$, tal y como se define en la siguiente tabla (que se corresponden con las tablas de verdad empleadas comúnmente en lógica matemática para las conectivas $\wedge$, $\vee$, $\to$ y $\leftrightarrow$):

		$v_\wedge$	$v_\vee$	$v_\to$	$v_\leftrightarrow$
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	0	0
1	1	1	1	1	1

- $[\![\forall x F]\!]^{\mathcal{A},\varepsilon} = min\{[\![F]\!]^{\mathcal{A},\varepsilon[x/a]} \mid a \in A\} = \begin{cases} 1 & \text{si } [\![F]\!]^{\mathcal{A},\varepsilon[x/a]} = 1 \text{ para todo } a \in A \\ 0 & \text{en otro caso} \end{cases}$
- $[\![\exists x F]\!]^{\mathcal{A},\varepsilon} = max\{[\![F]\!]^{\mathcal{A},\varepsilon[x/a]} \mid a \in A\} = \begin{cases} 1 & \text{si } [\![F]\!]^{\mathcal{A},\varepsilon[x/a]} = 1 \text{ para algún } a \in A \\ 0 & \text{en otro caso} \end{cases}$

En lo sucesivo, y dado que los lenguajes de programación lógica objeto de este trabajo no contemplan la igualdad, siempre que se haga referencia a la lógica, se hará en relación a la lógica sin igualdad.

Modelo y Consecuencia Lógica

Cuando ocurre que $[\![F]\!]^{\mathcal{A},\varepsilon} = 1$ se dice que $\mathcal{A}$ *satisface* a F en el estado ε, o que la interpretación $\langle \mathcal{A}, \varepsilon \rangle$ es *modelo* de F. Si $[\![F]\!]^{\mathcal{A},\varepsilon} = 0$, se dice que $\mathcal{A}$ no satisface F en el estado ε, o que $\langle \mathcal{A}, \varepsilon \rangle$ no es modelo de F. Se pone también $\langle \mathcal{A}, \varepsilon \rangle \models F$ cuando $[\![F]\!]^{\mathcal{A},\varepsilon} = 1$.

Si extendemos esta idea a un conjunto de fórmulas Φ, se pone $\langle \mathcal{A}, \varepsilon \rangle \models \Phi$ cuando $\langle \mathcal{A}, \varepsilon \rangle \models F$ para toda fórmula $F \in \Phi$, y se dice que $\langle \mathcal{A}, \varepsilon \rangle$ es modelo de Φ.

Dados un conjunto de fórmulas Φ y una fórmula F. Si cualquier interpretación $\langle \mathcal{A}, \varepsilon \rangle$ que sea modelo de Φ cumple que es también modelo de F, entonces se dice que F es consecuencia lógica de Φ y se escribe $\Phi \models F$ (empleamos el mismo símbolo que cuando una interpretación es modelo de un conjunto de fórmulas, distinguiéndolo según tengamos a su izquierda una

interpretación o un conjunto de fórmulas). Formalmente: $\Phi \models F \Leftrightarrow_{def} \forall \langle \mathcal{A}, \varepsilon \rangle \ (\langle \mathcal{A}, \varepsilon \rangle \models \Phi \Rightarrow \langle \mathcal{A}, \varepsilon \rangle \models F)$.

2.2. Sustituciones

Llamaremos *sustitución* a una aplicación $\sigma : \mathcal{V} \to Term_\Sigma$ que asigna un término a cada variable, y diremos que $Sust_\Sigma$ es el conjunto de todas las sustituciones. Definiremos el dominio de una sustitución de la siguiente forma $dom(\sigma) =_{def} \{x \in \mathcal{V} \mid \sigma(x) \neq x\}$. Definimos también el rango de una sustitución como $ran(\sigma) =_{def} \{t \in Term_\Sigma \mid \sigma(x) = t, x \in dom(\sigma)\}$, y definiremos una función especial *vran* que nos dará el conjunto de variables que aparecen en los términos que conforman el rango de la sustitución, así: $vran(\sigma) =_{def} \bigcup\{var(\sigma(x)) \mid x \in dom(\sigma)\}$. Diremos que σ es finita si y sólo si $dom(\sigma)$ es finito.

Dadas dos sutituciones σ y σ' y un conjunto de variables $\mathcal{X} \subseteq \mathcal{V}$, usaremos la notación $\sigma = \sigma' \ [\mathcal{X}]$ para indicar que $\sigma(x) = \sigma'(x)$ para cualquier $x \in \mathcal{X}$. Usaremos también la notación $\sigma \preccurlyeq \sigma' \ [\mathcal{X}]$ para indicar la existencia de una tercera sustitución μ tal que $\sigma\mu = \sigma' \ [\mathcal{X}]$. En este caso se dice que σ es al menos tan general como σ' sobre el conjunto de variables $\mathcal{X}$.

Una sustitución puede aplicarse tanto a términos como a fórmulas, aunque el proceso de aplicación difiere en ambos casos:

1. Aplicación de una sustitución a un término

   Dado un término t y una sustitución σ, $\sigma(t)$ se calcula sustituyendo simultáneamente todas las variables de t por su correspondiente término según especifica la sustitución σ.

   Ejemplo 2.2.1. Sean $t = f(x, y)$ y $\sigma = \{x \mapsto y, y \mapsto g(z)\}$. Entonces, $\sigma(t) = f(y, g(z))$.

2. Aplicación de una sustitución a una fórmula

   Dada una fórmula F, $\sigma(F)$ se calcula siguiendo el siguiente proceso:

   a) Se renombran las variables ligadas en F de forma que $bv(F) \cap vran(\sigma) = \emptyset$.
   b) Se sustituyen simultáneamente todas las apariciones de $x \in fv(F) \cap dom(\sigma)$ por $\sigma(x)$.

   Ejemplo 2.2.2. Sean $F = \forall y.p(x, y)$ y $\sigma = \{x \mapsto f(y)\}$. Entonces, $\sigma(F) = \forall z.p(f(y), z)$.

Si convenimos en llamar *expresión* tanto a un término como a una fórmula, entonces dadas una expresión E y una sustitución σ, diremos que $\sigma(E)$ es una *instancia* de E; y si además no contiene variables libres, se dice que la instancia es *cerrada*.

Una sustitución σ que se comporte como una permutación de variables se llama renombramiento. Si σ es un renombramiento, $\sigma(E)$ se llama *variante* de E.

Composición de sustituciones

Dadas dos sustituciones σ y θ, se define la sustitución $\theta \circ \sigma$ formada por composición de ambas como la sustitución que dada una expresión E, aplica primero la sustitución σ, y al resultado le aplica la sustitución θ, es decir, $(\theta \circ \sigma)(E) =_{def} \theta(\sigma(E))$.

Sustitución idempotente

Se dice que una sustitución σ es *idempotente* cuando ocurre que $\sigma \circ \sigma = \sigma$, es decir, la aplicación sucesiva de una sustitución no produce cambios en las expresiones. Para que esto ocurra, es condición necesaria y suficiente que se cumpla $dom(\sigma) \cap vran(\sigma) = \emptyset$.

Notación. A fin de simplificar la notación a lo largo del trabajo, escribiremos $E\sigma$ en vez de $\sigma(E)$ cuando nos refiramos a la aplicación de una sustitución a una expresión. A consecuencia de esto, la composición de sustituciones será $\sigma\theta$ cuando queramos referirnos a $\theta \circ \sigma$ porque $(\theta \circ \sigma)(E) = \theta(\sigma(E)) = (E\sigma)\theta = E\sigma\theta$.

A continuación enunciamos y demostramos un lema que nos será necesario para demostraciones posteriores. En él se demuestra que la composición de dos sustituciones idempotentes es también idempotente siempre que sus dominios sean disjuntos y el dominio de la primera no incluya ninguna variable del rango de la segunda. Formalmente:

Lema 2.2.1. *Dadas dos sustituciones idempotentes σ y σ' tales que (1) $dom(\sigma') \cap dom(\sigma) = \emptyset$ y (2) $vran(\sigma') \cap dom(\sigma) = \emptyset$ se tiene que la composición $\sigma\sigma'$ es también idempotente (se cumple $\sigma\sigma' = \sigma\sigma'\sigma\sigma'$).*

Demostración. Queremos ver que se tiene $x\sigma\sigma'\sigma\sigma' = x\sigma\sigma'$ para toda variable $x \in \mathcal{V}$. Vemos primero que podemos deducir (3) $\sigma'\sigma\sigma' = \sigma'$ $[vran(\sigma)]$ de las hipótesis.

Sea $y \in vran(\sigma)$. Si $y \notin dom(\sigma')$, entonces $y\sigma'\sigma\sigma' = y\sigma\sigma' = y\sigma'$, donde la segunda igualdad resulta de $y \notin dom(\sigma)$ por ser σ idempotente e $y \in vran(\sigma)$. Y si $y \in dom(\sigma')$, entonces $y\sigma'\sigma\sigma' =_{(2)} y\sigma'\sigma' = y\sigma'$. Y queda demostrado (3).

Demostramos ahora la tesis del lema. Sea $x \in \mathcal{V}$ una variable cualquiera. Distinguimos tres casos:

- $x \notin dom(\sigma)$, $x \notin dom(\sigma')$: $x\sigma\sigma'\sigma\sigma' = x = x\sigma\sigma'$.
- $x \notin dom(\sigma)$, $x \in dom(\sigma')$: Entonces $var(x\sigma') \subseteq vran(\sigma')$ y $x\sigma\sigma'\sigma\sigma' = x\sigma'\sigma\sigma' =_{(2)} x\sigma'\sigma' = x\sigma' = x\sigma\sigma'$.
- $x \in dom(\sigma)$: Entonces $var(x\sigma) \subseteq vran(\sigma)$ y $x\sigma\sigma'\sigma\sigma' =_{(3)} x\sigma\sigma'$. □

Algoritmo de unificación de Robinson

Dados dos átomos A y B, se dice que σ es el *unificador más general* (u.m.g.) en el sentido de Robinson [51] si se cumplen las tres condiciones siguientes:

(1) $A\sigma \equiv B\sigma$ (σ es un unificador de A y B o σ unifica A y B).

(2) Para cualquier otro unificador θ de A y B, se verifica que $\theta = \sigma\theta$ (en particular, si $\theta = \sigma$ entonces se tiene $\sigma = \sigma\sigma$, o sea, σ idempotente).

(3) σ es relevante, es decir, $dom(\sigma) \subseteq var(A) \cup var(B)$.

El *Teorema de Unificación de Robinson* nos garantiza la existencia de un u.m.g. en el sentido de Robinson (único salvo variantes) entre dos átomos cualesquiera siempre que sean unificables, y el *Algoritmo de Unificación de Robinson* permite calcularlo. Para una presentación actualizada del algoritmo de unificación, se recomienda [1, 6, 40].

2.3. Conjuntos ordenados, CPOs y retículos

Sobre conjuntos ordenados necesitamos conocer ciertas definiciones para poder demostrar tres propiedades que nos interesarán, en particular, para dar semántica a nuestro lenguaje en capítulos sucesivos. A pesar de que podríamos retrasar estas propiedades para cuando nos hicieran falta, no harían más que complicar el desarrollo de la semántica de nuestro lenguaje pues se basan únicamente en conjuntos ordenados y pueden perfectamente desarrollarse de forma que sean autocontenidas, y a pesar de que pueden particularizarse para la programación lógica, consideramos beneficioso mantenerlas lo más generales posibles a fin de emplearlas como herramientas y no presentarlas como parte de los resultados. Como referencia auxiliar, se puede consultar [1, 40] para una aplicación directa a la programación lógica, donde se exponen estos resultados a lo largo del texto según se hacen necesarios; [31], u otros textos de Matemática Discreta, para una explicación más detallada y general sobre conjuntos ordenados; y [27, 54] sobre dominios semánticos para conocer el uso de CPOs en la descripción de la semántica de lenguajes de programación.

Un *orden parcial* es una relación binaria $\sqsubseteq$ sobre un conjunto D que sea reflexiva, antisimétrica y transitiva. Cuando nos referimos a este conjunto y a su orden parcial, se dice que constituyen un *conjunto parcialmente ordenado* que escribiremos como $(D, \sqsubseteq)$. Y si además, para cualesquiera dos elementos d_1 y d_2 de D, se tiene que o bien $d_1 \sqsubseteq d_2$, o bien $d_2 \sqsubseteq d_1$, se dice que $\sqsubseteq$ es un orden *total* sobre los elementos de D.

Dado un conjunto parcialmente ordenado $(D, \sqsubseteq)$ y una transformación $T : D \to D$, se dice que T es *monótona* si y solo si para cualesquiera dos elementos $d_1, d_2 \in D$ tales que $d_1 \sqsubseteq d_2$, se cumple que $T(d_1) \sqsubseteq T(d_2)$.

Definición 2.3.1. *Sean $(D, \sqsubseteq)$ un conjunto parcialmente ordenado y $T : D \to D$ una transformación monótona. Entonces un elemento d de D se llama:*

(i) Punto prefijo *de T, si su imagen es menor o igual en el orden parcial que d, es decir, $T(d) \sqsubseteq d$.*

(ii) Punto fijo *si su imagen es estrictamente igual a d, es decir, $T(d) = d$.*

Llamaremos $PPF(T)$ al conjunto de todos los puntos prefijos de T y $PF(T)$ al conjunto de todos los puntos fijos de T. Es fácil ver que existe una relación inmediata entre estos dos conjuntos, que es que $PF(T) \subseteq PPF(T)$, pues todo punto fijo es también punto prefijo.

El menor punto prefijo de una transformación monótona, en caso de existir, coincide con el menor punto fijo. Formalmente:

Proposición 2.3.1. *Sean $(D, \sqsubseteq)$ un conjunto parcialmente ordenado y $T : D \to D$ una transformación monótona. Si existe $d_0 = min(PPF(T))$, entonces $d_0 \in PF(T)$ y además $d_0 = min(PF(T))$.*

Demostración. Sea d_0 el menor punto prefijo de T. Por ser punto prefijo, sabemos que $T(d_0) \sqsubseteq d_0$, y por ser T una transformación monótona, $T(T(d_0)) \sqsubseteq T(d_0)$, por lo que $T(d_0)$ es también un punto prefijo de T. Como d_0 es el menor punto prefijo, se cumplirá que $d_0 \sqsubseteq T(d_0)$, por lo que se tiene que $T(d_0) = d_0$ y d_0 es también punto fijo. Ahora es trivial ver que d_0 es el menor punto fijo, pues $PF(T) \subseteq PPF(T)$. □

Si dado un conjunto parcialmente ordenado $(D, \sqsubseteq)$ se tiene que existen el máximo y el mínimo de D, y que para cualquier subconjunto X de D existen en D la cota superior mínima o supremo de X y la cota inferior máxima o ínfimo de X, se dice que $(D, \sqsubseteq)$ es un *retículo completo*. Escribiremos $\triangle$ para referirnos al mínimo de un retículo completo y $\triangledown$ para el máximo, por lo tanto $min(D) = \triangle$ y $max(D) = \triangledown$. Además, para cada $X \subseteq D$ escribiremos $\bigsqcup X$ para indicar el supremo de X e $\bigsqcap X$ para indicar el ínfimo de X. Obsérvese que $\bigsqcup \emptyset = \triangle$ e $\bigsqcap \emptyset = \triangledown$.

Mientras en el caso de tener un conjunto parcialmente ordenado no podíamos asegurar la existencia del menor punto [pre]fijo de una transformación T (a pesar de que si sabemos que sí existe, es también el menor punto fijo)[1], cuando nos encontramos con un retículo completo sí estamos en condiciones de decir que sí existe ese menor punto [pre]fijo, y además podemos caracterizarlo:

Proposición 2.3.2. *Si $(D, \sqsubseteq)$ es un retículo completo y $T : D \to D$ es una transformación monótona, se cumple que el ínfimo del conjunto de puntos prefijos de T es también el menor punto prefijo de T.*

Demostración. Por ser $PPF(T)$ un subconjunto de D, sabemos que existe $d_0 = \bigsqcap PPF(T)$. Sea d un punto prefijo cualquiera de T. Se tiene que $d_0 \sqsubseteq d$ y, por ser T una transformación monótona, también que $T(d_0) \sqsubseteq T(d)$. Como $T(d) \sqsubseteq d$, sabemos que $T(d_0) \sqsubseteq d$, por lo que se tiene que $T(d_0)$ es también cota inferior de $PPF(T)$, pero d_0 es, por definición del ínfimo, la mayor de las cotas inferiores, así que $T(d_0) \sqsubseteq d_0$ y $d_0 \in PPF(T)$. Por ello, $d_0 = min(PPF(T))$. □

Notación. Escribiremos $\mu(T)$ para referirnos al menor punto [pre]fijo de T.

Aparte de los retículos completos, otra clase interesante de conjuntos parcialmente ordenados son los CPOs (del inglés *Complete Partial Order*), que estudiaremos en el resto de esta sección.

Sea $(D, \sqsubseteq)$ un conjunto parcialmente ordenado. Se llama ω-cadena a todo subconjunto de D que sea de la forma $\{d_n \mid n \in \mathbb{N}\}$ y tal que $d_n \sqsubseteq d_{n+1}$ para todo $n \in \mathbb{N}$. Se llama ω-CPO con fondo a cualquier conjunto parcialmente ordenado que tenga un mínimo $\triangle = min(D)$ y tal que cualquier ω-cadena contenida en D tenga supremo $\bigsqcup_{n\in\mathbb{N}} d_n \in D$. Nótese que cualquier retículo completo es un ω-CPO con fondo. El recíproco es falso en general.

Sea $(D, \sqsubseteq)$ un ω-CPO con fondo. Una transformación monótona $T : D \to D$ se dice que es continua si para cualquier ω-cadena $\{d_n \mid n \in \mathbb{N}\} \subseteq D$ se cumple que $T(\bigsqcup_{n\in\mathbb{N}} d_n) = \bigsqcup_{n\in\mathbb{N}} T(d_n)$.

Notación. Escribiremos $T \uparrow^n (d)$ para referirnos a la aplicación de n veces T a d; de forma recursiva: $T \uparrow^0 (d) = d$ y $T \uparrow^n (d) = T(T \uparrow^{n-1} (d))$ para $n > 0$.

[1] Por la proposición 2.3.1 sabemos que el menor punto prefijo y el menor punto fijo son el mismo punto.

La siguiente proposición caracteriza el menor punto fijo de una transformación continua:

Proposición 2.3.3 (Teorema del Punto Fijo) (Knaster y Tarski [60]). *Si $(D, \sqsubseteq)$ es un ω-CPO con fondo y T es una transformación monótona y continua, entonces $\{T \uparrow^n (\triangle) \mid n \in \mathbb{N}\} \subseteq D$ es una ω-cadena, y se cumple $\bigsqcup_{n\in\mathbb{N}} T \uparrow^n (\triangle) = \mu(T)$.*

Demostración. Veamos primero que $\{T \uparrow^n (\triangle) \mid n \in \mathbb{N}\} \subseteq D$ es una cadena por inducción sobre n:

- **Base ($n = 0$)**

  Se tiene que $T \uparrow^0 (\triangle) \sqsubseteq T \uparrow^1 (\triangle)$ por ser $T \uparrow^0 (\triangle) = \triangle$, el mínimo de D según el orden parcial $\sqsubseteq$.

- **Paso inductivo ($n > 0$)**

  Por H.I. sabemos que $T \uparrow^{n-1} (\triangle) \sqsubseteq T \uparrow^n (\triangle)$ y $T(T \uparrow^{n-1} (\triangle)) \sqsubseteq T(T \uparrow^n (\triangle))$ por ser T monótono. Por lo anterior se tiene que $T \uparrow^n (\triangle) \sqsubseteq T \uparrow^{n+1} (\triangle)$.

Como $\{T \uparrow^n (\triangle) \mid n \in \mathbb{N}\} \subseteq D$ es una cadena, sabemos que existe su supremo, que llamaremos d_0.

$$d_0 =_{def} \bigsqcup_{n\in\mathbb{N}} T \uparrow^n (\triangle)$$

Ahora necesitamos comprobar que d_0 es el menor punto [pre]fijo ($d_0 = \mu(T)$):

Por ser T una transformación continua, $T(d_0) = \bigsqcup_{n\in\mathbb{N}} T \uparrow^{n+1} (\triangle)$. Como sabemos también que el supremo de una cadena no varía por quitar un elemento menor o igual que todos los demás, podemos quitar $T \uparrow^0 (\triangle) = \triangle$, y entonces $\bigsqcup_{n\in\mathbb{N}} T \uparrow^{n+1} (\triangle) = \bigsqcup_{n\in\mathbb{N}\setminus\{0\}} T \uparrow^{n+1} (\triangle) = \bigsqcup_{n\in\mathbb{N}} T \uparrow^n (\triangle) = d_0$.

Por lo que acabamos de razonar, $T(d_0) = d_0$, es decir, $d_0 \in PF(T)$. Para que se tenga $d_0 = \mu(T)$ es suficiente demostrar que para cualquier $d \in PPF(T)$ se cumple $d_0 \sqsubseteq d$. Sea $d \in PPF(T)$. Por inducción sobre n, demostramos que $T \uparrow^n (\triangle) \sqsubseteq d$ para todo $n \in \mathbb{N}$. En efecto:

- **Base ($n = 0$)**

  Trivial por ser $T \uparrow^0 (\triangle) = \triangle \sqsubseteq d$.

- **Paso inductivo ($n > 0$)**

  Supongamos (H.I.) que $T \uparrow^n (\triangle) \sqsubseteq d$. Por ser T monótona, se deduce que $T \uparrow^{n+1} (\triangle) \sqsubseteq T(d)$. Por ser $d \in PPF(T)$, se tiene $T(d) \sqsubseteq d$. Luego $T \uparrow^{n+1} (\triangle) \sqsubseteq d$.

Al cumplirse $T \uparrow^n (\triangle) \sqsubseteq d$ para todo $n \in \mathbb{N}$, se deduce por definición de supremo que $\bigsqcup_{n\in\mathbb{N}} T \uparrow^n (\triangle) \sqsubseteq d$. Es decir, $d_0 \sqsubseteq d$, como queríamos demostrar. □

Capítulo 3

Fundamentos de la Programación Lógica Clásica

Un requisito indispensable a la hora de abordar cualquier extensión a un lenguaje de programación es conocer, con cierta exactitud y profundidad, el lenguaje que estamos extendiendo. Es por esto que este capítulo intenta dar una visión general de los fundamentos de la programación lógica clásica, de manera que en la extensión con incertidumbre considerada en el capítulo 5 podamos seguir el mismo esquema de desarrollo que en este. Tendremos así un caso *cualitativo* (este capítulo) y *cuantitativo* (capítulo 5) de cada uno de los resultados propuestos.

Debido a este motivo, y para facilitar la introducción de la incertidumbre, algunos de los resultados aquí propuestos diferirán de la manera tradicional de presentarlos, no suponiendo, a nuestro parecer, ninguna dificultad añadida a la hora de comprender cuáles son las características fundamentales del lenguaje de programación lógica aquí presentado.

Se pueden consultar como referencia auxiliar otros textos generales sobre la Programación Lógica como *Apt* [1] o *Lloyd* [40]. También puede resultar de ayuda acudir al artículo de *van Emden* [61], que a pesar de presentar ambas versiones (cualitativa y cuantitativa) de forma simultánea, supone la base que este trabajo pretende desarrollar y ampliar.

3.1. Sintaxis

Partiendo de la Lógica de Primer Orden introducida en el capítulo 2, llamaremos *programa lógico clásico* o simplemente *programa* $\mathcal{P}$ a un conjunto finito y no vacío de cláusulas definidas o *de Horn* sin igualdad.

Notación. Emplearemos un sintaxis similar a la de PROLOG para escribir nuestros programas con la salvedad de emplear `<-` en lugar de `:-` como se emplea en PROLOG para separar la cabeza del cuerpo de una cláusula. Al igual que en PROLOG, serán variables todos aquellos identificadores que comiencen por mayúscula o por `_`, entendiéndose que todos los demás son, o bien símbolos de proposición, o bien símbolos de función (que denominaremos *constructoras* de aquí en adelante).

Empleando estas cláusulas podemos construir pequeños programas como los mostrados a continuación, que nos servirán de ejemplo a lo largo del capítulo.

Ejemplo 3.1.1. Suponiendo que tenemos los números naturales representados mediante el cero (`c`) y el sucenor (`s`) de un numero natural, el siguiente programa permite comprobar si un número natural es par o impar, y calcular la suma de dos naturales cualesquiera.

```
even(c) <-
even(s(X)) <- odd(X)
odd(s(c)) <-
odd(s(X)) <- even(X)

plus(X,c,X) <-
plus(X,s(Y),s(Z)) <- plus(X,Y,Z)
```

En un programa podremos emplear también listas mediante las mismas constructoras que en PROLOG como puede verse en el siguiente ejemplo.

Ejemplo 3.1.2. El predicado `lo` del siguiente ejemplo comprueba si una lista de números está ordenada. El predicado auxiliar `mig` comprueba si su primer argumento es menor o igual que el segundo.

```
lo([]) <-
lo([N]) <-
lo([N1,N2|R]) <- mig(N1,N2), lo([N2|R])

mig(c,N) <-
mig(s(N1),s(N2)) <- mig(N1,N2)
```

También se pueden utilizar variables en el cuerpo de una cláusula que no aparezcan en la cabeza (variables extra) tal y como muestra el siguiente ejemplo.

Ejemplo 3.1.3. Escribimos un predicado `tieneHijas` que nos dice si una determinada persona tiene al menos una hija.

```
mujer(maria) <-
hombre(juan) <-
hombre(miguel) <-
padre(juan,maria) <-
madre(maria,miguel) <-

tieneHijas(P) <- padre(P,H), mujer(H)
tieneHijas(P) <- madre(P,H), mujer(H)
```

En lo sucesivo emplearemos estos tres ejemplos, bien tal y como están, o bien con alguna ampliación que se indicará en su momento, para ilustrar los diferentes resultados que se consideren de interés.

Llamaremos *base de Herbrand abierta*, que escribiremos At_Σ, al conjunto de todos los átomos que se puedan construir con la signatura Σ; e *interpretación [de Herbrand abierta]*, que representaremos por $\mathcal{I}$, a todo subconjunto de átomos de At_Σ que cumpla que es cerrado bajo la aplicación de sustituciones. Es decir, cualquier interpretación $\mathcal{I}$ cumplirá:

1. $\mathcal{I} \subseteq At_\Sigma$.
2. Dado un átomo $A \in \mathcal{I}$, se tiene que $A\sigma \in \mathcal{I}$ para cualquier sustitución $\sigma \in Sust_\Sigma$.

Al conjunto de todas las posibles interpretaciones de Herbrand abiertas de un programa lo llamaremos Int_Σ.

A pesar de que toda interpretación nos permite conocer qué átomos de un programa se consideran válidos, no todas las interpretaciones harán una interpretación *coherente* con lo que el programa pretende decir. Según la semántica de la LPO, cuando el cuerpo de una cláusula se verifique para una determinada sustitución de sus variables (en interpretaciones de Herbrand, la instancia del cuerpo de la cláusula es subconjunto de la interpretación), debe verificarse también la instancia que dicha sustitución hace de la cabeza de la cláusula. A las interpretaciones que cumplen esto, se les denomina *modelos [de Herbrand]* del programa. Formalmente tenemos que dada una interpretación $\mathcal{I} \in Int_\Sigma$, se dice que es *modelo* de $\mathcal{P}$, y escribimos $\mathcal{I} \models \mathcal{P}$, si y sólo si para toda cláusula $(A \leftarrow \overline{B}) \in \mathcal{P}$ y para toda sustitución $\sigma \in Sust_\Sigma$ tales que $\overline{B}\sigma \subseteq \mathcal{I}$ se cumple que $A\sigma \in \mathcal{I}$. Llamaremos $HMod(\mathcal{P})$ a la colección de modelos de Herbrand de $\mathcal{P}$.

Interpretación en LPO asociada a una interpretación de Herbrand

Como se ha podido observar, las interpretaciones de Herbrand son conjuntos de átomos, incluyendo en cada caso todos los átomos que se consideran válidos bajo la interpretación en cuestión. Esto parece no tener relación con las interpretaciones en LPO tal y como se definían en el capítulo 2, sin embargo, es posible construir una interpretación en LPO a partir de cualquier interpretación de Herbrand empleando un procedimiento estándar. La idea consiste en construir una Σ-estructura $\mathcal{A}_\mathcal{I}$ y un estado ε que constituyan una Σ-interpretación en LPO, de forma que, dada una interpretación de Herbrand $\mathcal{I} \in Int_\Sigma$, la Σ-interpretación $\mathfrak{I}_\mathcal{I} = \langle \mathcal{A}_\mathcal{I}, \varepsilon \rangle$ verifique los mismos átomos que pertenecen a la interpretación $\mathcal{I}$. Para conseguir esto, construimos $\mathfrak{I}_\mathcal{I} = \langle \mathcal{A}_\mathcal{I}, \varepsilon \rangle$ como sigue:

- $\mathcal{A}_\mathcal{I} = \langle A_\mathcal{I}, \{f^{\mathcal{A}_\mathcal{I}}\}_{f \in FS_\Sigma}, \{p^{\mathcal{A}_\mathcal{I}}\}_{p \in PS_\Sigma} \rangle$

  donde $A_\mathcal{I}$ lo definimos como el conjunto $Term_\Sigma$ de los términos, y la interpretación de cada símbolo de función es:

  - $c^{\mathcal{A}_\mathcal{I}} =_{def} c$ si $c \in FS^0_\Sigma$
  - $f^{\mathcal{A}_\mathcal{I}}(t_1, \ldots, t_n) =_{def} f(t_1, \ldots, t_n)$ si $f \in FS^n_\Sigma$, con $n > 0$, y $t_1, \ldots, t_n \in Term_\Sigma$.

  y la de los símbolos de predicado es:

- $p^{\mathcal{A}_\mathcal{I}} =_{def} \begin{cases} 1 & \text{si } p \in \mathcal{I} \\ 0 & \text{si } p \notin \mathcal{I} \end{cases}$ si $p \in PS^0_\Sigma$

- $p^{\mathcal{A}_\mathcal{I}} =_{def} \begin{cases} 1 & \text{si } p(t_1, \ldots, t_n) \in \mathcal{I} \\ 0 & \text{si } p(t_1, \ldots, t_n) \notin \mathcal{I} \end{cases}$ si $p \in PS^n_\Sigma$, con $n > 0$, y $t_1, \ldots, t_n \in Term_\Sigma$.

- $\varepsilon : \mathcal{V} \to A_\mathcal{I}$ con $\varepsilon(x) =_{def} x$, el estado que asocia a cada variable la misma variable.

De esta forma, dado un término $t \in Term_\Sigma$, su interpretación es el propio término, es decir, $[\![t]\!]^{\mathfrak{I}_\mathcal{I}} = t$; y la interpretación de una fórmula atómica A, es $[\![A]\!]^{\mathfrak{I}_\mathcal{I}} = 1$ si $A \in \mathcal{I}$ y $[\![A]\!]^{\mathfrak{I}_\mathcal{I}} = 0$ si $A \notin \mathcal{I}$.

Inferencia a partir de un programa lógico

Una vez conocido cómo escribir un programa en nuestro lenguaje de programación lógica, necesitamos saber cómo emplear nuestro programa de manera que podamos obtener algún resultado de él. Existen dos formas de hacerlo, bien para inferencia, bien para resolución. En la inferencia, se comprueba la validez lógica de una fórmula (cuantificada universalmente) a partir de las cláusulas del programa; y en la resolución, buscamos la prueba de que una conjunción existencial de átomos es deducible a partir programa. Como a la resolución le dedicaremos toda una sección posterior, no nos extenderemos más en ella.

Cuando comprobamos la validez lógica de una fórmula en un programa, lo que hacemos es intentar inferir la validez de la fórmula empleando las cláusulas del programa sabiendo que podremos inferir la cabeza de una cláusula si hemos podido hacerlo con su cuerpo. Esta idea de inferencia que pasa del cuerpo de una cláusula a su cabeza se puede formalizar de dos maneras alternativas: mediante un operador que permita calcular el conjunto de átomos deducibles a partir de un programa y mediante una lógica particular conocida como *Lógica de Horn*. Veamos una descripción más detallada de ambas maneras.

El siguiente operador nos permite obtener todas aquellas fórmulas que pueden inferirse de las cláusulas del programa. Definimos el operador, que llamaremos $T_\mathcal{P} : Int_\Sigma \to Int_\Sigma$, de la siguiente forma:

$$T_\mathcal{P}(\mathcal{I}) =_{def} \{A\sigma \mid \sigma \in Sust_\Sigma, (A \leftarrow \overline{B}) \in \mathcal{P}, \overline{B}\sigma \subseteq \mathcal{I}\}$$

Informalmente tenemos que dada una interpretación $\mathcal{I} \in Int_\Sigma$, $T_\mathcal{P}(\mathcal{I})$ será el conjunto de átomos deducibles a partir de las instancias de cláusulas de $\mathcal{P}$.

La *Lógica de Horn* nos aporta este mecanismo de inferencia mediante el cual podemos verificar si un átomo dado es consecuencia lógica de las cláusulas que conforman nuestro programa. Este cálculo lógico, que llamaremos HL (por *Horn Logic*), presenta una única regla de inferencia denominada *modus ponens* que viene a decir que podemos demostrar un átomo A siempre que sea igual a la cabeza H, afectada de una sustitución σ, de una de las cláusulas del programa y podamos demostrar el cuerpo de dicha cláusula $\overline{B}$ afectado de la misma sustitución σ (que deberá instanciar adecuadamente todas sus variables). El proceso de cálculo sería así recursivo hasta alcanzar átomos que se corresponden con instancias de hechos del programa. Formalmente tenemos:

$$\frac{B_1\sigma \quad \cdots \quad B_n\sigma}{H\sigma}\ MP \quad \text{si } \sigma \in Sust_\Sigma \text{ y } (H \leftarrow B_1, \ldots, B_n) \in \mathcal{P}$$

Los ejemplos que siguen muestran como emplear la regla *Modus Ponens* de inferencia del cálculo lógico.

Ejemplo 3.1.4. A partir del programa del ejemplo 3.1.2, intentaremos verificar que el átomo

```
lo([s(c), s(s(c)) | []])
```

es deducible de las cláusulas del programa. Es fácil comprobar que el átomo es igual a la cabeza de la tercera regla de `lo` cuando sustituimos sus variables mediante la sustitución $\sigma_1 = \{N1 \mapsto s(c), N2 \mapsto s(s(c)), R \mapsto []\}$, por lo tanto podemos realizar el siguiente proceso de inferencia:

$$\dfrac{\dfrac{\dfrac{}{\texttt{mig(c, s(c))}}\,(\text{mig.1})_{\sigma_4}}{\texttt{mig(s(c), s(s(c)))}}\,(\text{mig.2})_{\sigma_2} \quad \dfrac{}{\texttt{lo([s(s(c)) | []])}}\,(\text{lo.2})_{\sigma_3}}{\texttt{lo([s(c), s(s(c)) | []])}}\,(\text{lo.3})_{\sigma_1}$$

donde las sustituciones empleadas son las que siguen:

$$\sigma_1 = \{N1 \mapsto s(c), N2 \mapsto s(s(c)), R \mapsto []\}$$
$$\sigma_2 = \{N1 \mapsto c, N2 \mapsto s(c)\}$$
$$\sigma_3 = \{N \mapsto s(s(c))\}$$
$$\sigma_4 = \{N \mapsto s(c)\}$$

□

Como ha demostrado el ejemplo anterior, el átomo `lo([s(c), s(s(c)) | []])` es deducible a partir del programa del ejemplo 3.1.2. Por lo tanto, podemos escribir $\mathcal{P} \vdash_{HL} A$ donde $\mathcal{P}$ es el programa del ejemplo 3.1.2 y $A =$ `lo([s(c), s(s(c)) | []])`. Además, podemos hacer explícito el número de inferencias HL realizadas escribiendo $\mathcal{P} \vdash^4_{HL} A$.

Notación. Extenderemos además la notación anterior para poder emplearla con una conjunción de átomos en vez de con un único átomo, por lo tanto, dados un programa $\mathcal{P}$ y una conjunción de átomos $\overline{A}$, podemos escribir $\mathcal{P} \vdash_{HL} \overline{A}$ o $\mathcal{P} \vdash_{HL} A_1, \ldots, A_n$ para indicar que existe demostración en HL para todos los átomos de la conjunción (esta demostración, a diferencia de la del ejemplo anterior, será un bosque dado que contendrá un árbol para cada átomo de la conjunción). Podremos también indicar el número de inferencias HL realizadas para demostrar todos los átomos de la conjunción de igual forma que hicimos para un único átomo, cumpliéndose, ahora, que el número total de inferencias HL realizadas para demostrar la conjunción de átomos es la suma del número de inferencias realizadas para demostrar cada uno de los átomos por separado.

Veamos ahora otro ejemplo de lo que ocurre cuando aparecen variables extra en el cuerpo de una cláusula.

Ejemplo 3.1.5. Intentamos verificar que el átomo `tieneHijas(juan)` se verifica para el programa del ejemplo 3.1.3. El siguiente árbol de inferencia muestra los pasos necesarios para comprobar que `tieneHijas(juan)` se verifica:

$$\dfrac{\dfrac{}{\texttt{padre(juan,maria)}}\,(\text{padre.1}) \quad \dfrac{}{\texttt{mujer(maria)}}\,(\text{mujer.1})}{\texttt{tieneHijas(juan)}}\,(\text{tieneHijas.1})_{\sigma}$$

donde $\sigma = \{P \mapsto juan, H \mapsto maria\}$. □

Podemos comprobar, en el ejemplo anterior, que la sustitución σ ha instanciado correctamente las variables de la cláusula del programa al hacer $H \mapsto maria$ dado que sólo así se consigue una instancia de la cláusula tieneHijas.1 cuyo cuerpo pueda ser verificado.

3.2. Semántica declarativa

La semántica de un lenguaje permite dar un significado formal a los programas y la declarativa en particular nos aporta la teoría lógica necesaria para poder comprobar o demostrar diferentes propiedades sobre nuestro programa basándonos en una lógica adecuada, que en el caso de la programación lógica clásica es la lógica de primer orden.

Nuestro objetivo es conseguir un modelo formal de nuestro programa que nos permita trabajar sobre él de manera que podamos estudiarlo y conocer con exactitud qué resultados podemos esperar. Ya en la sección anterior se introdujo el concepto de interpretación de un programa; ahora nos interesarán aquellas interpretaciones que además sean modelos del programa, es decir, cumplan que para toda cláusula $A \leftarrow \overline{B}$ de un programa $\mathcal{P}$ dado, y para toda sustitución σ, si $\overline{B}\sigma$ está contenido en la interpretación, entonces lo esté también $A\sigma$. De estos modelos, nos fijaremos en aquellos átomos que sean válidos en todos ellos, estos átomos serán las consecuencias lógicas del programa. Y es demostrable que coinciden con los átomos deducibles en HL y con los átomos pertenecientes a un modelo singular del programa, el modelo mínimo. A lo largo del resto de la sección, desarrollaremos esta idea.

En [1, 40] tenemos dos referencias clásicas que emplean las interpretaciones de Herbrand cerradas. Además, en [2] se pueden consultar otros enfoques basados en interpretaciones de Herbrand abiertas, como es el caso que sigue en este trabajo (y que se corresponde con el de la $\mathcal{C}$-semántica, en la terminología explicada en [2]). Según se explica en [2], la $\mathcal{C}$-semántica fue introducida en [13] y estudiada con mayor profundidad en [22].

Partiremos de un programa $\mathcal{P}$ cualquiera para la signatura Σ definida según se introdujo en el capítulo 2. Dado que las interpretaciones de $\mathcal{P}$ son conjuntos de átomos, podemos ver que Int_Σ es un retículo completo según el orden parcial $\subseteq$ como demuestra la siguiente proposición.

Proposición 3.2.1. *Dado un programa $\mathcal{P}$ y su conjunto Int_Σ de las posibles interpretaciones de Herbrand, $(Int_\Sigma, \subseteq)$ conforma un retículo completo (el retículo de la interpretaciones de Herbrand). Además, para cualquier subconjunto $I \in Int_\Sigma$ de interpretaciones de $\mathcal{P}$, se cumple que el supremo de I (que escribiremos $\bigsqcup I$) coincide con la unión de las interpretaciones contenidas en I y, el ínfimo de I (que escribiremos $\bigsqcap I$) coincide con la intersección de las interpretaciones contenidas en I.*

Demostración. Para que $(Int_\Sigma, \subseteq)$ sea un retículo completo, deberá tenerse que existen el máximo y el mínimo de Int_Σ, y que para todo subconjunto I de Int_Σ, existen su supremo y su ínfimo.

Veamos primero que podemos encontrar el mínimo y el máximo del conjunto de interpretaciones de $\mathcal{P}$:

1. En el caso del mínimo, debemos encontrar una interpretación $\mathcal{I} \in Int_\Sigma$ que cumpla que para cualquier interpretación $\mathcal{J} \in Int_\Sigma$, $\mathcal{I} \subseteq \mathcal{J}$. Supongamos $\mathcal{I} = \emptyset$. Es fácil comprobar

que se trata de una interpretación pues $\emptyset \subseteq At_\Sigma$ y es cerrado bajo la aplicación de sustituciones. Además se cumple que para toda interpretación $\mathcal{J} \in Int_\Sigma$, $\emptyset \subseteq \mathcal{J}$. Por lo tanto $min(Int_\Sigma) = \triangle = \emptyset$.

2. En el caso del máximo, debemos encontrar una interpretación $\mathcal{I} \in Int_\Sigma$ que cumpla que para cualquier interpretación $\mathcal{J} \in Int_\Sigma$, $\mathcal{J} \subseteq \mathcal{I}$. Supongamos $\mathcal{I} = At_\Sigma$. At_Σ es una interpretación pues se tiene que $At_\Sigma \subseteq At_\Sigma$ y es cerrado bajo la aplicación de sustituciones pues por defínción contiene a todos los átomos. Además, para cualquier interpretación $\mathcal{I} \in Int_\Sigma$, $\mathcal{I} \subseteq At_\Sigma$ por la misma defínción de interpretación. Así, $max(Int_\Sigma) = \triangledown = At_\Sigma$.

Ahora debemos poder encontrar el supremo y el ínfimo de todo subconjunto de Int_Σ. Sea I un subconjunto de Int_Σ. Se tiene que :

(1) $\bigsqcup I = \bigcup\{\mathcal{I} \in Int_\Sigma \mid \mathcal{I} \in I\}$ (que escribiremos $\bigcup I$).

a) La unión es una interpretación ($\bigcup I \in Int_\Sigma$). Para todo átomo $A \in \bigcup I$, existe una interpretación $\mathcal{I} \in I$ tal que $A \in \mathcal{I}$. Y por ser $\mathcal{I}$ una interpretación, se tiene que $B \in \mathcal{I}$ para todo átomo B que fuera de la forma $A\sigma$ para alguna sustitución $\sigma \in Sust_\Sigma$; por lo que se tiene también que $B \in \bigcup I$.

b) La unión es cota superior de I ($\mathcal{I} \subseteq \bigcup I$ para toda interpretación $\mathcal{I} \in I$). Trivial por la definición de la unión de una familia de conjuntos.

c) La unión es la menor de las cotas superiores (si $\mathcal{J} \in Int_\Sigma$ es cota superior de I, entonces $\bigcup I \subseteq \mathcal{J}$). Si $\mathcal{J}$ es cota superior de $\mathcal{I}$, entonces $\mathcal{I} \subseteq \mathcal{J}$ para toda $\mathcal{I} \in I$. Por definición de unión de una familia de conjuntos, se deduce que $\bigcup I \subseteq \mathcal{J}$.

Por lo tanto se tiene que existe el supremo de I y coincide con $\bigcup I$.

(2) $\bigsqcap I = \bigcap\{\mathcal{I} \in Int_\Sigma \mid \mathcal{I} \in I\}$ (que escribiremos $\bigcap I$).

a) La intersección es una interpretación ($\bigcap I \in Int_\Sigma$). Para todo átomo $A \in \bigcap I$, $A \in \mathcal{I}$ para toda interpretación $\mathcal{I} \in I$. Y por ser $\mathcal{I}$ una interpretación, se tiene que $B \in \mathcal{I}$ para todo átomo B que fuera de la forma $A\sigma$ para alguna sustitución $\sigma \in Sust_\Sigma$; por lo que se tiene también que $B \in \bigcap I$.

b) La intersección es cota inferior de I ($\bigcap I \subseteq \mathcal{I}$ para toda interpretación $\mathcal{I} \in I$). Trivial por la definición de la intersección de una familia de conjuntos.

c) La intersección es la mayor de las cotas inferiores (si $\mathcal{J} \in Int_\Sigma$ es cota inferior de I, entonces $\mathcal{J} \subseteq \bigcap I$). Si $\mathcal{J}$ es cota inferior de I, entonces $\mathcal{J} \subseteq \mathcal{I}$ para toda $\mathcal{I} \in I$. Por definición de intersección de una familia de conjuntos, se deduce que $\mathcal{J} \subseteq \bigcap I$.

Por lo tanto se tiene que existe el ínfimo de I y coincide con $\bigcap I$.

Por todo lo anterior, $(Int_\Sigma, \subseteq)$ es un retículo completo. □

Dado que ahora sabemos que $(Int_\Sigma, \subseteq)$ es un retículo completo, podemos utilizar algunos resultados sobre conjuntos ordenados que veíamos en la sección 2.1 con el objetivo de obtener el modelo que nos interesa de un programa: el *modelo mínimo*. Sin embargo, necesitamos además tener una transformación monótona y continua para poder hacerlo.

En la sección dedicada a la sintaxis de nuestros programas lógicos definimos un operador, que llamamos $T_{\mathcal{P}}$, que nos permitía calcular, dada una interpretación de un programa, los átomos que resultaban consecuencia de los que podíamos verificar según la interpretación y las cláusulas del programa. Formalmente la definición del operador era $Tp(\mathcal{I}) =_{def} \{A\sigma \mid \sigma \in Sust_{\Sigma}, (A \leftarrow \overline{B}) \in \mathcal{P}, \overline{B}\sigma \subseteq \mathcal{I}\}$. Teniendo en cuenta la definición anterior podemos comprobar que $T_{\mathcal{P}}$ es un operador monótono y continuo:

Proposición 3.2.2. *El operador $T_{\mathcal{P}} : Int_{\Sigma} \to Int_{\Sigma}$, según se ha definido anteriormente, es monótono y continuo.*

Demostración. Demostramos monotonía y continuidad por separado.

- **$T_{\mathcal{P}}$ es monótono**
  Dadas dos interpretaciones $\mathcal{I}$ y $\mathcal{J}$ tales que $\mathcal{I} \subseteq \mathcal{J}$ se tiene según la definición del operador $T_{\mathcal{P}}$ que:

  (1) $Tp(\mathcal{I}) = \{A\sigma \mid \sigma \in Sust_{\Sigma}, (A \leftarrow \overline{B}) \in \mathcal{P}, \overline{B}\sigma \subseteq \mathcal{I}\}$
  (2) $Tp(\mathcal{J}) = \{A\sigma \mid \sigma \in Sust_{\Sigma}, (A \leftarrow \overline{B}) \in \mathcal{P}, \overline{B}\sigma \subseteq \mathcal{J}\}$

  pero $\overline{B}\sigma \subseteq \mathcal{I} \Rightarrow \overline{B}\sigma \subseteq \mathcal{J}$ por lo que $T_{\mathcal{P}}(\mathcal{I}) \subseteq T_{\mathcal{P}}(\mathcal{J})$.

- **$T_{\mathcal{P}}$ es continuo**
  Sea $\{\mathcal{I}_n\}_{n\in\mathbb{N}}$ una cadena de interpretaciones tal que $\mathcal{I}_n \subseteq \mathcal{I}_{n+1}$ para todo $n \in \mathbb{N}$. Debemos ver que se tiene $\bigcup_{n\in\mathbb{N}} T_{\mathcal{P}}(\mathcal{I}_n) = T_{\mathcal{P}}(\bigcup_{n\in\mathbb{N}} \mathcal{I}_n)$. Separamos la igualdad en dos inclusiones:

  (3) $\bigcup_{n\in\mathbb{N}} T_{\mathcal{P}}(\mathcal{I}_n) \subseteq T_{\mathcal{P}}(\bigcup_{n\in\mathbb{N}} \mathcal{I}_n)$. Para cada $n_0 \in \mathbb{N}$ se tiene por definición de la unión entre conjuntos que $\mathcal{I}_{n_0} \subseteq \bigcup_{n\in\mathbb{N}} \mathcal{I}_n$. Como $T_{\mathcal{P}}$ es monótono, $T_{\mathcal{P}}(\mathcal{I}_{n_0}) \subseteq T_{\mathcal{P}}(\bigcup_{n\in\mathbb{N}} \mathcal{I}_n)$, y como lo anterior se cumple para todo $n_0 \in \mathbb{N}$, se tiene también que $\bigcup_{n\in\mathbb{N}} T_{\mathcal{P}}(\mathcal{I}_n) \subseteq T_{\mathcal{P}}(\bigcup_{n\in\mathbb{N}} \mathcal{I}_n)$.
  (4) $T_{\mathcal{P}}(\bigcup_{n\in\mathbb{N}} \mathcal{I}_n) \subseteq \bigcup_{n\in\mathbb{N}} T_{\mathcal{P}}(\mathcal{I}_n)$. $T_{\mathcal{P}}(\bigcup_{n\in\mathbb{N}} \mathcal{I}_n)$ es, por definición de $T_{\mathcal{P}}$, el conjunto $\{A\sigma \mid \sigma \in Sust_{\Sigma}, (A \leftarrow \overline{B}) \in \mathcal{P}, \overline{B}\sigma \subseteq \bigcup_{n\in\mathbb{N}} \mathcal{I}_n\}$. Puesto que $\{\mathcal{I}_n\}_{n\in\mathbb{N}}$ es una cadena, para cada $A\sigma \in T_{\mathcal{P}}(\bigcup_{n\in\mathbb{N}} \mathcal{I}_n)$ se puede encontrar una cláusula $(A \leftarrow \overline{B}) \in \mathcal{P}$ y un $n_0 \in \mathbb{N}$ suficientemente grande, tal que $\overline{B}\sigma \subseteq \mathcal{I}_{n_0}$. Por lo tanto, $A\sigma \in T_{\mathcal{P}}(\mathcal{I}_{n_0})$ y con ello $A\sigma \in \bigcup_{n\in\mathbb{N}} T_{\mathcal{P}}(\mathcal{I}_n)$, con lo cual queda demostrada la inclusión.

  Y ahora por (3) y (4) se tiene que $\bigcup_{n\in\mathbb{N}} T_{\mathcal{P}}(\mathcal{I}_n) = T_{\mathcal{P}}(\bigcup_{n\in\mathbb{N}} \mathcal{I}_n)$. □

Una propiedad que podemos comprobar para nuestro operador $T_{\mathcal{P}}$, es que los modelos de Herbrand de $\mathcal{P}$ coinciden con los puntos prefijos de $T_{\mathcal{P}}$:

Proposición 3.2.3. *Dado un programa $\mathcal{P}$, el conjunto de puntos prefijos de $T_{\mathcal{P}}$ es igual al conjunto de modelos de Herbrand de $\mathcal{P}$. Es decir, $PPF(T_{\mathcal{P}}) = HMod(\mathcal{P})$.*

Demostración. Demostramos que, para toda interpretación $\mathcal{I} \in Int_{\Sigma}$, se cumple $T_{\mathcal{P}}(\mathcal{I}) \subseteq \mathcal{I} \Leftrightarrow \mathcal{I} \models \mathcal{P}$.

($\Rightarrow$) Trivial puesto que $T_{\mathcal{P}}(\mathcal{I}) \subseteq \mathcal{I}$ implica que para toda cláusula $A \leftarrow \overline{B}$ de $\mathcal{P}$ y para toda sustitución σ tal que $\overline{B}\sigma \in \mathcal{I}$, $A\sigma \in \mathcal{I}$.

($\Leftarrow$) Sea $\mathcal{I}$ un modelo de $\mathcal{P}$. Dado $A' \in T_{\mathcal{P}}(\mathcal{I})$, se deduce por definición de $T_{\mathcal{P}}$ que existen una cláusula $A \leftarrow \overline{B}$ en $\mathcal{P}$ y una sustitución $\sigma \in Sust_{\Sigma}$ tales que $A' = A\sigma$ y $\overline{B}\sigma \subseteq \mathcal{I}$. Ahora por ser $\mathcal{I}$ un modelo de $\mathcal{P}$ se tendrá que $A' \in \mathcal{I}$.

Por lo tanto se tiene que $PPF(T_{\mathcal{P}}) = HMod(\mathcal{P})$. □

De lo que sabemos hasta ahora podemos concluir dos cosas. Por un lado, que existe un modelo menor que todos los demás: el modelo mínimo; y por otro, que podemos calcular este modelo mínimo mediante el operador $T_{\mathcal{P}}$. Estos dos resultados se enuncian formalmente en los dos siguientes corolarios:

Corolario 3.2.1. *Dado un programa $\mathcal{P}$ y el operador $T_{\mathcal{P}}$ según se ha definido con anterioridad, se tiene que el menor punto prefijo de $T_{\mathcal{P}}$ es igual a $\bigcap\{\mathcal{I} \in Int_{\Sigma} \mid T_{\mathcal{P}}(\mathcal{I}) \subseteq \mathcal{I}\}$ y coincide con el menor modelo de Herbrand de $\mathcal{P}$. Pondremos $\mu(T_{\mathcal{P}}) = \mathcal{M}_{\mathcal{P}}$.*

Demostración. El conjunto $\{\mathcal{I} \in Int_{\Sigma} \mid T_{\mathcal{P}}(\mathcal{I}) \subseteq \mathcal{I}\}$ es el conjunto de los puntos prefijos de $T_{\mathcal{P}}$, y por la proposición 3.2.3, $PPF(T_{\mathcal{P}}) = HMod(\mathcal{P})$. Por la proposición 3.2.1, sabemos que $\bigcap PPF(T_{\mathcal{P}}) = \bigsqcap PPF(T_{\mathcal{P}})$ y por la proposición 2.3.2, $\bigsqcap PPF(T_{\mathcal{P}}) = \mu(T_{\mathcal{P}})$. Como $PPF(T_{\mathcal{P}}) = HMod(\mathcal{P})$, $\mu(T_{\mathcal{P}})$ es también el menor modelo de Herbrand de $\mathcal{P}$ que denotaremos por $\mathcal{M}_{\mathcal{P}}$. □

Corolario 3.2.2. *Dado un programa $\mathcal{P}$ y el operador $T_{\mathcal{P}}$ según se ha definido con anterioridad, el modelo mínimo de $\mathcal{P}$ es igual a $\bigcup_{n\in\mathbb{N}} T_{\mathcal{P}} \uparrow^n (\emptyset)$.*

Demostración. Por las proposiciones 3.2.1 y 3.2.2, $(Int_{\Sigma}, \subseteq)$ es retículo completo (y en particular, ω-CPO con fondo) y $T_{\mathcal{P}}$ es monónoto y continuo. Por lo tanto, por la proposición 2.3.3 se tiene que $\bigcup_{n\in\mathbb{N}} T_{\mathcal{P}} \uparrow^n (\emptyset) = \mu(T_{\mathcal{P}})$. Y por el corolario anterior, $\mu(T_{\mathcal{P}}) = \mathcal{M}_{\mathcal{P}}$. □

El modelo mínimo $\mathcal{M}_{\mathcal{P}}$ se puede caracterizar también como la colección de aquellos átomos que se deducen de $\mathcal{P}$ empleando el cálculo lógico HL. Formalmente:

Proposición 3.2.4. $\mathcal{M}_{\mathcal{P}} = \{A \in At_{\Sigma} \mid \mathcal{P} \vdash_{HL} A\}$.

Demostración. Por el corolario 3.2.2 tenemos que $\mathcal{M}_{\mathcal{P}} = \bigcup_{n\in\mathbb{N}} T_{\mathcal{P}} \uparrow^n (\emptyset)$. Y también podemos poner que $\{A \in At_{\Sigma} \mid \mathcal{P} \vdash_{HL} A\}$ es igual a $\bigcup_{n\in\mathbb{N}}\{A \in At_{\Sigma} \mid \mathcal{P} \vdash^n_{HL} A\}$. Basta entonces demostrar que las dos afirmaciones siguientes son ciertas:

(1) $\mathcal{P} \vdash^n_{HL} A \Rightarrow \exists m(A \in T_{\mathcal{P}} \uparrow^m (\emptyset))$

(2) $A \in T_{\mathcal{P}} \uparrow^n (\emptyset) \Rightarrow \exists m(\mathcal{P} \vdash^m_{HL} A)$

Comprobamos ahora ambas afirmaciones:

1. $\mathcal{P} \vdash^n_{HL} A \Rightarrow \exists m(A \in T_{\mathcal{P}} \uparrow^m (\emptyset))$. Razonamos por inducción sobre $n \geq 1$ (ya que $\mathcal{P} \vdash^0_{HL} A$ es imposible):

- **Caso base** ($n = 1$)

  $\mathcal{P} \vdash^1_{HL} A \Rightarrow$ Existe una demostración en una inferencia HL para A. Es decir, A es de la forma $H\sigma$ para una regla $(H \leftarrow) \in \mathcal{P}$ y $\sigma \in Sust_\Sigma$. Por lo tanto se cumple que $A \in T_\mathcal{P} \uparrow^1 (\emptyset)$ $(=_{def} \{H\sigma \mid \sigma \in Sust_\Sigma, (H \leftarrow) \in \mathcal{P}\})$.

- **Caso inductivo** ($n > 1$)

  $\mathcal{P} \vdash^n_{HL} A \Rightarrow A = H\sigma$ es instancia de la cabeza de una regla $(H \leftarrow B_1, \ldots, B_k) \in \mathcal{P}$ donde cada $B_i\sigma$ se tiene con $m_i < n$ inferencias HL y donde $\sum_{i=1}^k m_i = n - 1$. Entonces, por H.I., y por ser $\{T_\mathcal{P} \uparrow^n (\emptyset) \mid n \in \mathbb{N}\}$ cadena, existe m suficientemente grande tal que para $1 \leq i \leq k$, $B_i\sigma \in T_\mathcal{P} \uparrow^m (\emptyset)$ por lo que $A = H\sigma \in T_\mathcal{P} \uparrow^{m+1} (\emptyset)$.

2. $A \in T_\mathcal{P} \uparrow^n (\emptyset) \Rightarrow \exists m(\mathcal{P} \vdash^m_{HL} A)$. Razonamos por inducción sobre $n \geq 1$ (ya que $T_\mathcal{P} \uparrow^0 (\emptyset) = \emptyset$):

   - **Caso base** ($n = 1$)

     $A \in T_\mathcal{P} \uparrow^1 (\emptyset) \Rightarrow A$ es instancia de una regla $(H \leftarrow) \in \mathcal{P}$, por lo que se tiene $\mathcal{P} \vdash^1_{HL} A$.

   - **Caso inductivo** ($n = k + 1, k \geq 1$)

     $A \in T_\mathcal{P} \uparrow^{k+1} (\emptyset) = \{H\sigma \mid \sigma \in Sust_\Sigma, (H \leftarrow C) \in \mathcal{P}, C\sigma \subseteq T_\mathcal{P} \uparrow^k (\emptyset)\}$. Como $C\sigma \subseteq T_\mathcal{P} \uparrow^k (\emptyset)$ se tiene que, por H.I., existe para cada $B_i \in C\sigma$, un m_i tal que $\mathcal{P} \vdash^{m_i}_{HL} B_i$, de forma que $\mathcal{P} \vdash^{\sum_{i=1}^k m_i}_{HL} C\sigma$ por lo que $\mathcal{P} \vdash^{(\sum_{i=1}^k m_i)+1}_{HL} H\sigma = A$.

Así tenemos que $\bigcup_{n\in\mathbb{N}} T_\mathcal{P} \uparrow^n (\emptyset) =_{(1),(2)} \bigcup_{n\in\mathbb{N}} \{A \in At_\Sigma \mid \mathcal{P} \vdash^n_{HL} A\} = \mathcal{M}_\mathcal{P}$ □

Para finalizar esta sección, vamos a ver que $\mathcal{M}_\mathcal{P}$ también coincide con el conjunto de los átomos que se deducen de $\mathcal{P}$ en el sentido de la consecuencia lógica de la LPO. Esto resulta del lema y proposición que siguen.

Lema 3.2.1 (Lema de corrección de HL)**.**

$$\mathcal{P} \vdash_{HL} A \Rightarrow \mathcal{P} \models A^\forall$$

Demostración. Si $\mathcal{P} \vdash_{HL} A$, existe una demostración en HL con exactamente n inferencias, i.e. se tiene $\mathcal{P} \vdash^n_{HL} A$. Demostramos $\mathcal{P} \models A^\forall$ razonando por inducción sobre n.

- **Caso base** ($n = 1$)

  De $\mathcal{P} \vdash^1_{HL} A$ se deduce que A es instancia de un hecho o, lo que es lo mismo, A es $H\sigma$ siendo $H \leftarrow$ una cláusula de $\mathcal{P}$. Entonces, se tiene que $\mathcal{P} \models A^\forall$ porque $(H \leftarrow) \in \mathcal{P}$ equivale a $H^\forall$, y $H^\forall \models (H\sigma)^\forall \equiv A^\forall$.

- **Caso inductivo** ($n > 1$)

  Como $\mathcal{P} \vdash^n_{HL} A$, sabemos que existen una cláusula $(H \leftarrow B_1, \ldots, B_k) \in \mathcal{P}$ y una sustitución $\sigma \in Sust_\Sigma$ tales que $H\sigma = A$ y $\mathcal{P} \vdash^{n_i}_{HL} B_i\sigma$ (con $1 \leq i \leq k$) donde $\sum_{i=1}^k n_i = n-1$. Por hipótesis de inducción, $\mathcal{P} \vdash^{n_i}_{HL} B_i\sigma$ implica que se tiene $\mathcal{P} \models (B_i\sigma)^\forall$ para todo $1 \leq i \leq k$. Además $\mathcal{P} \models (B_1\sigma)^\forall \wedge \ldots \wedge (B_k\sigma)^\forall \rightarrow (H\sigma)^\forall$, debido a que $H \leftarrow B_1, \ldots, B_k$ es una cláusula de $\mathcal{P}$. Luego $\mathcal{P} \models (H\sigma)^\forall \equiv A^\forall$. □

Proposición 3.2.5. *Para cualquier $A \in At_\Sigma$, las cuatro afirmaciones siguientes son equivalentes:*

(1) $\mathcal{P} \models A^\forall$.

(2) $\mathcal{P} \models A$ *(i.e.* $[\![A]\!]^\mathfrak{I} = 1$ *para cualquier interpretación* $\mathfrak{I} = \langle A, \varepsilon \rangle$ *que sea modelo de* $\mathcal{P}$*).*

(3) $A \in \mathcal{I}$ *para cualquier modelo de Herbrand* $\mathcal{I} \in HMod(\mathcal{P})$.

(4) $A \in \mathcal{M}_\mathcal{P}$.

Demostración. Demostramos las cuatro implicaciones necesarias para demostrar la equivalencia:

- (1) ⇒ (2): $A^\forall \models A$ porque en general $\varphi^\forall \models \varphi$ para cualquier fórmula φ de la lógica de primer orden.
- (2) ⇒ (3): Sea $\mathcal{I} \in HMod(\mathcal{P})$, y sea $\mathfrak{I}_\mathcal{I}$ la interpretación en lógica de primer orden representada por $\mathcal{I}$. Por (2), $\mathcal{P} \models A$, y podemos asegurar que $[\![A]\!]^{\mathfrak{I}_\mathcal{I}} = 1$. Por la construcción de $\mathfrak{I}_\mathcal{I}$ a partir de $\mathcal{I}$, $[\![A]\!]^{\mathfrak{I}_\mathcal{I}} = 1$ equivale a $A \in \mathcal{I}$.
- (3) ⇒ (4): Trivial porque $\mathcal{M}_\mathcal{P} \in HMod(\mathcal{P})$.
- (4) ⇒ (1): Por (4) $A \in \mathcal{M}_\mathcal{P}$ y por la proposición 3.2.4, $\mathcal{P} \vdash_{HL} A$. Ahora por el Lema de Corrección de HL (lema 3.2.1) se tiene que $\mathcal{P} \models A^\forall$. □

3.3. Semántica operacional

Si la semántica declarativa nos proporcionaba un modelo lógico para trabajar con nuestro programa, la semántica operacional nos proporcionará un modelo de cómputo a partir del cual podremos resolver objetivos y obtener soluciones adecuadas a dichos objetivos de forma que sean deducibles del programa.

Sea $\mathcal{P}$ un programa para la signatura Σ. Comenzaremos por definir qué es un objetivo de $\mathcal{P}$ y a qué llamaremos solución de dicho objetivo. Sea $\overline{A}$ una conjunción de átomos, y sea $\sigma \in Sust_\Sigma$ una sustitución. Llamaremos *objetivo*, y lo denotaremos por G, a $\overline{A} \;[\![\; \sigma$ siempre que cumpla que σ es idempotente y $\overline{A}\sigma = \overline{A}$, es decir, que $var(\overline{A}) \cap dom(\sigma) = \emptyset$ donde $var(\overline{A})$ es el conjunto de variables que aparecen en cualquiera de los átomos de la conjunción $\overline{A}$. Para un objetivo general como el anterior, $\overline{A}$ serán los átomos pendientes de resolución y σ la sustitución acumulada en pasos de resolución anteriores (respuesta parcial).

Interesan dos tipos de objetivos particulares: los objetivos *iniciales* y los *finales* o *resueltos*. Llamaremos *objetivo inicial* a todo objetivo de la forma $G \equiv \overline{A} \;[\![\; \epsilon$ donde $\epsilon \in Sust_\Sigma$ es la sustitución vacía; y llamaremos *objetivo final* o *resuelto* a todo objetivo de la forma $G \equiv \;[\![\; \sigma$ (también simplemente σ).

Sea ahora $G \equiv \overline{A} \;[\![\; \sigma$ un objetivo cualquiera. Definimos el conjunto de variables de G ($var(G)$) como la unión del conjunto de variables de $\overline{A}$ con los conjuntos de variables del dominio y el rango de σ, es decir, $var(G) =_{def} var(\overline{A}) \cup dom(\sigma) \cup vran(\sigma)$.

Definición 3.3.1. *Se llama* solución *de un objetivo cualquiera* $G \equiv \overline{A} \parallel \sigma$ *a una sustitución* θ *tal que (i)* $\mathcal{P} \vdash_{HL} \overline{A}\theta$ *y (ii)* $\theta = \sigma\theta$*. Pondremos* $\theta \in Sol_{\mathcal{P}}(G)$*, el conjunto de soluciones de* G*.*

Notación. Cuando además se cumple para un objetivo $G \equiv \overline{A} \parallel \sigma$ y una solución $\theta \in Sol_{\mathcal{P}}(G)$ que $\mathcal{P} \vdash^n_{HL} \overline{A}\theta$, se dice que $\theta \in Sol^n_{\mathcal{P}}(G)$. Así, $Sol_{\mathcal{P}}(G) = \bigcup_{n\in\mathbb{N}} Sol^n_{\mathcal{P}}(G)$.

De la definición anterior, se deduce trivialmente que, dado un objetivo resuelto $G \equiv \parallel \sigma$, se tiene que $\sigma \in Sol^0_{\mathcal{P}}(G)$. Diremos, en este caso, que σ es la *solución asociada* a G.

Resolución de objetivos

Llamaremos *resolución* al proceso de cómputo en el cual se intenta obtener una solución para un objetivo dado. Generalmente, este objetivo será un objetivo inicial para el que buscamos una solución. Sin embargo, generalizaremos esta idea para un objetivo cualquiera, pues así podremos también buscar soluciones a objetivos parcialmente resueltos, es decir, para los que ya conocemos parte de su solución (en el caso normal, la propia sustitución presente en el objetivo).

Definimos formalmente esta idea a continuación.

Definición 3.3.2. *Se llama* resolución *de un objetivo cualquiera* G *a una serie de* pasos de resolución *tales que* $G \equiv \overline{A_0} \parallel \sigma_0 \Vdash_{\sigma_1} G_1 \Vdash_{\sigma_2} \cdots \Vdash_{\sigma_n} G_n \equiv \parallel \sigma_0\sigma_1 \ldots \sigma_n$ *donde cada paso* $G_i \Vdash_{\sigma_i+1} G_{i+1}$ *es un* paso de resolución*. Y llamamos* paso de resolución *a un paso de la forma*

$$G \equiv \overline{L}, A, \overline{R} \parallel \sigma \Vdash_{\sigma_1} (\overline{L}, \overline{B}, \overline{R})\sigma_1 \parallel \sigma\sigma_1$$

siendo $(H \leftarrow \overline{B}) \in_{var} \mathcal{P}$ *una variante sin variables en común con* G *de una cláusula de* $\mathcal{P}$ *y* σ_1 *el u.m.g. entre* A *y* H*.*

Notación. En ocasiones utilizaremos la notación $G \Vdash_{\sigma_1, C_1} G'$ para indicar un paso de resolución que emplea la cláusula C_1 y el u.m.g. σ_1.

Más adelante comprobaremos que la solución asociada al objetivo final de una resolución es la solución buscada al objetivo inicial de dicha resolución. De hecho, será solución para todos los objetivos de la resolución.

A continuación pueden verse dos ejemplos de resolución de objetivos. En cada uno resolveremos un objetivo planteado sobre el ejemplo 3.1.2 primero, y sobre el ejemplo 3.1.3 después.

Ejemplo 3.3.1. Sobre el ejemplo 3.1.2 resolvemos el objetivo `lo([X,s(s(c))])` para intentar conseguir una lista ordenada de dos elementos cuyo segundo elemento sea el 2. Una posible resolución es la siguiente:

`lo([X, s(s(c))])` $\parallel \epsilon$	$\Vdash_{\epsilon,\ \text{lo.3}}$
`mig(X, s(s(c))), lo([s(s(c))])` $\parallel \epsilon$	$\Vdash_{\{X\mapsto s(X1)\},\ \text{mig.2}}$
`mig(X1, s(c)), lo([s(s(c))])` $\parallel \{X \mapsto s(X1)\}$	$\Vdash_{\{X1\mapsto c\},\ \text{mig.1}}$
`lo([s(s(c))])` $\parallel \{X \mapsto s(c)\}$	$\Vdash_{\epsilon,\ \text{lo.2}}$
$\parallel \{X \mapsto s(c)\}$	

De la resolución anterior, hemos obtenido la solución $\{X \mapsto s(c)\}$, que sabemos que es solución dado que ya comprobamos (en el ejemplo 3.1.4) que `lo([s(c), s(s(c))])` se verifica para el programa del ejemplo 3.1.2. □

Ejemplo 3.3.2. A partir del ejemplo 3.1.3 intentamos obtener el nombre de una persona que tenga hijas, empleando para ello el objetivo `tieneHijas(X)`. De forma similar que con el ejemplo anterior, buscamos en particular la solución $\{X \mapsto juan\}$ dado que ya sabemos que `tieneHijas(juan)` se verifica. La resolución que nos conduce a este resultado es:

`tieneHijas(X)` $[\!] \epsilon$	$\Vdash_{\epsilon,\ \text{tieneHijas.1}}$
`padre(X,H), mujer(H)` $[\!] \epsilon$	$\Vdash_{\{X \mapsto juan, H \mapsto maria\},\ \text{padre.1}}$
`mujer(maria)` $[\!] \{X \mapsto juan, H \mapsto maria\}$	$\Vdash_{\epsilon,\ \text{mujer.1}}$
$[\!] \{X \mapsto juan, H \mapsto maria\}$	

Ahora, a pesar de ser $\{X \mapsto juan, H \mapsto maria\}$ la solución calculada, podemos restringirla a las variables del objetivo y obtener la que buscábamos: $\{X \mapsto juan\}$. □

Propiedades de la resolución de objetivos

Ahora necesitamos comprobar que la resolución de objetivos que hemos planteado presenta ciertas propiedades interesantes como son la corrección y la completitud. Antes de ello, veremos que los pasos de cómputo preservan los invariantes de los objetivos, es decir, que el resultado de dar un paso de cómputo a partir de un objetivo es también un objetivo.

Proposición 3.3.1. *Dado un objetivo cualquiera G, se tiene que se preservan los invariantes para todo G' resultante de dar un paso de cómputo, es decir, G' es un objetivo.*

Demostración. Sea $G \equiv \overline{L}, A, \overline{R} \,[\!]\, \sigma$ un objetivo, y sea $G' \equiv (\overline{L}, \overline{B}, \overline{R})\sigma' \,[\!]\, \sigma\sigma'$ el resultado de dar un paso de cómputo a partir de G con $(H \leftarrow \overline{B}) \in_{var} \mathcal{P}$ y $\sigma' \in Sust_\Sigma$ u.m.g. entre A y H. Necesitamos comprobar que (1) $\sigma\sigma'$ es idempotente y (2) $((\overline{L}, \overline{B}, \overline{R})\sigma')\sigma\sigma' = (\overline{L}, \overline{B}, \overline{R})\sigma'$.

(1) Sabemos que σ y σ' cumplen (a) σ es idempotente, (b) σ' es idempotente, (c) $dom(\sigma') \cap dom(\sigma) = \emptyset$ y (d) $vran(\sigma') \cap dom(\sigma) = \emptyset$. (En efecto: por ser σ' u.m.g. de Robinson de A y H cumple que σ' es idempotente y además que $dom(\sigma') \cup vran(\sigma') \subseteq var(A) \cup var(H)$. De esto último se deducen (c) y (d) teniendo en cuenta que $H \leftarrow \overline{B}$ se puede elegir con variables disjuntas de G). De (a), (b), (c) y (d) se deduce por el lema 2.2.1 que $\sigma\sigma' = \sigma\sigma'\sigma\sigma'$.

(2) Es fácil comprobar que $var(\overline{L}, \overline{B}, \overline{R}) \cap dom(\sigma) = \emptyset$ puesto que $H \leftarrow \overline{B}$ se elige sin variables en común con G y $dom(\sigma) \subseteq var(G)$. Sabiendo lo anterior, tenemos que $(\overline{L}, \overline{B}, \overline{R})\sigma' = (\overline{L}, \overline{B}, \overline{R})\sigma\sigma'$. Por (1), $(\overline{L}, \overline{B}, \overline{R})\sigma\sigma' = (\overline{L}, \overline{B}, \overline{R})\sigma\sigma'\sigma\sigma' = (\overline{L}, \overline{B}, \overline{R})\sigma'\sigma\sigma'$.

Y se deduce, dado que se tienen (1) y (2), que G' es un objetivo. □

Notación. Cuando tenemos una resolución de la forma $G_0 \Vdash_{\sigma_1} G_1 \Vdash_{\sigma_2} G_2 \Vdash_{\sigma_3} \cdots \Vdash_{\sigma_n} G_n$ podemos escribir $G_0 \Vdash_\sigma^n G_n$ siendo $\sigma = \sigma_1\sigma_2\cdots\sigma_n$. Además podemos escribir $G \Vdash_\sigma^* G'$ si existe $n \in \mathbb{N}$ tal que $G \Vdash_\sigma^n G'$.

Conocido que el paso de resolución está bien construido, debemos comprobar ahora que las soluciones que podemos calcular son correctas. El teorema de corrección nos permitirá demostrarlo. Para hacerlo, enunciamos primero un lema de corrección que nos facilitará la tarea posterior. El lema nos asegura que si tenemos un objetivo desde el que podemos dar un paso de resolución, y una solución para el objetivo al que llegamos tras el paso de solución, podemos construir una solución para el primero. Formalmente:

Lema 3.3.1 (Lema de Corrección)**.** *Sean G_0 un objetivo no final, G_1 un objetivo cualquiera y θ una sustitución. Entonces, $G_0 \Vdash_{\sigma_1} G_1, \theta \in Sol_{\mathcal{P}}(G_1) \Rightarrow \sigma_1\theta \in Sol_{\mathcal{P}}(G_0)$.*

Demostración. Sea G_0 de la forma $\overline{L}, A, \overline{R} \;[\!]\; \sigma_0$ y sea $\sigma_1 \in Sust_\Sigma$ el u.m.g. entre $\{A, H\}$ siendo $C_1 \equiv (H \leftarrow \overline{B}) \in_{var} \mathcal{P}$ una variante de cláusula del programa sin variables en común con G. Entonces $G_1 \equiv (\overline{L}, \overline{B}, \overline{R})\sigma_1 \;[\!]\; \sigma_0\sigma_1$. Por ser $\theta \in Sol_{\mathcal{P}}(G_1)$ tenemos que:

(1) $\sigma_0\sigma_1\theta = \theta$

(2) $\mathcal{P} \vdash_{HL} (\overline{L}, \overline{B}, \overline{R})\sigma_1\theta$

Para que θ sea solución de G_0, deberá cumplirse que:

(3) $\sigma_0\theta = \theta$

(4) $\mathcal{P} \vdash_{HL} (\overline{L}, A, \overline{R})\theta$

Para ver (3), observamos primero que para toda variable y, si $y \in vran(\sigma_0)$, entonces $y \notin dom(\sigma_0)$ por ser σ_0 idempotente. Entonces, $y \in vran(\sigma_0) \Rightarrow y\sigma_1\theta = y\sigma_0\sigma_1\theta =_{(1)} y\theta$, por lo que tenemos que:

$$(5) \quad \sigma_1\theta = \theta \quad [vran(\sigma_0)]$$

Y ahora podemos ver que:

$x \notin dom(\sigma_0) \Rightarrow x\sigma_0\theta = x\theta$.

$x \in dom(\sigma_0) \Rightarrow var(x\sigma_0) \subseteq vran(\sigma_0) \Rightarrow x\sigma_0\theta =_{(5)} x\sigma_0\sigma_1\theta =_{(1)} x\theta$.

por lo que se tiene (3).

Para ver (4), lo separamos en tres casos y lo vemos para cada uno de ellos:

(4*a*) $\mathcal{P} \vdash_{HL} \overline{L}\theta$. Vemos que $\overline{L}\theta = \overline{L}\sigma_1\theta$ y aplicamos (2). Sabemos que σ_0 no afecta a las variables del objetivo G_0, por lo que $\overline{L}\sigma_1\theta = \overline{L}\sigma_0\sigma_1\theta$, pero $\overline{L}\sigma_0\sigma_1\theta =_{(1)} \overline{L}\theta$.

(4*b*) $\mathcal{P} \vdash_{HL} A\theta$. A partir de (2), sabemos que $\mathcal{P} \vdash_{HL} \overline{B}\sigma_1\theta$. Empleando ahora la cláusula C_1 instanciada por la sustitución $\sigma_1\theta$, podemos ver, en un paso de inferencia HL, $\mathcal{P} \vdash_{HL} H\sigma_1\theta$. Por otra parte, sabemos que $H\sigma_1\theta = A\sigma_1\theta = A\sigma_0\sigma_1\theta = A\theta$ donde la primera igualdad es cierta porque σ_1 es u.m.g. de A y H, la segunda igualdad es cierta porque σ_0 no afecta a las variables de G_0, y la tercera igualdad es cierta por (1). Luego $\mathcal{P} \vdash_{HL} H\sigma_1\theta$ equivale a $\mathcal{P} \vdash_{HL} A\theta$, c.q.d.

(4*c*) $\mathcal{P} \vdash_{HL} \overline{R}\theta$. Por igual razonamiento que en (4a).

por lo que se tiene (4). □

Y ahora estamos en condiciones de enunciar el Teorema de Corrección.

Teorema 3.3.1 (Corrección). *Sean $G_0 \equiv \overline{A_0} \parallel \sigma_0$ un objetivo cualquiera, y G' un objetivo resuelto tal que $G_0 \Vdash_\sigma^* G'$. Entonces $\sigma_0\sigma \in Sol_\mathcal{P}(G_0)$.*

Demostración. Para que sea cierto que $\sigma_0\sigma \in Sol_\mathcal{P}(G_0)$ deben cumplirse dos condiciones:

(1) $\sigma_0\sigma_0\sigma = \sigma_0\sigma$, que es cierto porque σ_0 es idempotente.

(2) $\mathcal{P} \vdash_{HL} \overline{A_0}\sigma_0\sigma$. Como $\overline{A_0}\sigma_0 = \overline{A_0}$, esto equivale a $\mathcal{P} \vdash_{HL} \overline{A_0}\sigma$, que se demuestra por inducción sobre el número de pasos de resolución usando el Lema de Corrección (lema 3.3.1). En efecto:

- **Caso base**
  $G_0 \Vdash_\epsilon^0 G' = G_0$. En este caso G_0 es vacío y $\mathcal{P} \vdash_{HL} \overline{A_0}$ es trivial.
- **Caso inductivo**
  En este caso se tiene $G_0 \equiv \overline{A_0} \parallel \sigma_0 \Vdash_{\sigma_1} G_1 \equiv \overline{A_1} \parallel \sigma_0\sigma_1 \Vdash_{\sigma'}^{n-1} G' \equiv \parallel \sigma_0\sigma_1\sigma'$ siendo $\sigma = \sigma_1\sigma'$. Aplicando la H.I. al cómputo $G_1 \Vdash_{\sigma'}^{n-1} G'$, resulta que $\mathcal{P} \vdash_{HL} \overline{A_1}\sigma'$. Entonces, el Lema de Corrección aplicado al paso de resolución $G_0 \Vdash_{\sigma_1} G_1$ con $\theta = \sigma'$ garantiza que $\mathcal{P} \vdash_{HL} \overline{A_0}\sigma_1\sigma'$, i.e. $\mathcal{P} \vdash_{HL} \overline{A_0}\sigma$. □

En la última parte de esta sección vamos a demostrar que la resolución de objetivos es capaz de calcular todas las soluciones de cualquier objetivo dado. Esto estará garantizado por un teorema de completitud que vamos a demostrar con una técnica similar a la utilizada por *Stärk* en [56]. Como paso previo, comenzamos por demostrar el lema que sigue.

Lema 3.3.2 (Lema de Completitud). *Sea $G_0 \equiv \overline{A_0} \parallel \sigma_0$ un objetivo no resuelto, y sea $\theta_0 \in Sol_\mathcal{P}^n(G_0)$, que cumplirá $\sigma_0\theta_0 = \theta_0$ y $\mathcal{P} \vdash_{HL}^n \overline{A_0}\theta_0$. Sea V_0 cualquier conjunto finito de variables elegido de manera que $var(G_0) \cup dom(\theta_0) \subseteq V_0$. Para cualquier selección arbitraria de un átomo A de $\overline{A_0}$, existe un paso de resolución $G_0 \equiv \overline{A_0} \parallel \sigma_0 \Vdash_{\sigma_1} \overline{A_1} \parallel \sigma_0\sigma_1 \equiv G_1$ usando el átomo elegido, y existe además una sustitución θ_1 con las siguientes propiedades:*

(a) $\theta_1 = \theta_0[V_0]$

(b) $\sigma_1\theta_1 = \theta_1$

(c) $\sigma_0\sigma_1\theta_1 = \theta_1$

(d) $\mathcal{P} \vdash_{HL}^{n-1} \overline{A_1}\theta_1$

En particular, (c) y (d) significan que $\theta_1 \in Sol_\mathcal{P}^{n-1}(G_1)$.

Demostración. Suponiendo que A sea el átomo seleccionado en G_0, podemos suponer que $G_0 \equiv \overline{L_0}, A, \overline{R_0} \parallel \sigma_0$. Por las hipótesis del lema, podemos suponer además:

(0) $\sigma_0\theta_0 = \theta_0$

(1) $\mathcal{P} \vdash_{HL}^{m_1} \overline{L_0}\theta_0$

(2) $\mathcal{P} \vdash_{HL}^{m_2} A\theta_0$

(3) $\mathcal{P} \vdash_{HL}^{m_3} \overline{R_0}\theta_0$

con $m_1 + m_2 + m_3 = n > 0$.

Por (2), deben existir una cláusula $C_1 \equiv (H \leftarrow \overline{B}) \in_{var} \mathcal{P}$ y una sustitución η_0 tales que

(4) $A\theta_0 = H\eta_0$ y $\mathcal{P} \vdash_{HL}^{m_2-1} \overline{B}\eta_0$.

Es posible elegir C_1 y η_0 de modo que se tenga $var(C_1) \cap V_0 = \emptyset$ y $dom(\eta_0) \subseteq var(C_1)$. Con ello, está garantizado que $dom(\eta_0) \cap dom(\theta_0) = \emptyset$, y resulta:

(5) $\theta_1 =_{def} \theta_0 \uplus \eta_0$ es una sustitución bien definida, que cumple: $dom(\theta_1) = dom(\theta_0) \uplus dom(\eta_0)$; $\theta_1 = \theta_0[V_0]$; $\theta_1 = \eta_0[\backslash V_0] \Rightarrow$ (a) del lema.

De (4) y (5) se deduce que θ_1 es un unificador de A y H. Eligiendo σ_1 como u.m.g. canónico (en el sentido de Robinson) de A y H, se tendrá:

(6) $A\sigma_1 = H\sigma_1$ y $\sigma_1\theta_1 = \theta_1 \Rightarrow$ (b) del lema.

Efectuando un paso de resolución con σ_1 y C_1 resulta

$$G_0 \equiv \overline{L_0}, A, \overline{R_0} \;[]\; \sigma_0 \Vdash_{\sigma_1, C_1} (\overline{L_0}, \overline{B}, \overline{R_0})\sigma_1 \;[]\; \sigma_0\sigma_1 \equiv G_1$$

Para completar el lema, sólo nos falta demostrar:

(c) $\sigma_0\sigma_1\theta_1 = \theta_1$

(d) $\mathcal{P} \vdash_{HL}^{n-1} (\overline{L_0}, \overline{B}, \overline{R_0})\sigma_1\theta_1$

- **Demostración de (c)**

  $\sigma_0\underline{\sigma_1\theta_1} =_{(b)} \sigma_0\theta_1 =_{(5)} \sigma_0(\theta_0 \uplus \eta_0) =_{(*)} \sigma_0\theta_0 \uplus \eta_0 =_{(0)} \theta_0 \uplus \eta_0 = \theta_1$.

  El paso $(*)$ se puede justificar porque $vran(\sigma_0) \subseteq V_0$ y $dom(\eta_0) \cap V_0 = \emptyset$.

- **Demostración de (d)**

  En primer lugar, observamos que se tiene:

  (7) $\mathcal{P} \vdash_{HL}^{m_1} \overline{L_0}\theta_1$ por (1) y por (5)

  (8) $\mathcal{P} \vdash_{HL}^{m_2-1} \overline{B}\theta_1$ por (4) y por (5)

  (9) $\mathcal{P} \vdash_{HL}^{m_3} \overline{R_0}\theta_1$ por (3) y por (5)

  Considerando que $m_1 + (m_2 - 1) + m_3 = n - 1$, (7), (8), (9) implican que $\mathcal{P} \vdash_{HL}^{n-1} (\overline{L_0}, \overline{B}, \overline{R_0})\theta_1$; y esto equivale a (d) debido a (b). □

Teorema 3.3.2 (Completitud)**.** *Dados un objetivo* $G_0 \equiv \overline{A_0} \;[\!]\; \sigma_0$ *y una solución* $\theta_0 \in Sol_{\mathcal{P}}(G_0)$*, existe un cómputo* $G_0 \Vdash^*_\sigma \;[\!]\; \sigma_0\sigma$ *que se puede realizar con cualquier estrategia de selección, tal que* $\sigma \preccurlyeq \theta_0[var(G_0)]$ *e incluso* $\sigma_0\sigma \preccurlyeq \theta_0[var(G_0)]$*.*

Demostración. Por ser G_0 un objetivo bien formado, se puede suponer que σ_0 es idempotente y que $\overline{A_0}\sigma_0 = \overline{A_0}$. Y por ser $\theta_0 \in Sol_{\mathcal{P}}(G_0)$, se puede elegir un número $n \in \mathbb{N}$ tal que $\theta_0 \in Sol^n_{\mathcal{P}}(G_0)$, lo cual quiere decir que se tiene

(1) $\sigma_0\theta_0 = \theta_0$

(2) $\mathcal{P} \vdash^n_{HL} \overline{A_0}\theta_0$

Se puede elegir también un conjunto finito V_0 de variables que verifique:

(3) $var(G_0) \cup dom(\theta_0) \subseteq V_0$

A partir de las condiciones (1), (2) y (3), vamos a demostrar lo siguiente:

(†) Existen un cómputo $G_0 \Vdash^*_\sigma \;[\!]\; \sigma_0\sigma$ (que se puede realizar con cualquier estrategia de selección) y una sustitución θ que verifica: (4) $\theta = \theta_0[V_0]$ (5) $\sigma\theta = \theta$ (6) $\sigma_0\sigma\theta = \theta$.

De (†) se deduce la tesis del teorema, ya que:

(5) $\Rightarrow_{(4)} \sigma\theta = \theta_0[V_0] \Rightarrow_{(3)} \sigma \preccurlyeq \theta_0[var(G_0)]$

(6) $\Rightarrow_{(4)} \sigma_0\sigma\theta = \theta_0[V_0] \Rightarrow_{(3)} \sigma_0\sigma \preccurlyeq \theta_0[var(G_0)]$

La demostración de (†) es por inducción sobre n:

- **Caso base**

  Si $n = 0$, (2) implica que $\overline{A_0}$ debe ser vacío. Entonces, tomando $\sigma = \epsilon$ y $\theta = \theta_0$ se tiene que $G_0 \Vdash^*_\epsilon \;[\!]\; \sigma_0$ con un cómputo trivial de 0 pasos; y además:

  (4) se reduce a $\theta_0 = \theta_0[V_0]$, que es trivial;

  (5) se reduce a $\theta_0 = \theta_0$, que también es trivial;

  (6) se reduce a $\sigma_0\theta_0 = \theta_0$ que se cumple por (1).

- **Caso inductivo**

  Si $n > 0$, (2) implica que $\overline{A_0}$ debe ser no vacío. Seleccionando un átomo A de $\overline{A_0}$ con cualquier estrategia, y aplicando el Lema de Completitud (lema 3.3.2), podemos realizar un paso de resolución

  (7) $G_0 \equiv \overline{A_0} \;[\!]\; \sigma_0 \Vdash_{\sigma_1} \overline{A_1} \;[\!]\; \sigma_0\sigma_1 \equiv G_1$

  de manera que exista una solución $\theta_1 \in Sol^{n-1}_{\mathcal{P}}(G_1)$ que cumpla las cuatro condiciones aseguradas por el lema. Aquí las numeramos como (8), (9), (1') y (2'):

  (8) $\theta_1 = \theta_0[V_0]$

  (9) $\sigma_1\theta_1 = \theta_1$

(1') $\sigma_0\sigma_1\theta_1 = \theta_1$

(2') $\mathcal{P} \vdash_{HL}^{n-1} \overline{A_1}\theta_1$

Sea V_1 cualquier conjunto finito de variables elegido de manera que se cumpla:

(3') $V_0 \cup var(G_1) \cup dom(\theta_1) \subseteq V_1$

Las condiciones (1'), (2') y (3') son del mismo estilo que (1), (2) y (3), pero ahora para $\theta_1 \in Sol_{\mathcal{P}}^{n-1}(G_1)$. Aplicando la H.I., podemos obtener un cómputo

(10) $G_1 \Vdash_{\sigma'}^{*} [] \sigma_0\sigma_1\sigma'$

y una sustitución θ que cumpla

(4') $\theta = \theta_1[V_1]$ (5') $\sigma'\theta = \theta$ (6') $\sigma_0\sigma_1\sigma'\theta = \theta$

De (7) y (10) resulta el cómputo

$$G_0 \equiv \overline{A_0} [] \sigma_0 \Vdash_{\sigma_1} G_1 \equiv \overline{A_1} [] \sigma_0\sigma_1 \Vdash_{\sigma'}^{*} [] \sigma_0\sigma_1\sigma'$$

Solo nos falta demostrar que (4), (5) y (6) se verifican si se toma la misma θ que cumple (4'), (5') y (6') y $\sigma = \sigma_1\sigma'$. En efecto:

- (4) es consecuencia inmediata de (4'), (3') y (8).
- (5) se razona así: por (4'), se puede suponer $\theta = \theta_1 \uplus \eta'$, con η' tal que $dom(\eta') \cap (V_1) = \emptyset$. Entonces: $\underline{\sigma}\theta = \sigma_1\underline{\sigma'\theta} =_{(5')} \sigma_1\underline{\theta} = \sigma_1(\theta_1 \uplus \eta') =_{(*)} \underline{\sigma_1\theta_1} \uplus \eta' =_{(9)} \theta_1 \uplus \eta' = \theta$. El paso $(*)$ se puede justificar porque $vran(\sigma_1) \subseteq V_1$ y $dom(\eta') \cap V_1 = \emptyset$.
- (6) es consecuencia trivial de (6'), ya que $\sigma = \sigma_1\sigma'$. □

Capítulo 4

Programación Lógica con Incertidumbre: Una Panorámica[1]

La programación lógica surgió en el siglo XX a partir de la idea de utilizar la lógica como lenguaje de programación. Sus fundamentos fueron desarrollados a partir de la década de los años 70, e incluyen diversas caracterizaciones de la semántica declarativa de los programas lógicos, así como la investigación de procedimientos de resolución de objetivos. Las referencias clásicas en este campo son [1] *Apt* y [40] *Lloyd*, pero existe una amplia bibliografía posterior, incluyendo nuevos avances en la semántica declarativa como los referidos en [2] y técnicas optimizadas para demostrar la corrección y completitud de los procedimientos de resolución de objetivos como la presentada en [56].

Se han desarrollado asimismo diferentes extensiones de la programación lógica, combinándola con el cálculo eficiente de restricciones sobre dominios específicos [33, 34], la programación funcional [29] y ambas cosas simultáneamente [15, 42, 52]. Algunos lenguajes de programación declarativa multiparadigma actuales, tales como [30] *Curry* y [5] $\mathcal{TOY}$, han sido diseñados e implementados para soportar las extensiones.

La toma en consideración de la Incertidumbre como parte de la Programación Lógica surge poco tiempo después de sentarse las bases de la Programación Lógica Clásica. Los primeros trabajos serios en este campo datan de principios de los años 80, cuando la necesidad de representar y trabajar con conocimiento incierto se hace palpable en áreas diversas de aplicación de la Programación Lógica: es precisamente en el campo de la Inteligencia Artificial y los Sistemas Expertos donde se realizan los primeros avances en esta materia.

Ehud Shapiro [55] definió en 1983 la certidumbre como un número real en el intervalo (0,1], y estudió programas lógicos en los que hacía corresponder una función de certidumbre con cada una de las reglas del programa; esta función de certidumbre era una aplicación de multiconjuntos de factores de certidumbre a factores de certidumbre. De forma intuitiva, esta función debía obtener los factores de certidumbre asociados a los átomos del cuerpo de la regla, y dar el factor de certidumbre que se asociaría con la cabeza.

En 1986, *Maarten H. van Emden* publicó el artículo [61] *Quantitative Logic Programming*

[1] La base y estructura principal de este capítulo se debe a *V.S. Subrahmanian* por su artículo [59] *Uncertainty in Logic Progamming: Some Recollections.*

que extendía los artículos de van Emden y Kowalski [62] y Apt y van Emden [3] que suponían el fundamento de la Programación Lógica. Junto con este trabajo de van Emden, aparecen otros como [7, 8, 32]. Todos estos trabajos se basaban en la sintaxis introducida por Shapiro pero sustituían la función de certidumbre asociada a cada regla del programa por un único factor de certidumbre que resultaba más sencillo de emplear. La principal aportación del trabajo de van Emden frente a los demás fue una teoría de punto fijo y otra de prueba basada en árboles de juego que se demostraba correcta y completa bajo ciertas restricciones. Este conjunto de trabajos vienen a dar lugar a lo que se conoce con el nombre de *Programación Lógica Cuantitativa*.

Ya en 1987, *V.S. Subrahmanian* dio a conocer en [57] *On the Semantics of Quantitative Logic Programs* una innovadora noción de Programa Lógico Cuantitativo en la que el espacio de valores veritativos formaba un retículo completo en el conjunto $[0, 1] \cup \{*\}$ y en el que $*$ (inconsistencia) era el máximo valor del retículo y $0,5$ (ignorancia) el mínimo; creciendo el resto de valores de certidumbre hacia el 0 o hacia el 1 según aumentara el grado de certeza de falsedad o veracidad, respectivamente, del átomo en cuestión. Esto se correspondía con un orden de "conocimiento" y aportaba la base para una semántica paraconsistente (la inconsistencia no permitía derivar cualquier cosa). Este mismo trabajo fue el primero en permitir la aparición explícita de una forma de negación y soportaba la anotación de todos los átomos del cuerpo de las reglas, lo que de facto daba lugar a la primera *lógica anotada*. Posteriormente en 1987 y 1988, y junto a *Howard Blair*, propuso otras lógicas anotadas en [11, 12] en las que los valores veritativos podrían tomarse de cualquier retículo, y demostraron que estas lógicas anotadas capturaban todos los métodos anteriores de trabajo con la incertidumbre. Podía hablarse en este momento de la *Programación Lógica Anotada* que suponía una generalización de la Cuantitativa. Muchos trabajos más se escribieron sobre la forma de manejar la incertidumbre en la Programación Lógica que pueden englobarse bien en la Programación Lógica Cuantitativa, bien en la Anotada; algunos de estos trabajos son [24, 36] de los que en particular el de *Melvin Fitting* hablaba de programas lógicos basados en birretículos que requerían que el conjunto de valores veritativos fueran retículos completos bajo un orden de "conocimiento" y otro de "verdad".

A pesar de la cantidad de trabajos relacionados con la incertidumbre, no es hasta una década después cuando aparecen las primeras aproximaciones para tratar la incertidumbre a través de teorías probabilísticas como se hace en [23, 47, 48], dando lugar a un nuevo enfoque: la *Programación Lógica Probabilística*. A principio de la década de los 90, *Raymond Ng* y el propio Subrahmanian, inspirados en un trabajo de Fagin, Halpern y Meggido [21] (a su vez basado en otro anterior de Hailperin [28]), propusieron un lenguaje de programación lógica probabilística basado en intervalos que hacía uso de una sintaxis del estilo de las lógicas anotadas pero con semántica probabilística en sentido matemático. Presentaron en [44, 45, 46] un modelo teórico y una semántica de punto fijo, junto a un procedimiento correcto y completo de resolución de objetivos. Prácticamente al mismo tiempo, Lakshmanan escribía una serie de artículos [38, 39] en los que proponía el concepto de trirretículo añadiendo un orden de "precisión" a los dos anteriormente empleados por Fitting.

Poco después surgen trabajos que intentan combinar los enfoques cuantitativos y probabilísticos mediante el cálculo de restricciones sobre dominios específicos, dando lugar a propuestas de *programación lógica con restricciones e incertidumbre* como las consideradas en [49, 50]. En esa época la Programación Lógica con Incertidumbre estaba preparada para dar

un salto cualitativo a campos diferentes en los que la investigación se centra en la actualidad:

- Por un lado, la incertidumbre entraba en el mundo de las bases de datos: el concepto de *base de datos probabilística* surgía de la mano de Lakshmanan, Leone, Ross y Subrahmanian en [37], en el que extendían el álgebra relacional para incluir información probabilística y construir el primer gestor de bases de datos probabilístico (llamado ProbView). Tras éste, una serie de trabajos crean los conceptos de base de datos probabilística-temporal [14, 43] y se extiende ProbView para adaptarse a la orientación a objetos [10, 19].
- Y por el otro lado, ciertos trabajos [16, 17, 20] de investigación en programación lógica con incertidumbre se centraban en Agentes Probabilísticos que se debían a la necesidad de modelizar agentes capaces de razonar en presencia de incertidumbre a la hora del acceso eficiente a datos o módulos software.

Además, junto a las líneas de investigación anteriormente citadas, surgen nuevos enfoques para el tratamiento de la incertidumbre: Dignos de mención son trabajos como [4, 26, 41] que introducen la incertidumbre en la misma operación de unificación empleada en la resolución de objetivos, dando lugar a una *unificación flexible* basada en la similitud entre símbolos. Este enfoque se engloba en la denominación genérica de *programación lógica cuantitativa* o en la más específica *programación lógica borrosa*.

Los dos capítulos siguientes constituyen el núcleo de este trabajo. Aportan una presentación revisada y mejorada de los fundamentos y la implementación del lenguaje de programación lógica cuantitativa propuesto en el trabajo clásico de *van Emden* [61]. Otras líneas de trabajo futuro se esbozarán en el capítulo de conclusiones.

Capítulo 5

Programación Lógica Cuantitativa

Como hemos visto en el capítulo anterior, el artículo clásico de *van Emden* [61] fue un hito importante en la formalización de un lenguaje de programación lógica con tratamiento cuantitativo de la incertidumbre. En este capítulo se presenta una versión revisada de la semántica propuesta en [61], introduciendo algunas modificaciones que intentan mejorarla. Las dos modificaciones más importantes son las siguientes:

1. En la semántica declarativa, se considerarán interpretaciones sobre el universo de Herbrand abierto.

2. Y en la semántica operacional, se considerará la resolución para objetivos arbitrarios en lugar de árboles *AND-OR* para objetivos atómicos cerrados. Además, como veremos en la sección 5.3, se emplearán restricciones aritméticas reales para poder optimizar el proceso de resolución y expresar las respuestas calculadas.

Para facilitar el desarrollo y la compresión de este capítulo, seguiremos un proceso similar al seguido en el capítulo 3, ya que aquél fue realizado de forma que la incorporación de la incertidumbre fuera fácilmente abordable. Así, casi todos los resultados propuestos en este capítulo tienen un reflejo directo en la versión cualitativa del lenguaje. Se omitirán también, por este mismo motivo, algunas explicaciones sobre el proceso seguido pues son igualmente válidas las ya dadas en el capítulo 3.

La versión cuantitativa de nuestro lenguaje lógico es básicamente la cualitativa una vez añadimos ciertos factores de certidumbre a las reglas y objetivos. En el caso que nos ocupa, una cláusula cuantitativa tendrá la forma $A \leftarrow \alpha - B_1, \ldots, B_n$ (abreviada como $A \leftarrow \alpha - \overline{B}$) siendo α un número real del intervalo $(0, 1]$ y donde α es el factor de certidumbre asociado a la cláusula. De manera informal, podemos decir que la certidumbre con la que podemos demostrar la cabeza de una cláusula es la certidumbre con la que hemos demostrado el cuerpo afectado multiplicativamente por el factor de certidumbre de la cláusula empleada, es decir, si para una cláusula $A \leftarrow 0{,}5 - \overline{B}$ tenemos que podemos encontrar una demostración de $\overline{B}$ para una certidumbre de 0,8, podremos decir que la cláusula anterior nos permite inferir que A se tendrá con una certidumbre de $0{,}4 = 0{,}5 \cdot 0{,}8$. Veremos todo este proceso con mayor profundidad y exactitud a lo largo del capítulo.

5.1. Sintaxis

Un programa cuantitativo $\mathcal{P}$[1] es una conjunto finito de cláusulas cuantitativas.

Notación. Al igual que en la versión anterior, emplearemos sintaxis PROLOG con la salvedad de emplear `<-c-` (o simplemente `<-` si `c` = 1) en vez de `:-`, siendo `c` el factor de certidumbre asociado a la cláusula.

A continuación mostramos dos posibles ejemplos de programas que hacen uso de la incertidumbre y con los que trabajaremos a lo largo de este capítulo.

Ejemplo 5.1.1 (Circuitos fiables)**.** El siguiente programa permite construir circuitos electrónicos a partir de componentes básicas mediante la composición, secuencial o paralela, de dos circuitos.

```
fi(cb(Id,acme))  <-0.99-
fi(cb(Id,tigre)) <-0.95-
fi(cb(Id,panda)) <-0.90-

fi(cs(C1,C2)) <-0.96- fi(C1), fi(C2)

fi(cp(C1,C2)) <-0.94- fi(C1), fi(C2)
fi(cp(C1,C2)) <-0.50- fi(C1)
fi(cp(C1,C2)) <-0.50- fi(C2)
```

Los tres primeros hechos, son los que representan a las componentes básicas de nuestros circuitos. Su fiabilidad es diferente dado que depende de la de cada fabricante (se tratará de la fiabilidad que le concedamos a cada fabricante). El resto de reglas del programa nos permiten realizar la composición, primero secuencial y luego paralela, de dos circuitos cualesquiera y su factor de certidumbre reflejaría la distinta complejidad y seguridad del proceso de fabricado del circuito final. De estas cuatro, las dos últimas, aunque viables por tratarse de una composición paralela, reducen mucho la fiabilidad del circuito final debido al desconocimiento de uno de los dos circuitos de la composición.

Como se puede observar, a medida que se componen los circuitos, su fiabilidad desciende. Por esto podemos asegurar que el número de circuitos con una fiabilidad mayor o igual a una certidumbre dada es necesariamente finito. □

Ejemplo 5.1.2 (Maltratadores)**.** Para este ejemplo nos centraremos en la caracterización de un rasgo personal. Definiremos primero un predicado `cruel` que nos indicará cuándo una persona es considerada cruel o no. Esto dependerá de si maltrata o no a animales y vegetales y de la certidumbre con la que podamos suponer que cada persona lo hace. Veamos el programa:

```
cruel(X) <-0.90- humano(X), maltrata(X,Y), animal(Y)
cruel(X) <-0.40- humano(X), maltrata(X,Y), vegetal(Y)
```

[1] Emplearemos nombres idénticos a la versión cualitativa siempre que sea posible por facilidad, dado que en la mayoría de los casos el contexto es suficiente para conocer si nos referimos a una u otra versión.

```
humano("rufo") <-
humano("petronila") <-
humano("romeo") <-
humano("julieta") <-
humano(Z) <- engendran(X,Y,Z), humano(X), humano(Y)

engendran("rufo", "petronila", "carlota") <-
engendran("romeo", "julieta", "pedro") <-
engendran("pedro", "carlota", "crispin") <-

animal("piolin") <-
animal("silvestre") <-
vegetal("alcornoque") <-
vegetal("lechuguino") <-

maltrata("rufo", X)      <-0.90- animal(X)
maltrata("petronila", X) <-0.75- animal(X)
maltrata("romeo", X)     <-0.40- animal(X)
maltrata("julieta", X)   <-0.30- animal(X)

maltrata("rufo", X)      <- vegetal(X)
maltrata("petronila", X) <-0.95- vegetal(X)
maltrata("romeo", X)     <-0.75- vegetal(X)
maltrata("julieta", X)   <-0.60- vegetal(X)

maltrata(X,Y) <-0.70- engendran(A,B,X),
                      maltrata(A,Y),
                      maltrata(B,Y)
```

La definición del predicado `cruel` depende principalmente del predicado `maltrata`. Tanto `humano` como `animal` y `vegetal` se corresponden con una base de conocimiento con la distinción de que `humano` se define de forma recursiva (los hijos de dos humanos son también humanos) para poder simular posteriormente la herencia del rasgo.

La definición de `maltrata` consta de la certidumbre conocida sobre la cual podemos basarnos para asegurar que un determinado humano maltrata a un animal o a un vegetal, siendo de especial interés la última de las reglas, la de la herencia del rasgo, que dependerá de las características particulares de cada uno de sus progenitores (aquí, con la intención de simplificar el ejemplo, se entiende que una persona maltrata a un animal o a una planta si sus dos progenitores lo hacen). En esta última regla, el factor de certidumbre se correspondería intuitivamente con un factor de atenuación del rasgo (por esto no puede suponerse la perpetuación infinita del rasgo sin que algún nuevo humano interfiera en el proceso -añadiendo una nueva regla para `maltrata` del estilo de las primeras-). □

Llamaremos a un número real $p \in (0, 1]$, pensado para representar un grado de certidumbre, *peso*; y a W *variable de peso*. Por analogía con el conjunto de todas las variables ($\mathcal{V}$), postularemos un conjunto infinito numerable $\mathcal{W}$ formado por todas las variables

de peso, y supondremos que $\mathcal{V} \cap \mathcal{W} = \emptyset$. Para las variables de peso definiremos también un tipo particular de sustituciones: las sustituciones de variables de pesos. Diremos que $\omega : \mathcal{W} \to [0,1]$ es una *sustitución de pesos*, y llamaremos $Sust_W$ al conjunto de todas las sustituciones de pesos. Dada una sustitución $\omega \in Sust_W$, definimos su *dominio* como el conjunto $dom(\omega) =_{def} \{W \in \mathcal{W} \mid \omega(W) \neq 0\}$. El motivo fundamental de definir el dominio de esta forma es que tener un factor de certidumbre de 0 asociado a un átomo equivale a no saber nada del átomo y, por este mismo motivo, resulta que cualquier átomo se verifica para una certidumbre de 0, por lo que no resulta de interés asignar explícitamente el valor 0 a una variable.

Notación. En adelante y siempre que E sea una expresión formal que contenga variables de peso, pondremos $E\omega$ para referirnos al resultado de reemplazar cada variable de peso W que aparezca en E por $\omega(W)$.

Todos los átomos de las cláusulas serán átomos como en el caso cualitativo, sin embargo, tendremos en los objetivos lo que llamaremos *átomos anotados* por un peso que será o bien un cierto $p \in (0,1]$, o bien una variable de peso $W \in \mathcal{W}$. Escribiremos $A \sharp p$ o $A \sharp W$ para referirnos a un átomo A anotado por un peso p, o a un átomo A anotado por una variable de peso W respectivamente. Dada una conjunción de átomos anotados $\overline{A}$ (nótese que la notación no varía con respecto a la de las conjunciones de átomos cualitativos[2]) se define el conjunto de variables de $\overline{A}$ como $var(\overline{A}) =_{def} \{x \in \mathcal{V} \mid x \in var(A), A \sharp W \in \overline{A}\}$; y el conjunto de variables de peso de $\overline{A}$ como $war(\overline{A}) =_{def} \{W \in \mathcal{W} \mid A \sharp W \in \overline{A}\}$.

Llamaremos *base de Herbrand abierta* (At_Σ) al conjunto de todos los átomos de la signatura Σ y *base de Herbrand abierta anotada* (QAt_Σ) al conjunto de todos los átomos de la signatura Σ anotados con todos los posibles pesos. Formalmente tendremos que $QAt_\Sigma =_{def} \{A \sharp p \mid A \in At_\Sigma, p \in (0,1]\}$. Ahora, una *interpretación [de Herbrand anotada]*, que representaremos también por $\mathcal{I}$, será un subconjunto de átomos de QAt_Σ que cumpla que si $A \sharp p \in \mathcal{I}$, $\sigma \in Sust_\Sigma$ y $0 < q \leq p$, entonces $A\sigma \sharp q \in \mathcal{I}$, siendo p y q pesos. Es decir, que ha de ser cerrada bajo la aplicación de sustituciones al igual que en el caso cualitativo y que si un cierto átomo A aparece anotado con un peso p, entonces aparecerá también anotado con todo peso $q \in (0,p]$. La idea que hay detrás de esto último viene dada por lo que entendemos a partir de un átomo anotado que pertenece a una determinada interpretación. Que se verifique en una interpretación $\mathcal{I}$ un átomo anotado $A \sharp p$ significa que A es cierto para un factor de certeza de p; o que podemos suponer A cierto con una certidumbre de p. De aquí se deduce inmediatamente que A será cierto también para cualquier factor de certidumbre menor o igual que p, por lo que si $q \in (0,p]$, podremos asegurar que $A \sharp q$ pertenece a la interpretación $\mathcal{I}$.

Al conjunto de todas las interpretaciones de Herbrand anotadas de un programa lo llamaremos $QInt_\Sigma$.

Dado un programa cuantitativo $\mathcal{P}$ y una interpretación $\mathcal{I} \in QInt_\Sigma$, se dice que $\mathcal{I}$ es *modelo* de $\mathcal{P}$ (y escribimos $\mathcal{I} \models \mathcal{P}$) cuando se tiene que $A\sigma \sharp p \in \mathcal{I}$ para toda cláusula $(A \leftarrow \alpha - B_1, \ldots, B_n) \in \mathcal{P}$ y para toda sustitución $\sigma \in Sust_\Sigma$ tales que $B_1\sigma \sharp q_1, \ldots, B_n\sigma \sharp q_n \in \mathcal{I}$ siendo $q_1, \ldots, q_n$ pesos y donde $p = \alpha \cdot min\{q_1, \ldots, q_n\}$. Llamaremos $QHMod(\mathcal{P})$ al conjunto de todos los *modelos [de Herbrand anotados]* de $\mathcal{P}$.

Obsérvese que si $\mathcal{I} \models \mathcal{P}$, entonces para cualquier cláusula $(A \leftarrow \alpha - B_1, \ldots, B_n) \in \mathcal{P}$, cualquier sustitución $\sigma \in Sust_\Sigma$ y cualquier elección de pesos $q_1, \ldots, q_n$ tales que $B_1\sigma \sharp q_1, \ldots,$

[2]Por este motivo su uso será menos frecuente que la versión extendida $A_1 \sharp W_1, \ldots, A_n \sharp W_n$.

$B_n\sigma \sharp q_n \in \mathcal{I}$ se cumplirá $A\sigma \sharp q \in \mathcal{I}$ para *cualquier* peso p tal que $0 < p \leq \alpha \cdot min\{q_1, \ldots, q_n\}$, y no sólo para $p = \alpha \cdot min\{q_1, \ldots, q_n\}$, debido a las condiciones de cierre que se han exigido a las interpretaciones de Herbrand anotadas. Este hecho se dará por supuesto en adelante siempre que se necesite para cualquier razonamiento referente a modelos.

Inferencia a partir de un programa lógico cuantitativo

De igual forma que ocurría en el caso cualitativo, tendremos dos formas de emplear un programa cuantitativo para obtener resultados a partir de él: la inferencia y la resolución. Veremos a continuación cuáles son las variaciones respecto del caso cualitativo.

Para poder realizar inferencias a partir de un programa lógico cuantitativo dado, emplearemos una modificación del Cálculo Lógico de Horn que llamaremos QHL (por *Quantitative Horn Logic*). Al igual que en el caso cualitativo, este cálculo tendrá una única regla de inferencia que llamaremos *Quantitative Modus Ponens* que nos permite inferir a partir del cuerpo de una cláusula su cabeza con cualquier factor de certidumbre mayor que cero y menor o igual que el producto del factor de certidumbre de la cláusula en cuestión por el mínimo de las certidumbres con las que se demuestran los átomos del cuerpo. Formalmente la regla es la siguiente.

Sea $A \leftarrow \alpha - B_1, \ldots, B_n$ una cláusula del programa y $\sigma \in Sust_\Sigma$ una sustitución. Entonces:

$$\frac{B_1\sigma \sharp q_1 \quad \cdots \quad B_n\sigma \sharp q_n}{A\sigma \sharp p} \quad \text{si } 0 < p \leq \alpha \cdot min\{q_1, \ldots, q_n\} \quad QMP$$

donde, por convenio, admitimos que $min\{\} = 1$, motivado por el hecho de que 1 es el máximo del intervalo $(0, 1]$. Este convenio hace que sea válida la regla de cómputo $min\,(\{p\} \uplus P) = min(p, min\,P)$ sea cual sea el conjunto de pesos P y el peso $p \notin P$.

Ejemplo 5.1.3. A partir del ejemplo de los Circuitos fiables (ejemplo 5.1.1), intentaremos comprobar que el átomo anotado

```
fi(cs(cb("T01",tigre), cb("A07",acme)))#0.90
```

es deducible de las cláusulas del programa. En este caso, y a diferencia del caso cualitativo, debemos asegurarnos para aplicar cada inferencia QHL que se cumple la condición de la regla QMP. Veamos una posible solución:

$$\dfrac{\dfrac{}{\texttt{fi(cb("T01",tigre))\#q1}}\,(2) \qquad \dfrac{}{\texttt{fi(cb("A07",acme))\#q2}}\,(3)}{\texttt{fi(cs(cb("T01",tigre), cb("A07",acme)))\#0.90}}\,(1)$$

donde las inferencias son:

(1) $(\text{cs.1})_{\sigma_1=\{C1\mapsto\texttt{cb("T01",tigre)},C2\mapsto\texttt{cb("A07",acme)}\}}$ si $0{,}90 \leq 0{,}96 \cdot min\{q1, q2\}$

(2) $(\text{cb.2})_{\sigma_2=\{Id\mapsto\texttt{"T01"}\}}$ si $q1 \leq 0{,}95$

(3) $(\text{cb.1})_{\sigma_3=\{Id\mapsto\texttt{"A07"}\}}$ si $q2 \leq 0{,}99$

Y tomamos entonces el mejor resultado posible para $q1$ y $q2$ y comprobamos que se cumplen las tres condiciones: $q1 = 0{,}95$ y $q2 = 0{,}99 \Rightarrow 0{,}90 \leq 0{,}96 \cdot min\{0{,}95; 0{,}99\} = 0{,}912$. Con esto hemos verificado que el átomo se cumple para un factor máximo de certidumbre de 0,912; y en particular para 0,90 como queríamos verificar. □

Ejemplo 5.1.4. A partir del ejemplo de los Maltratadores (ejemplo 5.1.2), intentaremos comprobar que el átomo anotado

```
maltrata("carlota","piolin")#0.50
```

es deducible de las cláusulas del programa. El siguiente árbol de inferencia sirve como prueba[3]:

$$\dfrac{\dfrac{}{\texttt{engendran("rf","pt","carlota")\#q1}}\,(2) \quad \dfrac{\dfrac{}{\texttt{animal("piolin")\#q4}}\,(5)}{\texttt{maltrata("rf","piolin")\#q2}}\,(3) \quad \dfrac{\dfrac{}{\texttt{animal("piolin")\#q5}}\,(6)}{\texttt{maltrata("pt","piolin")\#q3}}\,(4)}{\texttt{maltrata("carlota","piolin")\#0.50}}\,(1)$$

donde las inferencias son:

(1) $(\text{maltrata.9})_{\sigma_1=\{X\mapsto\texttt{"carlota"}, Y\mapsto\texttt{"piolin"}, A\mapsto\texttt{"rf"}, B\mapsto\texttt{"pt"}\}}$ si $0{,}50 \leq 0{,}70 \cdot min\{q1, q2, q3\}$

(2) $(\text{engendran.1})_{\sigma_2=\epsilon}$ si $q1 \leq 1$

(3) $(\text{maltrata.1})_{\sigma_3=\{X\mapsto\texttt{"piolin"}\}}$ si $q2 \leq 0{,}90 \cdot min\{q4\}$

(4) $(\text{maltrata.2})_{\sigma_4=\{X\mapsto\texttt{"piolin"}\}}$ si $q3 \leq 0{,}75 \cdot min\{q5\}$

(5) $(\text{animal.1})_{\sigma_5=\epsilon}$ si $q4 \leq 1$

(6) $(\text{animal.1})_{\sigma_6=\epsilon}$ si $q5 \leq 1$

Tomando ahora los valores máximos para $q1$, $q4$ y $q5$, obtenemos el mejor valor para los demás: $q2 = 0{,}90$ y $q3 = 0{,}75$. Puesto que $0{,}50 \leq 0{,}525$, la deducción del átomo anotado propuesto tiene éxito. □

Cuando podemos demostrar la validez de un átomo anotado $A\,\sharp\,p$ en QHL, podemos escribir $\mathcal{P} \vdash_{QHL} A\,\sharp\,p$ lo que quiere decir que existe demostración en QHL para demostrar la validez del átomo (esta demostración tiene forma de árbol). Además, podemos indicar el número exacto de inferencias QHL empleadas en la demostración poniendo $\mathcal{P} \vdash^n_{QHL} A\,\sharp\,p$ que indica que se han relizado exactamente n inferencias QHL (o que el árbol de demostración tiene exactamente n nodos).

Notación. Dada una conjunción de átomos anotados $\overline{A}$, empleamos también la notación $\mathcal{P} \vdash_{QHL} \overline{A}$ para indicar que existe demostración (esta vez un bosque) para todos los átomos anotados $A\,\sharp\,p \in \overline{A}$. De igual modo, con $\mathcal{P} \vdash^n_{QHL} \overline{A}$ indicamos que el número de inferencias QHL realizadas para demostrar todos los átomos anotados de la conjunción es n (la suma del número de nodos de todos los árboles del bosque es n).

[3]Para mejorar la legibilidad, abreviaremos `"rufo"` como `"rf"` y `"petronila"` como `"pt"`.

Tendremos también la versión cuantitativa del operador $T_{\mathcal{P}}$ definido para el caso cualitativo. En esta ocasión, el operador $T_{\mathcal{P}} : QInt_{\Sigma} \to QInt_{\Sigma}$ queda definido como sigue:

$$\begin{aligned} T_{\mathcal{P}}(\mathcal{I}) =_{def} \{A\sigma \sharp p \mid\ & \sigma \in Sust_{\Sigma}, \\ & (A \leftarrow \alpha - B_1, \ldots, B_n) \in \mathcal{P}, \\ & \{B_i\sigma \sharp q_i \mid 1 \leq i \leq n\} \subseteq \mathcal{I} \text{ para ciertos } q_1, \ldots, q_n \text{ pesos}, \\ & 0 < p \leq \alpha \cdot min\{q_1, \ldots, q_n\}\} \end{aligned}$$

Intuitivamente podemos ver que para una interpretación $\mathcal{I}$, $T_{\mathcal{P}}(\mathcal{I})$ serán aquellos átomos anotados que son cabeza de una cláusula cuyo cuerpo aparece en $\mathcal{I}$ para unos ciertos pesos, siendo el peso de la cabeza menor o igual (y estrictamente mayor que cero) que el producto del factor de certidumbre de la cláusula por el mínimo de los pesos de los átomos del cuerpo.

5.2. Semántica declarativa

Mediante la semántica declarativa de nuestros programas lógicos cuantitativos podremos encontrar un modelo mínimo para cada programa y caracterizar las soluciones válidas para cada objetivo. En este caso, para realizar inferencias emplearemos la lógica QHL que definimos en la sección anterior, así como el operador $T_{\mathcal{P}}$ con su nueva definición.

Partiremos también de un programa lógico cuantitativo $\mathcal{P}$ para la signatura Σ definida según se introdujo en el capítulo 2. Dado que las interpretaciones de Herbrand anotadas son también conjuntos de átomos anotados comprobamos primero que el conjunto $QInt_{\Sigma}$ de las interpretaciones de Herbrand anotadas sigue siendo un retículo completo bajo el orden parcial $\subseteq$.

Proposición 5.2.1. *Dado un programa $\mathcal{P}$ y el conjunto $QInt_{\Sigma}$ de las posibles interpretaciones de Herbrand, $(QInt_{\Sigma}, \subseteq)$ conforma un retículo completo (el retículo de las interpretaciones de Herbrand anotadas). Además, para cualquier subconjunto $I \in QInt_{\Sigma}$ de interpretaciones de $\mathcal{P}$, se cumple que el supremo de I (que escribiremos $\bigsqcup I$) coincide con la unión de las interpretaciones contenidas en I y, el ínfimo de I (que escribiremos $\bigsqcap I$) coincide con la intersección de las interpretaciones contenidas en I.*

Demostración. Para que $(QInt_{\Sigma}, \subseteq)$ sea un retículo completo, deberá tenerse que existen el máximo y el mínimo de $QInt_{\Sigma}$, y que para todo subconjunto I de $QInt_{\Sigma}$, existen su supremo y su ínfimo.

Veamos primero que podemos encontrar el mínimo y el máximo del conjunto de interpretaciones de $\mathcal{P}$:

1. Para el mínimo, deberemos encontrar una interpretación en $\mathcal{I} \in QInt_{\Sigma}$ que cumpla que para cualquier interpretación $\mathcal{J} \in QInt_{\Sigma}$, $\mathcal{I} \subseteq \mathcal{J}$. Supongamos $\mathcal{I} = \emptyset$. Es fácil ver que se trata de una interpretación pues es subconjunto de QAt_{Σ} y para todo $A \sharp p \in \mathcal{I}$ se tiene trivialmente $A\sigma \sharp q \in \mathcal{I}$ para cualquier $\sigma \in Sust_{\Sigma}$ y $0 < q \leq p$. Además, para toda interpretación $\mathcal{J} \in QInt_{\Sigma}$, $\emptyset \subseteq \mathcal{J}$. Por lo tanto, $min(QInt_{\Sigma}) = \triangle = \emptyset$.

2. En el caso del máximo, nos interesa una interpretación $\mathcal{I} \in QInt_{\Sigma}$ que cumpla que para cualquier interpretación $\mathcal{J} \in QInt_{\Sigma}$, $\mathcal{J} \subseteq \mathcal{I}$. Supongamos $\mathcal{I} = QAt_{\Sigma}$. Es trivial

que QAt_Σ es una interpretación, puesto que $QAt_\Sigma \subseteq QAt_\Sigma$ y por definición contiene a todos los átomos anotados. Además, para cualquier interpretación $\mathcal{J} \in QInt_\Sigma$, se tiene $\mathcal{J} \subseteq QAt_\Sigma$ por la propia definición de interpretación. Así, $max(QInt_\Sigma) = \triangledown = QAt_\Sigma$.

Ahora debe ocurrir que exista el supremo y el ínfimo de todo conjunto de interpretaciones de $QInt_\Sigma$. Y se demuestra que:

(1) $\bigsqcup I = \bigcup\{\mathcal{I} \mid \mathcal{I} \in I\}$ con $I \subseteq QInt_\Sigma$ (que escribiremos $\bigcup I$).

a) $\bigcup I \in QInt_\Sigma$. Para todo átomo anotado $A \sharp p \in \bigcup I$, existe $\mathcal{I} \in I$ tal que $A \sharp p \in \mathcal{I}$. Y por ser $\mathcal{I}$ una interpretación, se tiene que $B \sharp q \in \mathcal{I}$ para todo átomo anotado $B \sharp q$ que fuera de la forma $A\sigma \sharp q$ para alguna sustitución $\sigma \in Sust_\Sigma$ y para algún $q \in (0, p]$; por lo que se tiene también que $B \sharp q \in \bigcup I$.

b) $\mathcal{I} \subseteq \bigcup I$ para toda interpretación $\mathcal{I} \in I$. Trivial por la definición de la unión de una familia de conjuntos.

c) Si $\mathcal{J} \in QInt_\Sigma$ es cota superior de I, entonces $\bigcup I \subseteq \mathcal{J}$. En efecto, si $\mathcal{J}$ es cota superior de $\mathcal{I}$, entonces $\mathcal{I} \subseteq \mathcal{J}$ para toda $\mathcal{I} \in I$. Por definición de unión de una familia de conjuntos, se deduce que $\bigcup I \subseteq \mathcal{J}$.

Por lo tanto se tiene que existe el supremo de I y que coincide con $\bigcup I$.

(2) $\bigsqcap I = \bigcap\{\mathcal{I} \mid \mathcal{I} \in I\}$ con $I \subseteq QInt_\Sigma$ (que escribiremos $\bigcap I$).

a) $\bigcap I \in QInt_\Sigma$. Para todo átomo $A \sharp p \in \bigcap I$, $A \sharp p \in \mathcal{I}$ para toda $\mathcal{I} \in I$. Y por ser $\mathcal{I}$ una interpretación, se tiene que $B \sharp q \in \mathcal{I}$ para todo átomo $B \sharp q$ que fuera de la forma $A\sigma \sharp q$ para alguna sustitución $\sigma \in Sust_\Sigma$ y para algún $q \in (0, p]$; por lo que se tiene también que $B \sharp q \in \bigcap I$.

b) $\bigcap I \subseteq \mathcal{I}$ para toda interpretación $\mathcal{I} \in I$. Trivial por la definición de la intersección de una familia de conjuntos.

c) Si $\mathcal{J} \in QInt_\Sigma$ es cota inferior de I, entonces $\mathcal{J} \subseteq \bigcap I$. En efecto, si $\mathcal{J}$ es cota inferior de I, entonces $\mathcal{J} \subseteq \mathcal{I}$ para toda $\mathcal{I} \in I$. Por definición de intersección de una familia de conjuntos, se deduce que $\mathcal{J} \subseteq \bigcap I$.

Por lo que se tiene que existe el ínfimo de I y que coincide con $\bigcap I$.

Por todo lo anterior, $(QInt_\Sigma, \subseteq)$ es un retículo completo. □

Una vez sabemos que $(QInt_\Sigma, \subseteq)$ sigue siendo un retículo completo, seguiremos con el mismo proceso que seguimos en el capítulo 3 para encontrar el modelo mínimo. Al igual que para la versión cualitativa habíamos definido una transformación $T_\mathcal{P}$, redefinimos ahora dicha transformación para adaptarla a los programas QLP. Tras esta redefinición podremos comprobar que continua siendo monótono y continuo mediante la siguiente proposición.

Proposición 5.2.2. *El operador $T_\mathcal{P} : QInt_\Sigma \to QInt_\Sigma$, según se definió en la sección 5.1, es monótono y continuo.*

Demostración. Demostramos monotonía y continuidad por separado.

- **$T_\mathcal{P}$ es monótono**

  Dadas dos interpretaciones $\mathcal{I}$ y $\mathcal{J}$ tales que $\mathcal{I} \subseteq \mathcal{J}$ se deduce a partir de la definición del operador $T_\mathcal{P}$ que $T_\mathcal{P}(\mathcal{I}) \subseteq T_\mathcal{P}(\mathcal{J})$.

- **$T_\mathcal{P}$ es continuo**

  Sea $\{\mathcal{I}_n\}_{n\in\mathbb{N}}$ una cadena de interpretaciones tal que $\mathcal{I}_n \subseteq \mathcal{I}_{n+1}$ para todo $n \in \mathbb{N}$. Debemos ver que se tiene $\bigcup_{n\in\mathbb{N}} T_\mathcal{P}(\mathcal{I}_n) = T_\mathcal{P}(\bigcup_{n\in\mathbb{N}} \mathcal{I}_n)$. Separamos la igualdad en dos inclusiones:

  (1) $\bigcup_{n\in\mathbb{N}} T_\mathcal{P}(\mathcal{I}_n) \subseteq T_\mathcal{P}(\bigcup_{n\in\mathbb{N}} \mathcal{I}_n)$. Para cada $n_0 \in \mathbb{N}$ se tiene que $\mathcal{I}_{n_0} \subseteq \bigcup_{n\in\mathbb{N}} \mathcal{I}_n$. Por ser $T_\mathcal{P}$ monótono, $T_\mathcal{P}(\mathcal{I}_{n_0}) \subseteq T_\mathcal{P}(\bigcup_{n\in\mathbb{N}} \mathcal{I}_n)$. Luego se tiene que $\bigcup_{n\in\mathbb{N}} T_\mathcal{P}(\mathcal{I}_n) \subseteq T_\mathcal{P}(\bigcup_{n\in\mathbb{N}} \mathcal{I}_n)$.

  (2) $T_\mathcal{P}(\bigcup_{n\in\mathbb{N}} \mathcal{I}_n) \subseteq \bigcup_{n\in\mathbb{N}} T_\mathcal{P}(\mathcal{I}_n)$. Para cada $B\,\sharp\,p \in T_\mathcal{P}(\bigcup_{n\in\mathbb{N}} \mathcal{I}_n)$ existen $\sigma \in Sust_\Sigma$, $(A \leftarrow \alpha - B_1, \ldots, B_m) \in \mathcal{P}$ y pesos $q_1, \ldots, q_n$ tales que B es $A\sigma$, $B_i\sigma\,\sharp\,q_i \in \bigcup_{n\in\mathbb{N}} \mathcal{I}_n$ para todo $1 \leq i \leq m$, y $0 < p \leq \alpha \cdot min\{q_1, \ldots, q_m\}$. Se puede encontrar un $n_0 \in \mathbb{N}$ tal que $\{B_i\sigma\,\sharp\,q_i \mid 1 \leq i \leq n\} \subseteq \mathcal{I}_{n_0}$ ya que $\{\mathcal{I}_n\}_{n\in\mathbb{N}}$ es una cadena. Por ello, $A\sigma\,\sharp\,p \in T_\mathcal{P}(\mathcal{I}_{n_0})$, y $T_\mathcal{P}(\bigcup_{n\in\mathbb{N}} \mathcal{I}_n) \subseteq \bigcup_{n\in\mathbb{N}} T_\mathcal{P}(\mathcal{I}_n)$.

  Y ahora por (3) y (4) se tiene que $\bigcup_{n\in\mathbb{N}} T_\mathcal{P}(\mathcal{I}_n) = T_\mathcal{P}(\bigcup_{n\in\mathbb{N}} \mathcal{I}_n)$. □

Podemos comprobar ahora que los modelos de Herbrand de $\mathcal{P}$ coinciden con los puntos prefijos de $T_\mathcal{P}$.

Proposición 5.2.3. *Dado un programa $\mathcal{P}$, el conjunto de puntos prefijos de $T_\mathcal{P}$ es igual al conjunto de modelos de Herbrand de $\mathcal{P}$. Es decir, $PPF(T_\mathcal{P}) = QHMod(\mathcal{P})$.*

Demostración. Demostramos que, para toda interpretación $\mathcal{I} \in QInt_\Sigma$, se cumple $T_\mathcal{P}(\mathcal{I}) \subseteq \mathcal{I} \Leftrightarrow \mathcal{I} \models \mathcal{P}$.

($\Rightarrow$) Para que se tenga $\mathcal{I} \models \mathcal{P}$ se debe cumplir que para toda cláusula $(A \leftarrow \alpha - B_1, \ldots, B_k) \in \mathcal{P}$ y para toda sustitución σ, si $B_1\sigma\,\sharp\,q_1, \ldots, B_k\sigma\,\sharp\,q_k \in \mathcal{I}$ para algunos $q_1, \ldots, q_k \in (0, 1]$, entonces $A\sigma\,\sharp\,p \in \mathcal{I}$ para cualquier p tal que $0 < p \leq \alpha \cdot min\{q_1, \ldots, q_k\}$. Pero esos $A\sigma\,\sharp\,p$ son, por definición, los elementos de $T_\mathcal{P}(\mathcal{I})$. Y como tenemos que $T_\mathcal{P}(\mathcal{I}) \subseteq \mathcal{I}$ se cumple que $\mathcal{I} \models \mathcal{P}$.

($\Leftarrow$) Dado un átomo anotado $B\,\sharp\,q \in T_\mathcal{P}(\mathcal{I})$, se tiene, por definición de $T_\mathcal{P}$, que existe una cláusula $(A \leftarrow \alpha - B_1, \ldots, B_k) \in \mathcal{P}$, una sustitución $\sigma \in Sust_\Sigma$ y unos pesos $q_1, \ldots, q_k \in (0, 1]$ tales que $B\,\sharp\,q = A\sigma\,\sharp\,p$ con $0 < q \leq p \leq \alpha * min\{q_1, \ldots, q_k\}$ y $B_1\sigma\,\sharp\,q_1, \ldots, B_k\sigma\,\sharp\,q_k \in \mathcal{I}$. Suponiendo $\mathcal{I} \models \mathcal{P}$, de estas condiciones se deduce que $B\,\sharp\,q \in \mathcal{I}$. Luego $T_\mathcal{P}(\mathcal{I}) \subseteq \mathcal{I}$ se deduce de $\mathcal{I} \models \mathcal{P}$.

Las dos implicaciones demuestran que $T_\mathcal{P}(\mathcal{I}) \subseteq \mathcal{I} \Leftrightarrow \mathcal{I} \models \mathcal{P}$. □

Y se deduce también que existe un modelo menor que todos los demás: el modelo mínimo, y que podemos calcularlo empleando el operador $T_\mathcal{P}$. Los dos siguientes corolarios enuncian y prueban formalmente lo anterior:

Corolario 5.2.1. *Dado un programa $\mathcal{P}$ y el operador $T_\mathcal{P}$ según ha sido definido con anterioridad, se tiene que el menor punto prefijo de $T_\mathcal{P}$ es igual a $\bigcap\{\mathcal{I} \in QInt_\Sigma \mid T_\mathcal{P}(\mathcal{I}) \subseteq \mathcal{I}\}$ y coincide con el menor modelo de Herbrand de $\mathcal{P}$. Pondremos $\mu(T_\mathcal{P}) = \mathcal{M}_\mathcal{P}$.*

Demostración. El conjunto $\{\mathcal{I} \in QInt_\Sigma \mid T_\mathcal{P}(\mathcal{I}) \subseteq \mathcal{I}\}$ es el conjunto de los puntos prefijos de $T_\mathcal{P}$. Por la proposición 5.2.1 sabemos que $\bigcap PPF(T_\mathcal{P}) = \prod PPF(T_\mathcal{P})$ y por la proposición 2.3.2 tenemos que $\prod PPF(T_\mathcal{P}) = \mu(T_\mathcal{P})$.

Ahora, por la proposición 5.2.3, $PPF(T_\mathcal{P}) = QHMod(\mathcal{P})$ por lo que $\mu(T_\mathcal{P})$ es también el menor modelo anotado de Herbrand; que denotaremos por $\mathcal{M}_\mathcal{P}$. □

Corolario 5.2.2. *Dado un programa $\mathcal{P}$ y el operador $T_\mathcal{P}$ según ha sido definido con anterioridad, el modelo mínimo de $\mathcal{P}$ es igual a $\bigcup_{n\in\mathbb{N}} T_\mathcal{P} \uparrow^n (\emptyset)$.*

Demostración. Por las proposiciones 5.2.1 y 5.2.2, $(QInt_\Sigma, \subseteq)$ es retículo completo (y en particular, $\omega - CPO$ con fondo) y $T_\mathcal{P}$ es monónota y continua. Por lo tanto, por la proposición 2.3.3 se tiene que $\bigcup_{n\in\mathbb{N}} T_\mathcal{P} \uparrow^n (\emptyset) = \mu(T_\mathcal{P})$. Y por el corolario 5.2.1, $\mu(T_\mathcal{P}) = \mathcal{M}_\mathcal{P}$. □

Otro modo de caracterizar el modelo mínimo $\mathcal{M}_\mathcal{P}$ es como la colección de átomos anotados que se deducen a partir de $\mathcal{P}$ empleando el cálculo lógico QHL que fue definido en la sección anterior. Formalmente:

Proposición 5.2.4. $\mathcal{M}_\mathcal{P} = \{A \sharp p \in QAt_\Sigma \mid \mathcal{P} \vdash_{QHL} A \sharp p\}$.

Demostración. Por el corolario 5.2.2 tenemos que $\mathcal{M}_\mathcal{P} = \bigcup_{n\in\mathbb{N}} T_\mathcal{P} \uparrow^n (\emptyset)$. Y también podemos poner que $\{A \sharp p \in QAt_\Sigma \mid \mathcal{P} \vdash_{QHL} A \sharp p\} = \bigcup_{n\in\mathbb{N}}\{A \sharp p \in QAt_\Sigma \mid \mathcal{P} \vdash^n_{QHL} A \sharp p\}$ donde $\mathcal{P} \vdash^n_{QHL} A \sharp p$ indica que existe una demostración para $A \sharp p$ en QHL con n inferencias. Basta entonces demostrar que las dos siguientes afirmaciones son ciertas:

(1) $\mathcal{P} \vdash^n_{QHL} A \sharp p \Rightarrow \exists m(A \sharp p \in T_\mathcal{P} \uparrow^m (\emptyset))$

(2) $A \sharp p \in T_\mathcal{P} \uparrow^n (\emptyset) \Rightarrow \exists m(\mathcal{P} \vdash^m_{QHL} A \sharp p)$

1. $\mathcal{P} \vdash^n_{QHL} A \sharp p \Rightarrow \exists m(A \sharp p \in T_\mathcal{P} \uparrow^m (\emptyset))$. Razonamos por inducción sobre $n \geq 1$ (ya que $\mathcal{P} \vdash^0_{QHL} A \sharp p$ es imposible):

   - **Caso base ($n = 1$)**
     $\mathcal{P} \vdash^1_{QHL} A \sharp p \Rightarrow$ Existe una demostración en una inferencia QHL para $A \sharp p$. Es decir, $A \sharp p$ es de la forma $H\sigma \sharp p$ para una cláusula $(H \leftarrow \alpha-) \in \mathcal{P}$ y una sustitución $\sigma \in Sust_\Sigma$ con $0 < p \leq \alpha$. Por lo tanto se cumple que $A \sharp p \in T_\mathcal{P} \uparrow^1 (\emptyset)$.
   - **Caso inductivo ($n > 1$)**
     $\mathcal{P} \vdash^n_{QHL} A \sharp p \Rightarrow$ existen una regla $(H \leftarrow \alpha - B_1, \ldots, B_k) \in \mathcal{P}$ y una sustitución $\sigma \in Sust_\Sigma$ tales que A es idéntico a $H\sigma$ y se cumple $0 < p \leq \alpha * min\{q_1, \ldots, q_k\}$ donde los q_i son pesos tales que para cada B_i, existe $m_i \in \mathbb{N}$ tal que $\mathcal{P} \vdash^{m_i}_{QHL} B_i \sharp q_i$, donde además, $\sum_{i=1}^k m_i = n-1$. Entonces, por H.I., existe m suficientemente grande para que $B_1\sigma \sharp q_1, \ldots, B_k\sigma \sharp q_k \in T_\mathcal{P} \uparrow^m (\emptyset)$, por lo que $A \sharp p \equiv H\sigma \sharp p \in T_\mathcal{P} \uparrow^{m+1} (\emptyset)$.

2. $A \sharp p \in T_{\mathcal{P}} \uparrow^n (\emptyset) \Rightarrow \exists m(\mathcal{P} \vdash^m_{QHL} A \sharp p)$. Razonamos por inducción sobre $n \geq 1$ (ya que $T_{\mathcal{P}} \uparrow^0 (\emptyset) = \emptyset$):

   - **Caso base** $(n = 1)$
     $A \sharp p \in T_{\mathcal{P}} \uparrow^1 (\emptyset) = \{A\sigma \sharp p \mid \sigma \in Sust_\Sigma \wedge (A \leftarrow \alpha-) \in \mathcal{P}\}$ con $0 < p \leq \alpha$, por lo que se tiene $\mathcal{P} \vdash^1_{QHL} A \sharp p$.
   - **Caso inductivo** $(n = k + 1, k \geq 1)$
     Como $A \sharp p \equiv H\sigma \sharp p \in T_{\mathcal{P}} \uparrow^{k+1} (\emptyset)$, sabemos que $B_1\sigma \sharp q_1, \ldots, B_k\sigma \sharp q_k \in Tp \uparrow^k (\emptyset)$ siendo $(A \leftarrow \alpha - B_1, \ldots, B_k) \in \mathcal{P}$ y $\sigma \in Sust_\Sigma$ la cláusula y sustitución empleadas para introducir $A \sharp p$ en $T_{\mathcal{P}} \uparrow^{k+1} (\emptyset)$. Por H.I., como $B_1\sigma \sharp q_1, \ldots, B_k\sigma \sharp q_k \in Tp \uparrow^k (\emptyset)$, entonces existen m_i $(1 \leq i \leq k)$ tales que $\mathcal{P} \vdash^{m_i}_{QHL} B_i\sigma \sharp q_i$ $(1 \leq i \leq k)$. Tomamos entonces $m = \sum_{i=1}^k m_i$, se tiene que $\mathcal{P} \vdash^{m+1}_{QHL} H\sigma \sharp p \equiv A \sharp p$. □

Así tenemos que $\bigcup_{n\in\mathbb{N}} T_{\mathcal{P}} \uparrow^n (\emptyset) =_{(1),(2)} \bigcup_{n\in\mathbb{N}} \{A \sharp p \in QAt_\Sigma \mid \mathcal{P} \vdash^n_{QHL} A \sharp p\} = \mathcal{M}_{\mathcal{P}}$.

En el caso de la programación lógica clásica, vimos en la proposición 3.2.5 que el modelo mínimo de un programa $\mathcal{P}$ también se puede caracterizar como el conjunto de todos los átomos que son consecuencia de $\mathcal{P}$ en LPO. El establecer un análogo de LPO con semántica cuantitativa cae fuera de los límites de este trabajo. Sin embargo, sí podemos concluir de los resultados ya demostrados un lema de corrección para QHL, y enunciar en una sola proposición varias caracterizaciones equivalentes de los átomos anotados pertenecientes al modelo mínimo de un programa lógico cuantitativo:

Lema 5.2.1 (Lema de Corrección de QHL)**.**

$$\mathcal{P} \vdash_{QHL} A \sharp p \Rightarrow A \sharp p \in \mathcal{I} \quad \forall \mathcal{I} \in QHMod(\mathcal{P})$$

Demostración. Este resultado se deduce de la proposición 5.2.4 y el corolario 5.2.1. Como ejercicio ilustrativo del funcionamiento de QHL, presentamos aquí otra demostración directa. La suposición $\mathcal{P} \vdash_{QHL} A \sharp p$ significa que existe un árbol de demostración T en QHL de profundidad n. Suponemos fijado cualquier modelo $\mathcal{I} \in QHmod(\mathcal{P})$ y demostramos $A \sharp p \in \mathcal{I}$ por inducción sobre n.

- **Caso base** $(n = 0)$

  T es de profundidad 0, por lo que $A \sharp p$ es instancia de un hecho, o lo que es lo mismo, $A \sharp p = H\sigma \sharp p$ donde $(H \leftarrow \alpha-) \in \mathcal{P}$ y $0 < p \leq \alpha$. Entonces, $A \sharp p \in \mathcal{I}$ por ser $\mathcal{I} \models \mathcal{P}$.

- **Caso inductivo** $(n > 0)$

  Entonces la inferencia QHL cuya conclusión es la raíz de T debe usar una cláusula $(H \leftarrow \alpha - B_1, \ldots, B_k) \in \mathcal{P}$ instanciada por una sustitución $\sigma \in Sust_\Sigma$, de modo que para ciertos pesos q_i se tenga $\mathcal{P} \vdash_{QHL} B_i\sigma \sharp q_i$ $(1 \leq i \leq k)$, con un bosque de profundidad $n - 1$, y además se cumpla $H\sigma = A$ y $0 < p \leq \alpha \cdot min\{q_1, \ldots, q_k\}$. Por H.I. se tiene $B_1\sigma \sharp q_1, \ldots, B_k\sigma \sharp q_k \in \mathcal{I}$; y por ser $\mathcal{I} \models \mathcal{P}$ se concluye $A \sharp p \in \mathcal{I}$. □

Proposición 5.2.5. *Para todo $A \in QAt_\Sigma$ y todo peso $p \in (0, 1]$, las tres afirmaciones siguientes son equivalentes:*

(1) $\mathcal{P} \vdash_{QHL} A \sharp p$

(2) $A \sharp p \in \mathcal{I}$ *para todo* $\mathcal{I} \in QHMod(\mathcal{P})$

(3) $A \sharp p \in \mathcal{M}_{\mathcal{P}}$

Demostración. Demostramos las tres implicaciones necesarias para demostrar la equivalencia:

- (1) $\Rightarrow$ (2): Dada $\mathcal{I} \in QHMod(\mathcal{P})$, si $\mathcal{P} \vdash_{QHL} A \sharp p$, entonces por el Lema de Corrección de QHL (lema 5.2.1), $A \sharp p \in \mathcal{I}$.
- (2) $\Rightarrow$ (3): Trivial porque $\mathcal{M}_{\mathcal{P}} \in QHMod(\mathcal{P})$.
- (3) $\Rightarrow$ (1): Por (3) sabemos que $A \sharp p \in \mathcal{M}_{\mathcal{P}}$, y por la proposición 5.2.4 tenemos que $\mathcal{P} \vdash_{QHL} A \sharp p$. □

5.3. Semántica operacional

Con esta semántica operacional esperamos conseguir un método de resolución de objetivos capaz de calcular las soluciones de cualquier objetivo arbitrario para un programa QLP dado. A diferencia del caso cualitativo, el caso cuantitativo presenta un componente adicional para objetivos y soluciones, y es todo aquello que se hace necesario para tratar con los pesos: el objetivo deberá tener en cuenta una serie de restricciones aritméticas sobre los valores reales del intervalo $(0, 1]$ que cada variable de peso podrá tomar, y las soluciones tendrán que aportar también las sustituciones oportunas para cada variable de peso presente en el objetivo. Dado que el lenguaje se implementará sobre $\mathcal{TOY}$ [5], las restricciones se implementarán utilizando el resolutor de reales aportado por el $\mathcal{TOY}$. La implementación de $\mathcal{TOY}$ utiliza a su vez un resolutor suministrado por el sistema de programación lógica *Sicstus Prolog*. El artículo [35] es una buena referencia general sobre la programación lógica con restricciones sobre el dominio de los números reales.

Sea $\mathcal{P}$ un programa lógico cuantitativo para la signatura Σ. Un objetivo cuantitativo estará formado por un conjunto de átomos anotados $\overline{A}$, una sustitución $\sigma \in Sust_{\Sigma}$ y un conjunto de restricciones que escribiremos Δ y que detallaremos más adelante. Así, un objetivo cualquiera G tendrá la forma $\overline{A} \;[\!]\; \sigma \;[\!]\; \Delta$. Al igual que para el caso cualitativo, existen ciertas condiciones que debe cumplir para ser un objetivo, pero antes de detallarlas veamos qué son las restricciones.

Llamaremos *restricción* bien a una ecuación de la forma (1) o bien a una inecuación de la forma (2), siendo:

(1) $W = \alpha \cdot min\{W_1, \ldots, W_n\}$ donde las $W, W_1, \ldots, W_n \in \mathcal{W}$ y $\alpha \in (0, 1]$. Una restricción de esta forma se llama *restricción definitoria* para W, y se reduce a $W = \alpha$ en el caso $n = 0$.

(2) $\alpha \cdot W \geq m$ donde $W \in \mathcal{W}$ y $\alpha, m \in (0, 1]$. Una restricción de esta forma se llama *restricción umbral* para W.

Dado un conjunto de restricciones Δ definimos su *dominio* como $dom(\Delta) =_{def} \{W \in \mathcal{W} \mid (W = \alpha \cdot min\{W_1, \ldots, W_n\}) \in \Delta\} \cup \{W \in \mathcal{W} \mid (\alpha \cdot W \geq m) \in \Delta\}$. Es decir, el conjunto de variables de peso que aparecen en el lado izquierdo de cualquier restricción de Δ. Y definimos el conjunto de variables de peso de Δ como $war(\Delta) =_{def} dom(\Delta) \cup \{W_i \mid (1 \leq i \leq n), (W = \alpha \cdot min\{W_1, \ldots, W_n\}) \in \Delta\}$, es decir, el conjunto de variables de peso que aparecen en cualquier lugar en cualquiera de las restricciones de Δ.

Nos interesará ahora definir dos características sobre los conjuntos de restricciones que nos serán necesarias para poder construir objetivos, pues como ya se introdujo al inicio de la sección, no cualquier conjunto de restricciones nos servirá. Estas características son *satisfactibilidad* y *admisibilidad*. Comenzaremos por la satisfactibilidad.

Dada una restricción R y una sustitución $\omega \in Sust_W$. Se dice que ω satisface R, y se pone $R\omega$ se satisface cuando se cumple $W\omega = \alpha \cdot min\{W_1\omega, \ldots, W_n\omega\}$ si R es definitoria y $\alpha \cdot W\omega \geq m$ si R es umbral.

Definición 5.3.1 (Satisfactibilidad)**.** *Dado un conjunto de restricciones Δ se dice que es* satisfactible *si y sólo si existe una sustitución $\omega \in Sust_W$ tal que $R\omega$ se satisface para toda restricción $R \in \Delta$. Se pone $\omega \in Sol(\Delta)$ para indicar que ω satisface Δ.*

Definición 5.3.2 (Admisibilidad)**.** *Diremos que un conjunto de restricciones Δ es admisible si cumple:*

(i) *Δ es satisfactible.*

(ii) *Para cada $W \in war(\Delta)$ existe una única restricción para W en Δ. (A consecuencia de esto se cumple también que $war(\Delta) = dom(\Delta)$).*

(iii) *La relación $>_\Delta$ definida por $W >_\Delta W_i \Leftrightarrow_{def} \exists(W = \alpha \cdot min\{W_1, \ldots, W_i, \ldots, W_k\})$ cumple que $>^*_\Delta$ es irreflexiva (no existe W tal que $W >^*_\Delta W$).*

Cuando ocurre que un conjunto de restricciones Δ es admisible y que todas sus restricciones son definitorias se dice que Δ está *resuelto*.

Dado un conjunto de restricciones resuelto Δ, podremos calcular un valor para cada variable de peso del dominio del conjunto, así podemos definir recursivamente la aplicación de Δ a una variable de peso $W \in dom(\Delta)$ como $W\hat{}\Delta =_{def} \alpha \cdot min\{W_1\hat{}\Delta, \ldots, W_k\hat{}\Delta\}$ siendo $W = \alpha \cdot min\{W_1, \ldots, W_k\}$ la restricción definitoria para W en Δ[4]. A partir de esta aplicación de Δ a las variables de peso de su dominio, podemos encontrar una única sustitución $\omega \in Sust_W$ para variables de peso, que llamaremos ω_Δ la sustitución de variables de peso definida por Δ, que cumple $dom(\omega_\Delta) = dom(\Delta)$ y $\omega_\Delta(W) = W\hat{}\Delta$ para toda $W \in dom(\Delta)$.

Ahora que conocemos qué son los conjuntos de restricciones podemos dar la definición formal de objetivo de un programa lógico cuantitativo. Sea $\overline{A}$ una conjunción de átomos anotados, $\sigma \in Sust_\Sigma$ una sustitución y Δ un conjunto de restricciones.

Definición 5.3.3 (Objetivo)**.** *Se llama objetivo G a $\overline{A} \,[\!]\, \sigma \,[\!]\, \Delta$ si y sólo si cumple lo siguiente:*

(i) *σ es idempotente y $dom(\sigma) \cap var(\overline{A}) = \emptyset$.*

[4]En el caso $k = 0$ resulta $W\hat{}\Delta = \alpha$, caso base de la recursión, que se alcanza eventualmente debido a la condición (iii) de la Definición 5.3.2.

(ii) Δ *es admisible.*

(iii) Para toda variable de peso $W \in war(\overline{A})$ *existe una restricción umbral para* W *en* Δ. *Además,* Δ *no contiene ninguna otra restricción umbral adicional.*

Dado un objetivo G, el conjunto de variables de G ($var(G)$) es $var(\overline{A}) \cup dom(\sigma) \cup vran(\sigma)$, y el conjunto de variables de peso de G ($war(G)$) es $war(\overline{A}) \cup war(\Delta)$.

Además, un objetivo G es *inicial* cuando es de la forma $\overline{A} \parallel \epsilon \parallel \Delta$ y se cumple que $dom(\Delta) = war(\overline{A})$ y que toda restricción de Δ es de la forma $W \geq m$. Y es *final* o *resuelto* cuando es de la forma $\parallel \sigma \parallel \Delta$ (también $\sigma \parallel \Delta$) en el que Δ está resuelto.

Veremos ahora cómo son las soluciones a un objetivo G de un programa QLP.

Definición 5.3.4 (Solución)**.** *Se llama* solución *de* G *a una pareja* $S \equiv (\theta, \rho)$ *con* $\theta \in Sust_\Sigma$ *y* $\rho \in Sust_W$ *tal que:*

(i) $\sigma\theta = \theta$

(ii) $\rho \in Sol(\Delta)$

(iii) $\mathcal{P} \vdash^*_{QHL} \overline{A}\hat{}S$, *siendo* $\overline{A}$ *la conjunción de átomos de* G *y* $\overline{A}\hat{}S =_{def} \{A\theta \sharp W\rho \mid A \sharp W \in \overline{A}\}$.

Se escribe $S \in QSol_\mathcal{P}(G)$, *el conjunto de soluciones de* G *en* $\mathcal{P}$.

Notación. Dado un programa $\mathcal{P}$, un objetivo $G \equiv \overline{A} \parallel \sigma \parallel \Delta$ para $\mathcal{P}$, y una solución $S \in QSol_\mathcal{P}(G)$, si se tiene además que $\mathcal{P} \vdash^n_{QHL} \overline{A}\hat{}S$, podemos poner $S \in QSol^n_\mathcal{P}(G)$.

A partir de todo objetivo resuelto podremos obtener una solución asociada a dicho objetivo que, como veremos más adelante, será solución de todos los objetivos de la resolución. Esta solución asociada la calcularemos de la siguiente forma: Dado un objetivo resuelto $G \equiv \sigma \parallel \Delta$, su solución asociada será (σ, μ), siendo $\mu = \omega_\Delta$ la sustitución de variables de peso definida por Δ.

Resolución de objetivos

Llamaremos *resolución* al proceso de cómputo a través del cual se intenta obtener una solución para un objetivo dado. Generalmente, este objetivo será un objetivo inicial para el que buscamos una solución. Sin embargo, generalizamos también aquí esta idea para permitir expresar la búsqueda de soluciones a partir de un objetivo parcialmente resuelto.

Formalmente podemos definir la resolución de un objetivo como sigue.

Definición 5.3.5. *Se llama* resolución *de un objetivo* G *a una serie de pasos de resolución tales que* $G \equiv G_0 \Vdash_{\sigma_1} G_1 \Vdash_{\sigma_2} G_2 \Vdash_{\sigma_3} \cdots \Vdash_{\sigma_n} G_n$ *donde* G_n *es un objetivo resuelto y cada* $G_i \Vdash_{\sigma_{i+1}} G_{i+1}$ *es un paso de resolución. Y llamamos* paso de resolución *a un paso de la forma*

$$\overline{L}, A\sharp W, \overline{R} \parallel \sigma \parallel \alpha \cdot W \geq m, \Delta_R \Vdash_{\sigma_1} (\overline{L}, B_1 \sharp W_1, \ldots, B_k \sharp W_k, \overline{R})\sigma_1 \parallel \sigma\sigma_1 \parallel \Delta_1$$

donde $\Delta_1 = \alpha_1 \cdot \alpha \cdot W_1 \geq m, \ldots, \alpha_1 \cdot \alpha \cdot W_k \geq m, W = \alpha_1 \cdot min\{W_1, \ldots, W_k\}, \Delta_R$ *siendo* $(H \leftarrow \alpha_1 - B_1, \ldots, B_k) \in_{var} \mathcal{P}$, σ_1 *el u.m.g. entre* A *y* H, $0 < m \leq \alpha_1 \cdot \alpha \leq 1$ *y* $W_1, \ldots, W_k \in \mathcal{W}$ *variables de peso nuevas* ($W_1, \ldots, W_k \notin war(G)$).

Notación. Al igual que en el caso cualitativo, utilizaremos en ocasiones la notación $G \Vdash_{\sigma_1, C_1} G'$ para indicar un paso de resolución que emplea la cláusula C_1 y el u.m.g. σ_1.

Obsérvese que la parte de restricciones de un objetivo se entiende como un conjunto, de modo que el orden no importa, y la notación "$\alpha \cdot W \geq m, \Delta_R$" indica un conjunto admisible de restricciones que incluye la restricción umbral $\alpha \cdot W \geq m$ y otras restricciones agrupadas en Δ_R.

A modo de ejemplo se muestran a continuación dos sencillas resoluciones de dos objetivos planteados sobre los ejemplos 5.1.1 y 5.1.2.

Ejemplo 5.3.1. Dado el ejemplo de los Circuitos fiables (ejemplo 5.1.1) buscamos un circuito cuya fiabilidad sea mayor o igual a 0,90. En particular buscaremos el circuito

```
cs(cb("T01",tigre), cb("A07",acme))
```

dado que ya sabemos que es deducible del programa para una certidumbre de 0,90 (véase Ejemplo 5.1.3) u otro más general. El objetivo que nos conduce a lo que buscamos es `fi(C)#W` $[\!] \; W \geq 0{,}90$, y la resolución es la siguiente:

`fi(C)#W` $[\!] \; \epsilon \; [\!] \; W \geq 0{,}90$ $\Vdash_{\{C \mapsto cs(C1,C2)\}, \text{ fi.4}}$

`fi(C1)#W1, fi(C2)#W2` $[\!] \; \{C \mapsto cs(C1, C2)\} \; [\!]$
$W = 0{,}96 \cdot min\{W1, W2\},$
$0{,}96 \cdot W1 \geq 0{,}90,$
$0{,}96 \cdot W2 \geq 0{,}90$ $\Vdash_{\{C1 \mapsto cb(Id1,tigre)\}, \text{ fi.2}}$

`fi(C2)#W2` $[\!] \; \{C \mapsto cs(C1, C2)\} \; [\!]$
$W = 0{,}96 \cdot min\{W1, W2\},$
$W1 = 0{,}95,$
$0{,}96 \cdot W2 \geq 0{,}90$ $\Vdash_{\{C2 \mapsto cb(Id2,acme)\}, \text{ fi.1}}$
$[\!] \; \{C \mapsto cs(cb(Id1, tigre), cb(Id2, acme))\} \; [\!]$
$W = 0{,}96 \cdot min\{W1, W2\},$
$W1 = 0{,}95,$
$W2 = 0{,}99$

Llegados a este punto, tenemos un objetivo resuelto del que podemos obtener su solución asociada: $\{C \mapsto cs(cb(Id1, tigre), cb(Id2, acme))\} \; [\!] \; \{W \mapsto 0{,}912, W1 \mapsto 0{,}95, W2 \mapsto 0{,}99\}$. Como podemos observar, la solución calculada es algo más general de lo que esperábamos, pues deja libres los identificadores, pero el programa no contiene reglas para ellos por lo que la solución calculada es el caso general del circuito buscado. En cuanto a la certidumbre calculada de la variable W, puede verse que se calcula el máximo posible por lo que cumple la restricción inicial de que fuera mayor o igual que 0,90. □

Es importante hacer notar que en estos ejemplos hay una condición que no se está comprobando explícitamente en cada paso de resolución: aquella para la que tienen sentido las restricciones umbrales. Tanto en el ejemplo anterior, como en el que veremos más adelante, las restricciones umbrales terminan cumpliéndose siempre, pero esto se debe a que conocemos de antemano los resultados. En el caso general, y una vez automatizado, muchos de los pasos de resolución intentados no terminarían en éxito precisamente por saber que no va a ser posible encontrar un valor para alguna de las variables de peso para las que tenemos una restricción umbral (en estos casos tendremos que probar con otra regla) que cumpla la restricción. Si nos

fijamos en el segundo paso de resolución dado en el ejemplo anterior (aquél en el que usamos la regla fi.2), podemos ver que estamos resolviendo el átomo `fi(C1)#W1` y que la restricción umbral de $W1$ es $0{,}96 \cdot W1 \geq 0{,}90$ (por lo tanto $\alpha = 0{,}96$ y $m = 0{,}90$). Pues bien, lo que ha de cumplirse para poder dar este paso (a parte de lo ya conocido y que coincide con el caso cualitativo) es $0{,}95 \cdot 0{,}96 \geq 0{,}90$ (viene de $0 < m \leq \alpha_1 \cdot \alpha \leq 1$ en la definición 5.3.5 de *paso de resolución*) en el que $\alpha_1 = 0{,}95$ es el factor de certidumbre asociado a la regla a emplear. Esta condición no utiliza la variable de peso, pero es que esta variable no puede valer más de 1 por lo que si el factor multiplicativo que posee la restricción umbral ($\alpha_1 \cdot \alpha$) ya es inferior a la certidumbre mínima (m), no podrá cumplirse dicha restricción.

Ejemplo 5.3.2. Dado el ejemplo de los Maltratadores (ejemplo 5.1.2) intentamos resolver el objetivo `maltrata(X,"piolin")#W` $[\![W \geq 0{,}5$ para obtener posibles maltratadores de `"piolin"`. En particular, nos interesa encontrar una solución que haga $\{X \mapsto$ `"carlota"`$\}$ porque ya hemos visto gracias al ejemplo 5.1.4 que dicho átomo es deducible a partir de las reglas del programa para una certidumbre menor o igual que 0,525. Veamos como sería la resolución:

`maltrata(X,"piolin")#W` $[\![\epsilon [\![W \geq 0{,}50$ $\Vdash_{\{X \mapsto \texttt{"carlota"}\},\ \text{maltrata.9}}$

`engendran(A,B,"carlota")#W1,`
`maltrata(A,"piolin")#W2,`
`maltrata(B,"piolin")#W3` $[\![\{X \mapsto$ `"carlota"`$\} [\![$
$W = 0{,}70 \cdot min\{W1, W2, W3\}$,
$0{,}70 \cdot W1 \geq 0{,}50$,
$0{,}70 \cdot W2 \geq 0{,}50$,
$0{,}70 \cdot W3 \geq 0{,}50$ $\Vdash_{\{A \mapsto \texttt{"rufo"}\ B \mapsto \texttt{"petronila"}\},\ \text{engendran.1}}$

`maltrata("rufo","piolin")#W2,`
`maltrata("petronila","piolin")#W3` $[\![$
$\{X \mapsto$ `"carlota"`$, A \mapsto$ `"rufo"` $B \mapsto$ `"petronila"`$\} [\![$
$W = 0{,}70 \cdot min\{W1, W2, W3\}$,
$W1 = 1$,
$0{,}70 \cdot W2 \geq 0{,}50$,
$0{,}70 \cdot W3 \geq 0{,}50$ $\Vdash_{\epsilon,\ \text{maltrata.1}}$

`animal("piolin")#W4,`
`maltrata("petronila","piolin")#W3` $[\![$
$\{X \mapsto$ `"carlota"`$, A \mapsto$ `"rufo"` $B \mapsto$ `"petronila"`$\} [\![$
$W = 0{,}70 \cdot min\{W1, W2, W3\}$,
$W1 = 1$,
$W2 = 0{,}90 \cdot min\{W4\}$,
$0{,}70 \cdot W3 \geq 0{,}50$,
$0{,}90 \cdot 0{,}70 \cdot W4 \geq 0{,}50$ $\Vdash_{\epsilon,\ \text{animal.1}}$

`maltrata("petronila","piolin")#W3` $[\![$
$\{X \mapsto$ `"carlota"`$, A \mapsto$ `"rufo"` $B \mapsto$ `"petronila"`$\} [\![$
$W = 0{,}70 \cdot min\{W1, W2, W3\}$,
$W1 = 1$,
$W2 = 0{,}90 \cdot min\{W4\}$,
$0{,}70 \cdot W3 \geq 0{,}50$,
$W4 = 1$ $\Vdash_{\epsilon,\ \text{maltrata.2}}$

$$
\begin{array}{l}
\texttt{animal("piolin")\#W5} \ [\!] \\
\quad \{X \mapsto \texttt{"carlota"}, A \mapsto \texttt{"rufo"}\ B \mapsto \texttt{"petronila"}\} \ [\!] \\
\quad W = 0{,}70 \cdot min\{W1, W2, W3\}, \\
\quad W1 = 1, \\
\quad W2 = 0{,}90 \cdot min\{W4\}, \\
\quad W3 = 0{,}75 \cdot min\{W5\}, \\
\quad W4 = 1, \\
\quad 0{,}75 \cdot 0{,}70 \cdot W5 \geq 0{,}50 \qquad\qquad \Vdash_{\epsilon,\ \text{animal.1}} \\
[\!]\ \{X \mapsto \texttt{"carlota"}, A \mapsto \texttt{"rufo"}\ B \mapsto \texttt{"petronila"}\} \ [\!] \\
\quad W = 0{,}70 \cdot min\{W1, W2, W3\}, \\
\quad W1 = 1, \\
\quad W2 = 0{,}90 \cdot min\{W4\}, \\
\quad W3 = 0{,}75 \cdot min\{W5\}, \\
\quad W4 = 1, \\
\quad W5 = 1
\end{array}
$$

Una vez llegamos a un objetivo resuelto, obtenemos su solución asociada y restringimos a las variables del objetivo. La solución asociada al objetivo resuelto es $\{X \mapsto \texttt{"carlota"}, A \mapsto \texttt{"rufo"}, B \mapsto \texttt{"petronila"}\}$ $[\!]$ $\{W \mapsto 0{,}525; W1 \mapsto 1; W2 \mapsto 0{,}90; W3 \mapsto 0{,}75; W4 \mapsto 1; W5 \mapsto 1\}$ y la solución restringida $\{X \mapsto \texttt{"carlota"}\}$ $[\!]$ $\{W \mapsto 0{,}525\}$. □

Nótese que las soluciones calculadas en ambos ejemplos de resolución proporcionan certidumbres mejores que las demostrados en los ejemplos 5.1.3 y 5.1.4 respectivamente. En general, en el Teorema de Completitud (Teorema 5.3.2) veremos que para cualquier solución dada de un objetivo que esté verificada por una deducción QHL, la resolución puede calcular la misma solución u otra más general (con variables menos particularizadas y/o certidumbre mejor).

Propiedades de la resolución de objetivos

Comprobamos ahora, y antes de continuar con las propiedades de la resolución de objetivos, que el paso de resolución está bien contruido, es decir, que el resultado de un paso de resolución es también un objetivo (o lo que es lo mismo, que se preservan los invariantes de los objetivos).

Proposición 5.3.1. *Dado un objetivo G, se tiene que se preservan los invariantes para todo objetivo G' resultante de dar un paso de resolución a partir de G.*

Demostración. Sea $G \equiv \overline{L}, A \sharp W, \overline{R} \ [\!] \ \sigma \ [\!] \ \alpha \cdot W \geq m, \Delta_R$. Y sea $G' \equiv (\overline{L}, \overline{B}, \overline{R})\sigma' \ [\!] \ \sigma\sigma' \ [\!] \ \Delta'$ donde $\Delta' = \alpha_1\alpha \cdot W_1 \geq m, \ldots, \alpha_1\alpha \cdot W_n \geq m, W = \alpha_1 \cdot min\{W_1, \ldots, W_n\}, \Delta_R$ el resultado de dar un paso de resolución a partir de G con $(H \leftarrow \alpha' - B_1, \ldots, B_n) \in_{var} \mathcal{P}$ y $\sigma' \in Sust_\Sigma$ el u.m.g. entre A y H. Necesitamos comprobar que se cumple (1a) $\sigma\sigma'$ es idempotente, (1b) $dom(\sigma\sigma') \cap var((\overline{L}, \overline{B}, \overline{R})\sigma') = \emptyset$, (2) Δ' es admisible y (3) para toda $W \in war((\overline{L}, \overline{B}, \overline{R})\sigma')$ existe la restricción umbral para W en Δ' y Δ' no contiene ninguna otra restricción umbral. En efecto:

(1a) Igual al apartado (1) de la proposición 3.3.1.

(1b) Análogo al apartado (2) de la proposición 3.3.1.

(2) Sabemos que Δ' es admisible porque es trivialmente satisfactible, existe una única restricción para cada variable de peso (en particular para W porque cambiamos la umbral para W en G por una definitoria para W en G') y la relación $>^*_\Delta$ sigue siendo irreflexiva pues la única nueva restricción definitoria depende únicamente de variables de peso en restricciones umbral.

(3) Dada una variable de peso $W \in war((\overline{L}, \overline{B}, \overline{R})\sigma')$ se tiene uno de los tres siguientes casos:

a) $W \in \overline{L}\sigma'$. Entonces $W \in war(G)$ y la restricción umbral para W que había en Δ sigue estando incluida en Δ'.

b) $W \in \overline{B}\sigma'$. Entonces $W \in \{W_1, \ldots, W_n\}$ y existen en Δ' restricciones umbrales para $W_1, \ldots, W_n$ en Δ' por la definición de paso de resolución.

c) $W \in \overline{R}\sigma'$. Entonces $W \in war(G)$ y la restricción umbral para W que había en Δ sigue estando incluida en Δ'.

y podemos asegurar que no contiene ninguna otra restricción umbral adicional aparte de las de las variables de peso de la conjunción de átomos anotados del objetivo puesto que es trivial comprobar que $war((\overline{L}, \overline{B}, \overline{R})\sigma') = war(\overline{L}, A, \overline{R})\backslash\{W\} \cup \{W_1, \ldots, W_n\}$ y, por la definición de paso de resolución, en Δ' desaparece la restricción umbral para W y aparecen las restricciones umbrales para $W_1, \ldots, W_n$.

Y de (1a), (1b), (2) y (3) se deduce que G' es un objetivo. □

Notación. Cuando tenemos una resolución de la forma $G_0 \Vdash_{\sigma_1} G_1 \Vdash_{\sigma_2} \cdots \Vdash_{\sigma_n} G_n$ podemos escribir $G_0 \Vdash^n_\sigma G_n$ siendo $\sigma = \sigma_1\sigma_2\cdots\sigma_n$. Además podemos escribir $G \Vdash^*_\sigma G'$ si existe $n \in \mathbb{N}$ tal que $G \Vdash^n_\sigma G'$.

Ahora que conocemos cómo es la resolución de objetivos y que sabemos que los pasos de resolución están bien construidos, debemos comprobar que satisface las dos propiedades fundamentales que ya conocemos en el caso de la programación lógica cualitativa, a saber: corrección y completitud.

La corrección nos permitirá demostrar que todas las soluciones que la resolución de objetivos aquí planteada es capaz de calcular para un objetivo dado, son soluciones correctas para dicho objetivo. Este resultado se presenta en el Teorema de Corrección que enunciamos y demostramos a continuación con ayuda del Lema de Corrección. Formalmente:

Lema 5.3.1 (Lema de Corrección)**.** *Dados dos objetivos G_0 y G_1 y una pareja de sustituciones (θ, ρ) tales que $G_0 \Vdash_{\sigma_1} G_1$ y $(\theta, \rho) \in QSol_\mathcal{P}(G_1)$, entonces $(\theta, \rho) \in QSol_\mathcal{P}(G_0)$.*

Demostración. Sea $G_0 \equiv \overline{L}, A \sharp W, \overline{R} \,[\!]\, \sigma_0 \,[\!]\, \Delta_0$ y sea $\sigma_1 \in Sust_\Sigma$ el u.m.g. entre A y H siendo $C_1 \equiv (H \leftarrow \alpha_1 - B_1, \ldots, B_k) \in_{var} \mathcal{P}$ una variante de cláusula de $\mathcal{P}$ sin variables en común con G_0. Entonces $G_1 \equiv (\overline{L},\ B_1 \sharp W_1,\ \ldots,\ B_k \sharp W_k,\ \overline{R})\sigma_1 \,[\!]\, \sigma_0\sigma_1 \,[\!]\, \Delta_1$, siendo además:

$\Delta_0 \equiv \alpha \cdot W \geq m, \Delta_R.$

$\Delta_1 \equiv W = \alpha_1 \cdot min\{W_1, \ldots, W_k\}, \alpha_1 \cdot \alpha \cdot W_1 \geq m, \ldots, \alpha_1 \cdot \alpha \cdot W_k \geq m, \Delta_R.$

Por ser $(\theta, \rho) \in QSol_{\mathcal{P}}(G_1)$ tenemos que (1) $\sigma_0\sigma_1\theta = \theta$, (2) $\rho \in Sol(\Delta_1)$ y (3) $\mathcal{P} \vdash_{QHL} (\overline{L}, B_1 \sharp W_1, \ldots, B_k \sharp W_k, \overline{R})\sigma_1\hat{}(\theta, \rho)$.

Para que (θ, ρ) sea solución de G_0, deberá cumplirse que:

(4) $\sigma_0\theta = \theta$

(5) $\rho \in Sol(\Delta_0)$

(6) $\mathcal{P} \vdash_{QHL} (\overline{L}, A \sharp W, \overline{R})\hat{}(\theta, \rho)$

Veamos que podemos verificar (4), (5) y (6). (4) se tiene por ser idéntico el razonamiento al del punto (3) del Lema de Corrección en el caso cualitativo (Lema 3.3.1). Para ver (5), comprobamos que se satisfacen todas sus restricciones:

(5a) $\rho \in Sol(\alpha \cdot W \geq m)$ significa que $\alpha \cdot W\rho \geq m$. Esto se cumple porque al ser ρ solución de Δ_1, se tiene en particular que $W\rho = \alpha_1 \cdot min\{W_1\rho, \ldots, W_k\rho\}$ y para todo $1 \leq i \leq k$, $\alpha_1 \cdot \alpha \cdot W_i\rho \geq m$.

(5b) $\rho \in Sol(\Delta_R)$. Trivial porque $\Delta_R \subseteq \Delta_1$ y $\rho \in Sol(\Delta_1)$.

por lo que se tiene (5). Y por último, para ver (6), lo separamos en tres casos y vemos cada uno de ellos:

(6a) $\mathcal{P} \vdash_{QHL} \overline{L}\hat{}(\theta, \rho)$. Vemos que $\overline{L}\hat{}(\theta, \rho) = \overline{L}\sigma_1\hat{}(\theta, \rho)$ y aplicamos $\mathcal{P} \vdash_{QHL} \overline{L}\sigma_1\hat{}(\theta, \rho)$, que está garantizado por (3). Sabemos que σ_0 no afecta las variables de G_0, por lo que $\overline{L}\sigma_1\hat{}(\theta, \rho) = \overline{L}\sigma_0\sigma_1\hat{}(\theta, \rho)$. Por definición de la aplicación de una solución a una conjunción de átomos, $\overline{L}\sigma_0\sigma_1\hat{}(\theta, \rho) = \overline{L}\hat{}(\sigma_0\sigma_1\theta, \rho)$, pero $\overline{L}\hat{}(\sigma_0\sigma_1\theta, \rho) =_{(1)} \overline{L}\hat{}(\theta, \rho)$

(6b) $\mathcal{P} \vdash_{QHL} A \sharp W\hat{}(\theta, \rho)$. A partir de (3) podemos ver que $\mathcal{P} \vdash_{QHL} (B_1 \sharp W_1, \ldots, B_k \sharp W_k)\sigma_1\hat{}(\theta, \rho)$. Ahora, en un paso de inferencia QHL, con cláusula C_1 y sustitución $\sigma_1\theta$, y dado que $W\rho = \alpha_1 \cdot min\{W_1\rho, \ldots, W_k\rho\}$, podemos obtener que $\mathcal{P} \vdash_{QHL} H\sigma_1\theta \sharp W\rho$ pero $H\sigma_1\theta = A\theta$ por ser σ_1 el u.m.g. entre A y H, y por definición de la aplicación de una solución a una conjunción de átomos, $A\theta \sharp W\rho = A \sharp W\hat{}(\theta, \rho)$.

(6c) $\mathcal{P} \vdash_{QHL} \overline{R}\hat{}(\theta, \rho)$. Por igual razonamiento que en (6a).

por lo que se tiene (6). □

Y enunciamos ahora el Teorema de Corrección.

Teorema 5.3.1 (Corrección)**.** *Sean G_0 y G dos objetivos tales que $G_0 \Vdash^* G$ y donde G es un objetivo resuelto. Y sea (σ, μ) la solución asociada a G. Entonces (σ, μ) –llamada la solución calculada– es una solución de G_0.*

Demostración. Sean $G_0 \Vdash^n_{\sigma'} G$ donde $G \equiv \sigma \,[\!]\, \Delta$ está resuelto. Probamos por inducción sobre n que la solución asociada (σ, μ) a G es solución de G_0.

- **Caso base**

  En este caso, $n = 0$ y $G_0 = G$ está resuelto y $\mathcal{P} \vdash_{QHL} \overline{A}\hat{}(\sigma, \mu)$ es trivial porque la conjunción de átomos $\overline{A}$ de G es vacía. Además, $\mu \in Sol(\Delta)$ porque $\mu = \omega_\Delta$.

- **Caso inductivo**

  En este caso tenemos que $n > 0$ y $G_0 \Vdash G_1 \Vdash^{n-1} G$. Entonces obtenemos $(\sigma, \mu) \in QSol_{\mathcal{P}}(G_1)$ por hipótesis de inducción, y $(\sigma, \mu) \in QSol_{\mathcal{P}}(G_0)$ debido al Lema de Corrección (Lema 5.3.1). □

En la última parte del capítulo demostraremos que la resolución de objetivos es capaz de calcular todas las soluciones de cualquier objetivo dado. Esto lo garantizará el Lema y Teorema de Completitud que siguen.

Lema 5.3.2 (Lema de Completitud)**.** *Sea $G_0 \equiv \overline{A_0} \parallel \sigma_0 \parallel \Delta_0$ un objetivo no resuelto, y sea $(\theta_0, \rho_0) \in QSol^n_{\mathcal{P}}(G_0)$, que cumplirá $\sigma_0\theta_0 = \theta_0$, $\rho_0 \in Sol(\Delta_0)$ y $\mathcal{P} \vdash^n_{QHL} \overline{A_0}\hat{}(\theta_0, \rho_0)$. Sea V_0 cualquier conjunto finito de variables elegido de manera que $var(G_0) \cup dom(\theta_0) \subseteq V_0$. Para cualquier selección arbitraria de un átomo A de $\overline{A_0}$, existe un paso de resolución*

$$G_0 \equiv \overline{A_0} \parallel \sigma_0 \parallel \Delta_0 \Vdash_{\sigma_1} \overline{A_1} \parallel \sigma_0\sigma_1 \parallel \Delta_1 \equiv G_1$$

usando el átomo elegido, y existe además (θ_1, ρ_1) con las siguientes propiedades:

(a) $\theta_1 = \theta_0[V_0]$

(b) $\sigma_1\theta_1 = \theta_1$

(c) $\sigma_0\sigma_1\theta_1 = \theta_1$

(d) $\rho_1 \geq \rho_0[war(G_0)]$

(e) $\rho_1 \in Sol(\Delta_1)$

(f) $\mathcal{P} \vdash^{n-1}_{QHL} \overline{A_1}\hat{}(\theta_1, \rho_1)$

En particular, (c), (e) y (f) significan que $(\theta_1, \rho_1) \in Sol^{n-1}_{\mathcal{P}}(G_1)$.

Demostración. Suponiendo que $A \sharp W$ sea el átomo seleccionado en G_0, podemos suponer que $G_0 \equiv \overline{L_0}, A \sharp W, \overline{R_0} \parallel \sigma_0 \parallel \alpha \cdot W \geq m, \Delta_R$. Por las hipótesis del lema, podemos suponer además:

(0) $\sigma_0\theta_0 = \theta_0$

(1) $\rho_0 \in Sol(\Delta_0)$

(2) $\mathcal{P} \vdash^{m_1}_{QHL} \overline{L_0}\hat{}(\theta_0, \rho_0)$

(3) $\mathcal{P} \vdash^{m_2}_{QHL} A \sharp W\hat{}(\theta_0, \rho_0)$

(4) $\mathcal{P} \vdash^{m_3}_{QHL} \overline{R_0}\hat{}(\theta_0, \rho_0)$

con $m_1 + m_2 + m_3 = n > 0$.

Por (3), deben existir una cláusula $C_1 \equiv (H \leftarrow \alpha_1 - B_1, \ldots, B_k) \in_{var} \mathcal{P}$ y una sustitución η_0 tales que

(5) $A\theta_0 = H\eta_0$ y $\mathcal{P} \vdash_{QHL}^{m_2-1} B_1\eta_0 \sharp q_1, \ldots, B_k\eta_0 \sharp q_k$ con $q_1, \ldots, q_k \in (0, 1]$ tales que $W\rho_0 \leq \alpha_1 \cdot min\{q_1, \ldots, q_k\}$.

Es posible elegir C_1 y η_0 de modo que se tenga $var(C_1) \cap V_0 = \emptyset$ y $dom(\eta_0) \subseteq var(C_1)$. Con ello, está garantizado que $dom(\eta_0) \cap dom(\theta_0) = \emptyset$, y resulta:

(6) $\theta_1 =_{def} \theta_0 \uplus \eta_0$ es una sustitución bien definida, que cumple: $dom(\theta_1) = dom(\theta_0) \uplus dom(\eta_0)$; $\theta_1 = \theta_0[V_0]$; $\theta_1 = \eta_0[\backslash V_0] \Rightarrow$ (a) del lema.

De (5) y (6) se deduce que θ_1 es un unificador de A y H. Eligiendo σ_1 como u.m.g. canónico (en el sentido de Robinson) de A y H, se tendrá:

(7) $A\sigma_1 = H\sigma_1$ y $\sigma_1\theta_1 = \theta_1 \Rightarrow$ (b) del lema.

Tomando ρ_1 tal que

$$(8)\ W'\rho_1 =_{def} \begin{cases} q_i & \text{si } W' = W_i \text{ para algún } 1 \leq i \leq k \\ \alpha_1 \cdot min\{q_1, \ldots, q_k\} & \text{si } W' = W \\ W'\rho_0 & e.o.c. \end{cases}$$

se tendrá, por (8) y (5), $\rho_1 \geq \rho_0[war(G_0)] \Rightarrow$ (d) del lema.

Efectuando un paso de resolución con σ_1 y C_1 resulta

$$G_0 \equiv \overline{L_0}, A \sharp W, \overline{R_0} \ [] \ \sigma_0 \ [] \ \Delta_0 \Vdash_{\sigma_1, C_1} (\overline{L_0}, B_1 \sharp W_1, \ldots, B_k \sharp W_k, \overline{R_0})\sigma_1 \ [] \ \sigma_0\sigma_1 \ [] \ \Delta_1 \equiv G_1$$

donde $\Delta_0 \equiv \alpha \cdot W \geq m, \Delta_R$ y $\Delta_1 \equiv W = \alpha_1 \cdot min\{W_1, \ldots, W_k\}, \alpha_1 \cdot \alpha \cdot W_1 \geq m, \ldots, \alpha_1 \cdot \alpha \cdot W_k \geq m, \Delta_R$.

Para completar el lema, sólo nos falta demostrar:

(c) $\sigma_0\sigma_1\theta_1 = \theta_1$

(e) $\rho_1 \in Sol(\Delta_1)$

(f) $\mathcal{P} \vdash_{QHL}^{n-1} (\overline{L_0}, B_1 \sharp W_1, \ldots, B_k \sharp W_k, \overline{R_0})\sigma_1\hat{}(\theta_1, \rho_1)$.

- **Demostración de (c)**

  $\sigma_0\underline{\sigma_1\theta_1} =_{(b)} \sigma_0\theta_1 =_{(6)} \sigma_0(\theta_0 \uplus \eta_0) =_{(*)} \sigma_0\theta_0 \uplus \eta_0 =_{(0)} \theta_0 \uplus \eta_0 = \theta_1$.

  El paso $(*)$ se puede justificar porque $vran(\sigma_0) \subseteq V_0$ y $dom(\eta_0) \cap V_0 = \emptyset$.

- **Demostración de (e)**

  Vemos que ρ_1 satisface cada una de las restricciones de Δ_1:

a) $W = \alpha_1 \cdot min\{W_1, \ldots, W_k\}$: se satisface por definición de ρ_1.

b) $\alpha_1 \cdot \alpha \cdot W_i \geq m$ con $1 \leq i \leq k$. Por (1) sabemos que $\alpha \cdot W\rho_0 \geq m$, y de (d) se deduce que $W\rho_1 \geq W\rho_0$. Por lo tanto, $\alpha \cdot W\rho_1 \geq m$. Por otro lado, $W\rho_1 = \alpha_1 \cdot min\{W_1\rho_1, \ldots, W_k\rho_1\}$, por la definición (8) de ρ_1. Luego $\alpha \cdot W\rho_1 \geq m$ implica que se debe cumplir $\alpha_1 \cdot \alpha \cdot W_i\rho_1 \geq m$ para todo $1 \leq i \leq k$.

c) Δ_R. De (7) se deduce que $\rho_1 = \rho_0[war(\Delta_R)]$, y como por (1) $\rho_0 \in Sol(\Delta_R) \Rightarrow \rho_1 \in Sol(\Delta_R)$.

- **Demostración de (f)**

  En primer lugar, observamos que se tiene:

  (9) $\mathcal{P} \vdash^{m_1}_{QHL} \overline{L_0}\hat{}(\theta_1, \rho_1)$ por (2), (6) y (8).

  (10) $\mathcal{P} \vdash^{m_2-1}_{QHL} (B_1 \sharp W_1, \ldots, B_k \sharp W_k)\hat{}(\theta_1, \rho_1)$ por (5), (6) y (8).

  (11) $\mathcal{P} \vdash^{m_3}_{QHL} \overline{R_0}\hat{}(\theta_1, \rho_1)$ por (4), (6) y (8).

  Considerando que $m_1 + (m_2 - 1) + m_3 = n - 1$, (9), (10) y (11) implican que $\mathcal{P} \vdash^{n-1}_{QHL} (\overline{L_0}, B_1 \sharp W_1, \ldots, B_k \sharp W_k, \overline{R_0})\hat{}(\theta_1, \rho_1)$; y esto equivale a (f) debido a (b). □

Teorema 5.3.2 (Completitud)**.** *Dados un objetivo* $G_0 \equiv \overline{A_0} \;[\!]\; \sigma_0 \;[\!]\; \Delta_0$ *y una solución* $(\theta_0, \rho_0) \in QSol_{\mathcal{P}}(G_0)$*, existe un cómputo* $G_0 \Vdash^*_\sigma \sigma_0\sigma \;[\!]\; \Delta$ *que se puede realizar con cualquier estrategia de selección, tal que* $\sigma_0\sigma \;[\!]\; \Delta$ *es final y su solución asociada* $(\sigma_0\sigma, \mu)$ *cumple* $\sigma \preccurlyeq \theta_0[var(G_0)]$ *e incluso* $\sigma_0\sigma \preccurlyeq \theta_0[var(G_0)]$ *y* $\mu \geq \rho_0[war(G_0)]$*, lo que quiere decir que es una solución al menos tan buena como* (θ_0, ρ_0)*.*

Demostración. Como sabemos que G_0 es un objetivo bien formado, sabemos que σ_0 es idempotente y que $\overline{A_0}\sigma_0 = \overline{A_0}$. Ahora por ser $(\theta_0, \rho_0) \in QSol_{\mathcal{P}}(G_0)$, se puede elegir un número $n \in \mathbb{N}$ tal que $(\theta_0, \rho_0) \in QSol^n_{\mathcal{P}}(G_0)$, lo cual quiere decir que se tiene (1) $\sigma_0\theta_0 = \theta_0$, (2) $\rho_0 \in Sol(\Delta_0)$ y (3) $\mathcal{P} \vdash^n_{QHL} \overline{A_0}\hat{}(\theta_0, \rho_0)$.

Se puede elegir también un conjunto finito V_0 de variables que verifique (4) $var(G_0) \cup dom(\theta_0) \subseteq V_0$.

A partir de las condiciones (1), (2), (3) y (4), veremos que podemos demostrar lo siguiente:

(†) Existen un cómputo $G_0 \Vdash^*_\sigma \;[\!]\; \sigma_0\sigma \;[\!]\; \Delta$ (que se puede realizar con cualquier estrategia de selección) que termina en un objetivo final y una sustitución θ que verifica (5) $\theta = \theta_0[V_0]$ (6) $\sigma\theta = \theta$ y (7) $\sigma_0\sigma\theta = \theta$.

De (†) se deduce la tesis del teorema a excepción de (8) $\mu \geq \rho_0[war(G_0)]$, ya que:

(6) $\Rightarrow_{(5)} \sigma\theta = \theta_0[V_0] \Rightarrow \sigma \preccurlyeq \theta_0[var(G_0)]$

(7) $\Rightarrow_{(5)} \sigma_0\sigma\theta = \theta_0[V_0] \Rightarrow \sigma_0\sigma \preccurlyeq \theta_0[var(G_0)]$

La demostración de (†) y (8) es por inducción sobre n:

- **Caso base**

  Si $n = 0$, (2) implica que $\overline{A_0}$ debe ser vacío. Entonces, tomando $\sigma = \epsilon$ y $\theta = \theta_0$ se tiene que $G_0 \Vdash^0_\epsilon \sigma_0 \;[\!]\; \Delta_0$ con un cómputo trivial de 0 pasos; y además:

(5) se reduce a $\theta_0 = \theta_0[V_0]$, que es trivial.

(6) se reduce a $\theta_0 = \theta_0$, que también es trivial.

(7) se reduce a $\sigma_0\theta_0 = \theta_0$ que se cumple por (1).

(8) se tiene porque $\mu = \rho_0\ [war(G_0)]$ se cumple ya que en este caso $war(G_0) = war(\Delta_0)$. Como Δ_0 está resuelto, sólo contiene restricciones definitorias, por lo cual (2) implica que para cualquier $X \in war(\Delta_0)$ se cumple $X\rho_0 = X\hat{}\Delta_0 = X\mu$.

- **Caso inductivo**

Si $n > 0$, (2) implica que $\overline{A_0}$ debe ser no vacío. Seleccionando un átomo $A \sharp W$ de $\overline{A_0}$ con cualquier estrategia, y aplicando el Lema de Completitud (lema 5.3.2), podemos realizar un paso de resolución

(9) $G_0 \equiv \overline{A_0} \,[\!]\, \sigma_0 \,[\!]\, \Delta_0 \Vdash_{\sigma_1} \overline{A_1} \,[\!]\, \sigma_0\sigma_1 \,[\!]\, \Delta_1 \equiv G_1$

de manera que exista una solución $(\theta_1, \rho_1) \in Sol_{\mathcal{P}}^{n-1}(G_1)$ que cumpla 6 condiciones aseguradas por el lema. Aquí las numeramos como (10), (11), (1'), (12), (2') y (3'):

(10) $\theta_1 = \theta_0[V_0]$

(11) $\sigma_1\theta_1 = \theta_1$

(1') $\sigma_0\sigma_1\theta_1 = \theta_1$

(12) $\rho_1 \geq \rho_0[war(G_0)]$

(2') $\Delta_1\rho_1$ se satisface

(3') $\mathcal{P} \vdash_{QHL}^{n-1} \overline{A_1}\hat{}(\theta_1, \rho_1)$

Sea V_1 cualquier conjunto finito de variables elegido de manera que se cumpla:

(4') $V_0 \cup var(G_1) \cup dom(\theta_1) \subseteq V_1$

Las condiciones (1'), (2'), (3') y (4') son del mismo estilo que (1), (2), (3) y (4), pero ahora para $(\theta_1, \rho_1) \in Sol_{\mathcal{P}}^{n-1}(G_1)$. Aplicando la H.I., podemos obtener un cómputo

(13) $G_1 \Vdash_{\sigma'}^{*} \,[\!]\, \sigma_0\sigma_1\sigma' \,[\!]\, \Delta'$

y una sustitución θ que cumpla

(5') $\theta = \theta_1[V_1]$ (6') $\sigma'\theta = \theta$ (7') $\sigma_0\sigma_1\sigma'\theta = \theta$

(8') $\mu' \geq \rho_1[war(G_1)]$ siendo $(\sigma_0\sigma_1\sigma', \mu')$ la solución asociada a $\,[\!]\, \sigma_0\sigma_1\sigma' \,[\!]\, \Delta'$.

De (9) y (13) resulta el cómputo

$$G_0 \equiv \overline{A_0} \,[\!]\, \sigma_0 \,[\!]\, \Delta_0 \Vdash_{\sigma_1} G_1 \equiv \overline{A_1} \,[\!]\, \sigma_0\sigma_1 \,[\!]\, \Delta_1 \Vdash_{\sigma'}^{*} \,[\!]\, \sigma_0\sigma_1\sigma' \,[\!]\, \Delta'$$

Solo nos falta demostrar que (5), (6) y (7) se verifican si se toma la misma θ que cumple (5'), (6') y (7') y $\sigma = \sigma_1\sigma'$; y que (8) también se verifica cuando $\mu = \mu'$. En efecto:

- (5) es consecuencia inmediata de (5'), (4') y (10).
- (6) se razona así: por (4'), se puede suponer $\theta = \theta_1 \uplus \eta'$, con η' tal que $dom(\eta') \cap (V_1) = \emptyset$. Entonces: $\underline{\sigma}\theta = \sigma_1\underline{\sigma'\theta} =_{(6')} \sigma_1\underline{\theta} = \sigma_1(\theta_1 \uplus \eta') =_{(*)} \underline{\sigma_1\theta_1} \uplus \eta' =_{(11)} \theta_1 \uplus \eta' = \theta$.

  El paso $(*)$ se puede justificar porque $vran(\sigma_1) \subseteq V_1$ y $dom(\eta') \cap V_1 = \emptyset$.

- (7) es consecuencia trivial de (7'), ya que $\sigma = \sigma_1\sigma'$.
- (8) es consecuencia de (8') y (12), porque $war(G_0) \subseteq war(G_1)$. □

Capítulo 6

Implementación en $\mathcal{TOY}$

El objetivo de este capítulo es explicar el proceso seguido para conseguir poder cargar programas cuantitativos y poder realizar la resolución de distintos objetivos de prueba sobre dichos programas en el sistema $\mathcal{TOY}$. En la sección 6.2 se explica, por un lado, el proceso teórico de traducción de un programa QLP a otro equivalente en $\mathcal{TOY}$ (esta equivalencia se muestra sólo de forma intuitiva y se comprueban únicamente sus resultados para los casos de prueba propuestos), y por otro, la extensión del propio sistema $\mathcal{TOY}$ para facilitar la compilación, carga y ejecución de los programas cuantitativos. Ya en la sección 6.3 se podrán consultar las ejecuciones de los ejemplos propuestos en el capítulo 5 con el objetivo de comprobar que podemos obtener las mismas soluciones que ya en aquel capítulo se encontraron para los objetivos propuestos en los ejemplos 5.3.1 y 5.3.2.

En el apéndice A puede consultarse tanto el código QLP de los ejemplos empleados en los capítulos 5 y 6 como la traducción de los mismos a $\mathcal{TOY}$. Además se encuentran los listados completos de código correspondientes a la implementación del compilador construido a lo largo de este capítulo y de las modificaciones realizadas sobre el propio sistema $\mathcal{TOY}$ para facilitar su uso al usuario.

6.1. Presentación de $\mathcal{TOY}$

$\mathcal{TOY}$ es un sistema y lenguaje de programación multiparadigma, diseñado para soportar los principales estilos de programación declarativa y una combinación de los mismos.

Los programas en $\mathcal{TOY}$ pueden incluir definiciones de funciones perezosas al estilo de *Haskell* o predicados al estilo de *Prolog*. Tanto las funciones como los predicados deben estar bien tipados con respecto a un sistema polimórfico de tipos. Más aún, los programas $\mathcal{TOY}$ pueden hacer uso de restricciones al estilo de CLP en la definición tanto de funciones como de predicados. Las restricciones soportadas por el sistema incluyen ecuaciones e inecuaciones simbólicas, restricciones aritméticas lineales sobre los números reales y restricciones en dominios finitos. Las computaciones en $\mathcal{TOY}$ resuelven objetivos por medio de estrategias de *narrowing*. Este modelo de computación reúne la evaluación perezosa y la resolución SLD como casos particulares.

Para una descripción más detallada del sistema $\mathcal{TOY}$ así como de las características del

lenguaje en sí mismo, puede acudirse a [5] *$\mathcal{TOY}$, a Multiparadigm Declarative Language. Version 2.2.3.*

6.2. Implementación

La implementación de cualquier lenguaje requiere de la toma de una serie de decisiones que fijen por un lado la idea exacta de lo que se pretende conseguir, y por otro los elementos con los que se cuenta en el proceso. En nuestro caso, la decisión fundamental es la elección de un sistema que funcione como base de nuestra implementación puesto que el desarrollo desde el principio de un sistema completo escapa con creces de los objetivos de un trabajo como éste.

La elección de $\mathcal{TOY}$ para la realización de la implementación de QLP se basa principalmente en tres motivos:

1. El más importante es que es posible trabajar con restricciones reales, lo que facilita la resolución de los conjuntos de restricciones necesarios para el cálculo de la certidumbre en la ejecución de los programas QLP.

2. Al ser un lenguaje declarativo multiparadigma permite la ejecución de programas lógicos y, además, permite la extensión, como trabajo futuro, de la incertidumbre a lenguajes declarativos multiparadigma más expresivos que QLP sin requerir un cambio de sistema.

3. Aunque esta razón es menos importante, podemos tener en cuenta que al estar $\mathcal{TOY}$ desarrollado en este mismo departamento de esta universidad, puede contarse con información y ayuda de primera mano sobre el funcionamiento del sistema a fin de conseguir realizar las modificaciones oportunas de la mejor forma posible para permitir la ejecución de programas QLP.

Una vez conocido el sistema que hará de base para nuestra implementación, debemos analizar sus características para saber qué elementos son los que necesitamos construir. Como ya se ha mencionado, nuestro objetivo es bastante modesto: se pretende poder ejecutar programas QLP y poder resolver objetivos sobre estos programas de una forma más o menos cómoda. Debido a esto, se optará por construir un traductor de programas QLP a programas equivalentes escritos en $\mathcal{TOY}$ que puedan ser ejecutados por dicho sistema, y a extender el propio sistema $\mathcal{TOY}$ mediante una serie de comandos que faciliten las tareas de compilación (traducción) y carga de programas y resolución de objetivos. Dado que $\mathcal{TOY}$ está implementado en Prolog, se realizará el compilador (traductor) en Prolog para poder después ser lanzado desde $\mathcal{TOY}$ a través de comandos.

Los siguientes apartados explican el proceso de implementación seguido.

Traducción de los programas QLP

A partir de lo expuesto en el capítulo 5, sabemos que un programa QLP es un conjunto finito de cláusulas QLP. Estás cláusulas tienen la forma $A \leftarrow \alpha - B_1, \ldots, B_n$ donde $A, B_1, \ldots, B_n$ son átomos y $\alpha \in (0, 1]$ es el factor de certidumbre asociado a la cláusula. Sin

embargo, es muy común permitir al programador insertar comentarios en sus programas para facilitar la comprensión posterior del código y la subdivisión del mismo en secciones. Gracias a la ayuda de Rafael Caballero que nos ha permitido utilizar el Analizador Léxicográfico que construyó para el propio sistema $\mathcal{TOY}$, tenemos esta opción directamente disponible puesto que este analizador ignora los comentarios insertados dentro del código.

Debido a lo anterior, pueden consultarse como ejemplos de programas QLP los ejemplos 5.1.1 y 5.1.2 (de los que pueden encontrarse los listados de código completos en los apartados A.1 y A.2, respectivamente, del apéndice).

Siguiendo con la idea de construir un traductor de programas QLP, necesitaremos primero saber cómo es la traducción de un programa QLP cualquiera. Dado que la traducción de cada una de las cláusulas no depende de las demás, veremos cómo traducir una cláusula genérica QLP a $\mathcal{TOY}$. Partiremos de la cláusula:

$$a(\bar{t}) \leftarrow \alpha_1 - b_1(\overline{r_1}), \ldots, b_k(\overline{r_k})$$

Para poder utilizar la cláusula en una resolución, necesitamos conocer dos elementos, por un lado el mejor factor de certidumbre ya calculado hasta el momento, que se actualizará multiplicándolo por α_1 al aplicar la cláusula que estamos considerando, y la certidumbre mínima que el usuario haya especificado para obtener solución. Además, tendremos que poder devolver la certidumbre con la que hemos conseguido demostrar la cabeza del átomo (que sabemos se calcula multiplicando el factor de certidumbre de la cláusula por el mínimo de las certidumbres con las que se demuestra cada uno de los átomos del cuerpo). Llamaremos F al factor de certidumbre ya calculado, M a la certidumbre mínima especificada por el usuario y W al factor de certidumbre con el que conseguimos demostrar la cabeza. Nótese que F, W y M corresponden, respectivamente, con α, W y m de la restricción umbral $\alpha \cdot W \geq m$ asociada al átomo anotado $A \sharp W$ seleccionado en un paso de resolución (véase Definición 5.3.5). Reescribimos entonces la cabeza añadiendo estas tres variables a su lista de argumentos:

$$a(\bar{t}, F, W, M)$$

Esto mismo ocurrirá con los átomos del cuerpo, cada uno de ellos tendrá a su vez estos tres mismos argumentos además de los que ya tuviera, sabiendo que F debe modificarse dado que conocemos el factor de certidumbre de esta cláusula (α_1), M se mantiene constante, y cada uno de ellos se demostrará para una certidumbre en particular (añadiremos una variable nueva para cada átomo del cuerpo), así tenemos que la cláusula traducida a $\mathcal{TOY}$ tendrá la siguiente forma:

$$\begin{array}{lll} a(\bar{t}, F, W, M) & :- & b_1(\overline{r_1}, \alpha_1 \cdot F, W_1, M), \\ & & \vdots \\ & & b_k(\overline{r_k}, \alpha_1 \cdot F, W_k, M) \end{array}$$

Sin embargo, esta cláusula $\mathcal{TOY}$ aún no está completa. Falta añadir a su cuerpo restricciones que expresen el cálculo del valor de W, representando la certidumbre con la que queda establecida la cabeza. Si sabemos que $W_1, \ldots, W_k$ son las certidumbres para las que hemos demostrado cada uno de los átomos del cuerpo, W tendrá que ser igual a $\alpha_1 \cdot min\{W_1, \ldots, W_k\}$ según nos dice la definición 5.3.5. Entonces, añadiendo esta restricción al final junto con los intervalos para cada una de las variables nuevas del cuerpo obtenemos una versión más completa de la cláusula traducida:

$$
\begin{array}{lll}
a(\bar{t}, F, W, M) & :- & W_1 > 0, W_1 <= 1, b_1(\overline{r_1}, \alpha_1 \cdot F, W_1, M), \\
& & \vdots \\
& & W_k > 0, W_k <= 1, b_k(\overline{r_k}, \alpha_1 \cdot F, W_k, M), \\
& & W == \alpha_1 \cdot min\{W_1, \ldots, W_k\}
\end{array}
$$

Y por último, nos faltaría añadir a su cuerpo una restricción que garantice la condición $0 < m \leq \alpha_1 \cdot \alpha \leq 1$ exigida en la definición 5.3.5 para poder realizar un paso de resolución. Puesto que m y α se corresponden con M y F en nuestra traducción, la restricción que debe añadirse al cuerpo de la cláusula $\mathcal{TOY}$ es $\alpha_1 \cdot F \geq m$, y la traducción completa de la cláusula QLP queda así:

$$
\begin{array}{lll}
a(\bar{t}, F, W, M) & :- & \alpha_1 \cdot F >= M, \\
& & W_1 > 0, W_1 <= 1, b_1(\overline{r_1}, \alpha_1 \cdot F, W_1, M), \\
& & \vdots \\
& & W_k > 0, W_k <= 1, b_k(\overline{r_k}, \alpha_1 \cdot F, W_k, M), \\
& & W == \alpha_1 \cdot min\{W_1, \ldots, W_k\}
\end{array}
$$

Intuitivamente la aplicación de la cláusula traducida implementa un paso de resolución realizado con la cláusula QLP original. La restricción umbral $\alpha \cdot W \geq m$ antes del paso de resolución se corresponde con la restricción $F \cdot W \geq M$ antes de aplicar la cláusula traducida, y las restricciones umbral $\alpha_1 \cdot \alpha \cdot W_i \geq m$ $(1 \leq i \leq k)$ después del paso de resolución se corresponden con restricciones $\alpha_1 \cdot F \cdot W_i \geq M$ que serán gestionadas por los átomos del cuerpo de la cláusula traducida asociados a los átomos del cuerpo de la cláusula original.

Implementación del traductor de programas QLP

La explicación detallada de cada uno de los predicados del prototipo de implementación resultaría demasiado larga y pesada dado que una vez conocido el procedimiento de traducción es fácil ver que el código fuente implementa directamente este proceso puesto que no realiza tareas adicionales como la gestión de errores u otros elementos que, por otro lado, resultarían estrictamente necesarias en cualquier implementación final de un lenguaje de programación.

El proceso de traducción es el siguiente: A partir de un archivo con el programa fuente (con extensión `.qlp`) obtenemos mediante el analizador lexicográfico la lista de *tokens* que componen el programa. A partir de esta lista de *tokens*, el analizador sintáctico realizará la comprobación de que se trata de un programa QLP válido al mismo tiempo que construirá una representación intermedia del programa que será, posteriormente, utilizada por el traductor (entiéndase aquí generador de código) para generar el código en $\mathcal{TOY}$ que dará lugar al programa traducido. Veamos un ejemplo de este proceso:

Ejemplo 6.2.1. Partiremos de un sencillo programa QLP como el siguiente:

```
p <-0.70- q
q <-0.80-
```

Llamando al analizador lexicográfico con el programa anterior, obtenemos la siguiente lista de unidades léxicas (*tokens* en inglés):

```
[sep("{",1),id(p,1),sep("<-",1),float(0.8,1),op(-,1),id(q,1),sep("}",2),
```

```
sep("{",2),id(q,2),sep("<-",2),float(0.7,2),op(-,2),sep("}",2)]
```

En la lista anterior, podemos ver que se han detectado dos secciones de código (las secciones sólo afectan a los "grupos" de código que se traducen cada vez y se separan mediante llaves)[1]; y en cada una de estas secciones podemos ver que se han identificado los elementos que componen cada una de las reglas.

Ahora, si le damos esta lista de *tokens* (suponiendo otra vez que lo hagamos de un única pasada) al analizador sintáctico, comprobaría que en efecto es un programa QLP válido y nos daría la representación intermedia siguiente:

```
[cl(at(p,[]),f(0.8),[at(q,[])])]
[cl(at(q,[]),f(0.7),[])]
```

Al igual que con la lista de *tokens* tenemos dos secciones, cada una de ellas con la representación de una cláusula cuya cabeza es un átomo con símbolo `p` en la primera y `q` en la segunda. Los factores de certidumbre de cada cláusula son 0,8 y 0,7, respectivamente, y el cuerpo de la primera es otro átomo con símbolo `q`.

Ahora conseguiríamos llamando al traductor con la representación intermedia anterior que generara el código traducido final para este programa, que sería (lo siguiente es el resultado de ejecutar el compilador para este programa en particular variando la indentación):

```
% Traducido por QLPcompil (v0.0.1)
min1 [] = 1
min1 [X|Xs] = min2 X (min1 Xs)
min2 W1 W2 = if W1 <= W2 then W1 else W2

p(V_5256, V_5257, V_5258) :- V_5256*0.8>=V_5258,
    V_6229>0, V_6229<=1.0, q(V_5256*0.8, V_6229, V_5258),
    V_5257 == 0.8 * min1 [V_6229]
q(V_10905, V_10906, V_10907) :- V_10905*0.7>=V_10907,
    V_10906 == 0.7 * min1 []
```

y una versión más legible (cambiando los nombres de las variables nuevas):

```
% Traducido por QLPcompil (v0.0.1)
min1 [] = 1
min1 [X|Xs] = min2 X (min1 Xs)
min2 W1 W2 = if W1 <= W2 then W1 else W2

p(F,W,M) :- F*0.8>=M,
    W1>0, W1<=1.0, q(F*0.8, W1, M),
    W == 0.8 * min1 [W1]
q(F,W,M) :- F*0.7>=M,
    W == 0.7 * min1 []
```

[1] En realidad esto sería un proceso iterativo para cada sección de código, es decir, el compilador realizaría dos procesos de traducción, uno para la primera sección y otro para la segunda.

que es un programa $\mathcal{TOY}$ válido. □

En la traducción anterior, puede observarse que se han añadido dos funciones al código del programa traducido que no aparecían en el programa original. Estas funciones son `min1` y `min2`. La primera calcula el mínimo de los elementos de un conjunto dado, y la segunda calcula el mínimo de dos elementos. La unión de estas dos funciones son las que nos aportan la funcionalidad requerida por la función *min* que aparece en las restricciones definitorias para calcular el valor de W (razón por la que aparece también en la traducción de una cláusula QLP). El hecho de que las hayamos denominado `min1` y `min2` es debido a que `min` es una palabra reservada en $\mathcal{TOY}$.

Otro aspecto a tener en cuenta en dicha traducción es que $\mathcal{TOY}$ necesita las declaraciones de tipos de todas las constructoras que aparecen en los programas para poder ejecutarlos, a diferencia de QLP que no posee tipos. En el ejemplo anterior no hay problema porque no tenemos constructoras pero cuando las hay es necesario añadir estas declaraciones al programa traducido antes a su ejecución en $\mathcal{TOY}$. En principio, estas declaraciones de tipos deberían aparecer en el código original en QLP y el compilador debería mantenerlas intactas en la traducción, pero en el estado actual del prototipo de implementación esto no se hace. Más adelante, en la sección 6.3 veremos como actuar más detalladamente.

La implementación final del traductor de programas QLP se dividió en tres archivos[2], en los que cada uno de ellos realiza la siguiente función:

`[qlp_token.pl]` Implementa el analizador lexicográfico a partir del analizador lexicográfico de $\mathcal{TOY}$. Aporta las funciones `readInitialSection/4` y `readNextSection/4` que permiten leer el archivo fuente por secciones y obtener la lista de *tokens* que componen el programa original.

`[qlp_gramma_dcg.pl]` Gramática (en formato DCG de Prolog) de los programas QLP que nos permite leer la lista de *tokens* obtenida por el analizar lexicográfico y obtener la representación intermedia del programa original que empleará el traductor para escribir el código $\mathcal{TOY}$.

`[qlp_compil.pl]` Implementa el compilador (traductor) propiamente dicho. El predicado `translateToToy/2` implementa el traductor. Recibe la representación intermedia del programa y el archivo de destino y escribe la traducción a $\mathcal{TOY}$. Los predicados `qlpcompil/1` y `qlpcompil/2` implementan el bucle de compilado que mediante llamadas sucesivas al analizador lexicográfico, al sintáctico (la gramática DCG) y al traductor obtiene el programa QLP traducido en un archivo `.toy`.

Traducción de los objetivos QLP

La traducción de los objetivos QLP difiere ligeramente de la de los programas. Hasta ahora un objetivo cualquiera se podía escribir como $G \equiv \overline{A} \parallel \sigma \parallel \Delta$. Ahora, como nos interesa resolver objetivos, entenderemos que nos interesa resolver a partir de objetivos iniciales, por lo tanto $\sigma = \epsilon$ y $G \equiv \overline{A} \parallel \Delta$. Así, tendremos que un objetivo genérico tendrá la forma:

$$A_1 \sharp W_1, \ldots, A_n \sharp W_n \parallel W_1 \geq m_1, \ldots, W_n \geq m_n$$

[2]Puede consultarse el código completo de estos archivos en el apartado A.3 del apéndice.

que representaremos en QLP de la siguiente forma:

```
A1#W1, ..., An#Wn | W1 >= m1, ..., Wn >= mn
```

Según las restricciones aportadas, F deberá valer 1 para todos los átomos anotados del objetivo, M será lo que cada restricción indique para su respectivo átomo anotado, y la variable de peso es la que aparece como anotación tras cada uno de los átomos. Así, el objetivo anterior quedaría traducido a $\mathcal{TOY}$ como sigue:

```
A1(1.0,W1,m1), ..., An(1.0,Wn,mn)
```

La notación anterior debe entenderse como el incremento, en los tres argumentos indicados en cada átomo, de la lista de argumentos de cada uno de ellos. Así, si `A1` es `a(t1, ..., tn)`, entonces `A1(1.0, W1, m1)` será `a(t1, ..., tn, 1.0, W1, m1)`.

Extensión del sistema $\mathcal{TOY}$

Se realizó el prototipo de implementación de QLP con la idea de permitir realizar pruebas de manera sencilla con distintos programas QLP. El resultado de la traducción anterior es un programa $\mathcal{TOY}$ que deberá ser ejecutado para poder intentar la resolución los objetivos que nos interese probar para nuestro programa. Sin embargo, la implementación está hecha en *Sicstus Prolog* por lo que no resultaría muy cómodo si dejásemos aquí el trabajo de implementación. Para evitar por completo el uso externo de *Sicstus Prolog* se crearon además una serie de comandos para poder cargar el compilador, compilar y lanzar objetivos QLP desde $\mathcal{TOY}$ de forma que no sea más complicado realizar las pruebas con un programa QLP que con uno escrito directamente en $\mathcal{TOY}$. Los comandos son relativamente sencillos de construir una vez tenemos todo lo anterior implementado. Veamos cómo son y qué hace cada uno de ellos[3]:

- El primero de ellos que nos permitirá cargar el compilador es `/qlpload`. Dado que es indispensable tener cargado el compilador para poder ejecutar cualquier otro comando relacionado con QLP en $\mathcal{TOY}$, deberemos asegurarnos de ejecutar el comando anterior antes de ningún otro.
- La compilación propiamente dicha la realiza `/qlptotoy`. Este comando requiere un argumento, el archivo fuente sin la extensión (que deberá ser `.qlp`). Así, si quisiésemos compilar el programa de los Maltratadores (Ejemplo 5.1.2) tendremos que hacer `/qlptotoy(maltratadores)`. Posteriormente tendremos en un archivo con el mismo nombre pero con extensión `.toy` (`maltratadores.toy` en este caso) el programa traducido que podremos cargar con cualquiera de los comandos normales de $\mathcal{TOY}$.
- Por último, tenemos el comando `/qlpgoal` que nos permite lanzar un objetivo mediante la sintaxis de QLP. Si suponemos cargado el programa de los Maltratadores, podríamos intentar obtener la solución que obtuvimos en el ejemplo 5.3.2 mediante el objetivo `/qlpgoal(maltrata(X,"piolin")#W | W >= 0.5)`.

[3]Puede consultarse el código fuente de la implementación de estos comandos en la sección A.4 del apéndice.

La explicación de la implementación de estos tres comandos no resulta inmediata sin tener un conocimiento de la implementación y estructura general de $\mathcal{TOY}$ por lo que no resulta interesante su explicación para este trabajo.

6.3. Casos de prueba

Para concluir este capítulo vamos a presentar unas pruebas de ejecución de los programas propuestos como ejemplo en el capítulo 5.

Para los dos casos siguientes de prueba supondremos que tenemos los programas escritos en dos archivos: `circuitos.qlp` y `maltratadores.qlp` (estos archivos contienen únicamente el código tal y como aparece en el apéndice).

Caso de prueba 6.3.1 (Circuitos fiables)**.** Comenzamos iniciando el sistema $\mathcal{TOY}$, cargando el resolutor de restricciones reales y el compilador QLP:

```
Toy 2.2.3: A Constraint Functional Logic Language.
(c) 1997-2006


Type "/h" for help.

Toy> /cflpr

Real Domain Constraints library loaded.
Toy(R)> /qlpload

QLP Compiler loaded.
Toy(R)>
```

Ahora que tenemos preparado el sistema, compilamos el programa de los Circuitos fiables mediante el comando `/qlptotoy`:

```
Toy(R)> /qlptotoy(c:/qlp/circuitos)
(QLPcompil) Translating to TOY.......
(QLPcompil) Translation complete.
Toy(R)>
```

Llegados a este punto y dado que nuestro lenguaje no tiene tipos y $\mathcal{TOY}$ sí, si intentásemos cargar el programa en $\mathcal{TOY}$ nos daría un error de tipos. Para remediar este problema es necesario que el sistema $\mathcal{TOY}$ pueda determinar a qué tipo de datos corresponde cada una de las constructoras que aparezcan en la traducción del programa QLP original (en este caso, `cb`, `cs`, `cp`, `acme`, `tigre` y `panda`). Para ello, es necesario a su vez que el programa $\mathcal{TOY}$ resultante de la traducción contenga las oportunas declaraciones de tipos de datos. Se espera que el usuario incluya estas declaraciones en el texto del programa QLP fuente y el compilador las mantenga en la traducción, pero dado que el prototipo no realiza esta labor

por el momento, deberán mantenerse comentadas para que sean ignoradas por el compilador y copiadas manualmente (retirando ahora sí los comentarios) por el usuario para incorporarlas al texto traducido a $\mathcal{TOY}$ del programa antes de procesar éste.

En el caso de este ejemplo, hay que incorporar manualmente al archivo `circuitos.toy` las declaraciones de tipos:

```
type id = [char]
data tm = acme | tigre | panda
data cr = cb(id,tm) | cs(cr,cr) | cp(cr,cr)
```

Acto seguido se puede compilar y cargar el programa en $\mathcal{TOY}$:

```
Toy(R)> /run(c:/qlp/circuitos)
Checking syntax
.....................................................Done.
Checking dependencies...Done.
Checking types...Done
Generating code.....................Done
PROCESS COMPLETE
Toy(R)>
```

Y ahora estamos en condiciones de probar a lanzar el objetivo y ver qué circuitos podemos construir con una fiabilidad mayor o igual a 0,90:

```
Toy(R)> /qlpgoal(fi(C)#W | W >= 0.90)
    { C -> (cb (_A, acme)),
        W -> 0.99 }
    Elapsed time: 0 ms.
sol.1, more solutions (y/n/d/a) [y]?
```

Si pedimos más soluciones podemos llegar a la que habíamos obtenido en el ejemplo 5.3.1:

```
    { C -> (cb (_A, tigre)),
        W -> 0.95 }
    Elapsed time: 0 ms.
sol.2, more solutions (y/n/d/a) [y]?
    { C -> (cb (_A, panda)),
        W -> 0.9 }
    Elapsed time: 0 ms.
sol.3, more solutions (y/n/d/a) [y]?
    { C -> (cs ((cb (_A, acme)), (cb (_B, acme)))),
        W -> 0.9503999999999999 }
    Elapsed time: 0 ms.
sol.4, more solutions (y/n/d/a) [y]?
    { C -> (cs ((cb (_A, acme)), (cb (_B, tigre)))),
        W -> 0.9119999999999999 }
```

```
    Elapsed time: 15 ms.
sol.5, more solutions (y/n/d/a) [y]?
    { C -> (cs ((cb (_A, acme)), (cs ((cb (_B, acme)), (cb (_C, acme))))))),
        W -> 0.9123839999999999 }
    Elapsed time: 0 ms.
sol.6, more solutions (y/n/d/a) [y]?
    { C -> (cs ((cb (_A, tigre)), (cb (_B, acme)))),
        W -> 0.9119999999999999 }
    Elapsed time: 0 ms.
sol.7, more solutions (y/n/d/a) [y]?
```

y en la solución 7 aparece la que habíamos obtenido con anterioridad. Si continuamos pidiendo más soluciones podremos ver que llegaremos a encontrar hasta 13 soluciones posibles. Si después de esto se piden más soluciones, $\mathcal{TOY}$ falla finitamente al intentar calcularlas y responde `no` al usuario. □

Caso de prueba 6.3.2 (Maltratadores)**.** Al igual que en caso anterior, debemos cargar el sistema $\mathcal{TOY}$, el resolutor de restricciones reales y el compilador QLP. Una vez hecho, compilamos el programa de los maltratadores con el comando:

```
Toy(R)> /qlptotoy(c:/qlp/maltratadores)
```

y lo cargamos directamente puesto que esta vez no nos dará problemas con los tipos (debido a que el tipo `[char]` de las cadenas de caracteres está predefinido en $\mathcal{TOY}$):

```
Toy(R)> /run(c:/qlp/maltratadores)
```

Una vez tenemos el programa cargado estamos listos para buscar todos aquellos individuos que pueden maltratan a `"piolin"` con una certidumbre igual o mayor que 0,50. Lanzamos el objetivo `maltrata(X,"piolin")#W | W >= 0.50` y vemos las soluciones que obtenemos:

```
Toy(R)> /qlpgoal(maltrata(X,"piolin")#W | W >= 0.50)
    { X ->  "rufo",
        W -> 0.9 }
    Elapsed time: 0 ms.
sol.1, more solutions (y/n/d/a) [y]?
    { X ->  "petronila",
        W -> 0.75 }
    Elapsed time: 0 ms.
sol.2, more solutions (y/n/d/a) [y]?
    { X ->  "carlota",
        W -> 0.5249999999999999 }
    Elapsed time: 0 ms.
sol.3, more solutions (y/n/d/a) [y]?
    no
    Elapsed time: 0 ms.
Toy(R)>
```

Hemos obtenido tres posibles soluciones: `"rufo"` con una certidumbre de 0,90, `"petronila"` con 0,75 y `"carlota"` con 0,525. Y es precisamente la tercera la que coincide con el ejemplo de resolución que vimos en el capítulo 5 (Ejemplo 5.3.2). □

Capítulo 7

Conclusiones y Trabajo Futuro

Conclusiones

El objetivo fundamental de este trabajo ha sido revisar y actualizar los principales trabajos existentes sobre la Programación Lógica Cuantitativa con especial énfasis en el trabajo de *van Emden* [61]. De lo realizado podemos decir que se ha mejorado el trabajo de *van Emden* en los dos aspectos fundamentales que se enumeraban al comienzo del capítulo 5, además de haber conseguido realizar un prototipo de implementación basado en el sistema $\mathcal{TOY}$. El prototipo utiliza una técnica de traducción de cláusulas que se corresponde fielmente con el método de resolución presentado en el capítulo 5, según se ha explicado de manera informal en la sección 6.2. El uso del prototipo permite ejecutar programas QLP y resolver objetivos sobre los mismos.

La comparación con los trabajos de *Subrahmanian* es algo más compleja puesto que existen diferencias más sustanciales con respecto al enfoque con el que se introduce la incertidumbre. Sus trabajos [57, 58] emplean un orden, contenido en el intervalo $[0, 1]$, cuyo mínimo se sitúa en el 0,5 para el que dice no saber nada acerca de la veracidad, o falsedad, del átomo en cuestión y crece hacia el 1 cuando aumenta la certeza de que el átomo se verifica, y hacia el 0 cuando la certeza es de que el átomo no se verifica. Su método de resolución también permite resolver objetivos abiertos, sin embargo, demuestra corrección y completitud sólo para aquellos programas bien construidos, o en su terminología, *nice QLPs*.

Trabajo futuro

Para finalizar este trabajo, podemos resumir algunas de las líneas de trabajo que quedan abiertas y que permitirían un futuro desarrollo, a corto o medio plazo, para la continuación de este trabajo.

- Realizar una generalización de la propuesta realizada en este trabajo de forma que podamos sustituir el conjunto de valores de certidumbre por otro cualquiera con forma de retículo. Esto permitiría incluir los trabajos de *Subrahmanian* y otros más que puedan ser englobados en la Programación Lógica Cuantitativa como casos particulares de dicha

generalización.

- En una línea más práctica, realizar una demostración de la equivalencia entre la implementación propuesta en este trabajo y la resolución también aquí propuesta. Y, una vez alcanzado el punto anterior, extender dicha implementación para abarcar diferentes casos particulares que incluya la generalización.
- Por último, y para un plazo mayor, realizar una aproximación de los enfoques con los que se ha tratado la incertidumbre en los lenguajes de programación lógica a los lenguajes declarativos multiparadigma con especial énfasis en la posibilidad de extender $\mathcal{TOY}$ con las ideas aportadas por tal aproximación.

Apéndice A

Listados

El presente apéndice muestra los listados completos de los ejemplos empleados en el capítulo 5 (así como de sus traducciones a código $\mathcal{TOY}$) y de los elementos construidos para la implementación en el capítulo 6 de modo que sirvan como referencia tanto para las ejecuciones realizadas en los casos de prueba (Sección 6.3) como para poder comprobar el código completo del traductor implementado y de las modificaciones introducidas en $\mathcal{TOY}$.

Nótese que las traducciones finales realizadas por el traductor difieren de las aquí mostradas en los nombres de las variables nuevas (que no aparecen en el código original), en la ausencia de comentarios y en el espaciado (aparece una cláusula completa por línea y todas las cláusulas de manera consecutiva).

A.1. Ejemplo: Circuitos fiables

```
% -------------------------------------------------
% CIRCUITOS FIABLES
% -------------------------------------------------

% Declaraciones de tipos y datos (copiar a la traducción)
%type id = [char]
%data tm = acme | tigre | panda
%data cr = cb(id,tm) | cs(cr,cr) | cp(cr,cr)

% Componentes básicas
fi(cb(Id,acme)) <-0.99-
fi(cb(Id,tigre)) <-0.95-
fi(cb(Id,panda)) <-0.90-

% Composición secuencial y paralela
fi(cs(C1,C2)) <-0.96- fi(C1), fi(C2)
fi(cp(C1,C2)) <-0.94- fi(C1), fi(C2)
fi(cp(C1,C2)) <-0.50- fi(C1)
fi(cp(C1,C2)) <-0.50- fi(C2)
```

```
% Objetivo de prueba
% ?- fi(C)#W | W >= 0.90
```

Para una explicación del código acúdase al ejemplo 5.1.1. Y para distintos ejemplos en la resolución de objetivos a la sección 6.3.

La traducción a $\mathcal{TOY}$ del código anterior es la que sigue:

```
% ------------------------------------------------
% CIRCUITOS FIABLES (TRADUCCIÓN TOY)
% ------------------------------------------------

% Función min1
minl [W] = W
minl [W1,W2|Ws] = min2 W1 (minl [W2|Ws])
min2 W1 W2 = if W1 <= W2 then W1 else W2

% Declaraciones de tipos y datos
type id = [char]
data tm = acme | tigre | panda
data cr = cb(id,tm) | cs(cr,cr) | cp(cr,cr)

% Componentes básicas
fi(cb(Id,acme),F,W,M) :- F*0.99 >= M, W == 0.99
fi(cb(Id,tigre),F,W,M) :- F*0.95 >= M, W == 0.95
fi(cb(Id,panda),F,W,M) :- F*0.90 >= M, W == 0.90

% Composición secuencial y paralela
fi(cs(C1,C2),F,W,M) :- F*0.96 >= M,
    W1 > 0, W1 <= 1.0, fi(C1,F*0.96,W1,M),
    W2 > 0, W2 <= 1.0, fi(C2,F*0.96,W2,M),
    W == 0.99 * minl [W1,W2]
fi(cp(C1,C2),F,W,M) :- F*0.94 >= M,
    W1 > 0, W1 <= 1.0, fi(C1,F*0.94,W1,M),
    W2 > 0, W2 <= 1.0, fi(C2,F*0.94,W2,M),
    W == 0.99 * minl [W1,W2]
fi(cp(C1,C2),F,W,M) :- F*0.50 >= M,
    W1 > 0, W1 <= 1.0, fi(C1,F*0.50,W1,M),
    W == 0.5 * minl [W1]
fi(cp(C1,C2),F,W,M) :- F*0.50 >= M,
    W2 > 0, W2 <= 1.0, fi(C2,F*0.50,W2,M),
    W == 0.5 * minl [W2]

% Objetivo de prueba
% ?- fi(C,1,W,0.90)
```

A.2. Ejemplo: Maltratadores

```
% ------------------------------------------------
% MALTRATADORES
% ------------------------------------------------

% Entes (humanos, animales y vegetales)
humano("rufo") <-
humano("petronila") <-
humano("romeo") <-
humano("julieta") <-
humano(Z) <- engendran(X,Y,Z), humano(X), humano(Y)

engendran("rufo","petronila","carlota") <-
engendran("romeo","julieta","pedro") <-
engendran("pedro","carlota","crispin") <-

animal("piolin") <-
animal("silvestre") <-
vegetal("alcornoque") <-
vegetal("lechuguino") <-

% Predicado "maltrata"
maltrata("rufo",X)      <-0.90- animal(X)
maltrata("petronila",X) <-0.75- animal(X)
maltrata("romeo",X)     <-0.40- animal(X)
maltrata("julieta",X)   <-0.30- animal(X)

maltrata("rufo",X)      <- vegetal(X)
maltrata("petronila",X) <-0.95- vegetal(X)
maltrata("romeo",X)     <-0.75- vegetal(X)
maltrata("julieta",X)   <-0.60- vegetal(X)

maltrata(X,Y) <-0.70- engendran(A,B,X), maltrata(A,Y), maltrata(B,Y)

% Predicado "cruel"
cruel(X) <-0.90- humano(X), maltrata(X,Y), animal(Y)
cruel(X) <-0.40- humano(X), maltrata(X,Y), vegetal(Y)

% Objetivo de prueba
% ?- maltrata(X,"piolin")#W | W >= 0.50
```

Para una explicación del código acúdase al ejemplo 5.1.2. Y para distintos ejemplos en la resolución de objetivos a la sección 6.3.

La traducción a $\mathcal{TOY}$ del código anterior es la que sigue:

```
% ------------------------------------------------
% MALTRATADORES (TRADUCCIÓN TOY)
% ------------------------------------------------

% Función min1
minl [W] = W
minl [W1,W2|Ws] = min2 W1 (minl [W2|Ws])
min2 W1 W2 = if W1 <= W2 then W1 else W2

% Entes (humanos, animales y vegetales)
humano("rufo",F,W,M) :- F*1.0>=M, W == 1.0 * min1 []
humano("petronila",F,W,M) :- F*1.0>=M, W == 1.0 * min1 []
humano("romeo",F,W,M) :-  F*1.0>=M, W == 1.0 * min1 []
humano("julieta",F,W,M) :- F*1.0>=M, W == 1.0 * min1 []
humano(Z,F,W,M) :-
    F*1.0>=M,
    W1>0, W1<=1.0,engendran(X,Y,Z,F*1.0, W1, M),
    W2>0, W2<=1.0,humano(X,F*1.0, W2, M),
    W3>0, W3<=1.0,humano(Y,F*1.0, W3, M),
    W == 1.0 * min1 [W1,W2,W3]

engendran("rufo","petronila","carlota",F,W,M) :-
    F*1.0>=M, W == 1.0 * min1 []
engendran("romeo","julieta","pedro",F,W,M) :-
    F*1.0>=M, W == 1.0 * min1 []
engendran("pedro","carlota","crispin",F,W,M) :-
    F*1.0>=M, W == 1.0 * min1 []

animal("piolin",F,W,M) :- F*1.0>=M, W == 1.0 * min1 []
animal("silvestre",F,W,M) :- F*1.0>=M, W == 1.0 * min1 []
vegetal("alcornoque",F,W,M) :- F*1.0>=M, W == 1.0 * min1 []
vegetal("lechuguino",F,W,M) :- F*1.0>=M, W == 1.0 * min1 []

% Predicado "maltrata"
maltrata("rufo",X,F,W,M) :- F*0.9>=M,
    W1>0, W1<=1.0,animal(X,F*0.9,W1,M),
    W == 0.9 * min1 [V_113158]
maltrata("petronila",X,F,W,M) :- F*0.75>=M,
    W1>0, W1<=1.0,animal(X,F*0.75,W1,M),
    W == 0.75 * min1 [W1]
maltrata("romeo",X,F,W,M) :- F*0.4>=M,
    W1>0, W1<=1.0,animal(X,F*0.4,W1,M),
    W == 0.4 * min1 [W1]
maltrata("julieta",X,F,W,M) :- F*0.3>=M,
    W1>0, W1<=1.0,animal(X,F*0.3,W1,M),
    W == 0.3 * min1 [W1]
```

```
maltrata("rufo",X,F,W,M) :- F*1.0>=M,
    W1>0, W1<=1.0,vegetal(X,F*1.0,W1,M),
    W == 1.0 * min1 [W1]
maltrata("petronila",X,F,W,M) :- F*0.95>=M,
    W1>0, W1<=1.0,vegetal(X,F*0.95,W1,M),
    W == 0.95 * min1 [W1]
maltrata("romeo",X,F,W,M) :- F*0.75>=M,
    W1>0, W1<=1.0,vegetal(X,F*0.75,W1,M),
    W == 0.75 * min1 [W1]
maltrata("julieta",X,F,W,M) :- F*0.6>=M,
    W1>0, W1<=1.0,vegetal(X,F*0.6,W1,M),
    W == 0.6 * min1 [W1]

maltrata(X,Y,F,W,M) :- F*0.7>=M,
    W1>0, W1<=1.0,engendran(A,B,X,F*0.7,W1,M),
    W2>0, W2<=1.0,maltrata(A,Y,F*0.7,W2,M),
    W3>0, W3<=1.0,maltrata(B,Y,F*0.7,W3,M),
    W == 0.7 * min1 [W1,W2,W3]

% Predicado "cruel"
cruel(X,F,W,M) :- F*0.9>=M,
    W1>0, W1<=1.0,humano(X,F*0.9,W1,M),
    W2>0, W2<=1.0,maltrata(X,Y,F*0.9,W2,M),
    W3>0, W3<=1.0,animal(Y,F*0.9,W3,M),
    W == 0.9 * min1 [W1,W2,W3]
cruel(X,F,W,M) :- F*0.4>=M,
    W1>0, W1<=1.0,humano(X,F*0.4,W1,M),
    W2>0, W2<=1.0,maltrata(X,Y,F*0.4,W2,M),
    W3>0, W3<=1.0,vegetal(Y,F*0.4,W3,M),
    W == 0.4 * min1 [W1,W2,W3]

% Objetivo de prueba
% ?- maltrata(X,"piolin",1.0,W,0.50)
```

A.3. Traductor de programas y objetivos QLP

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% qlp_token.pl
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
/*
    - Autor: Carlos Alb. Romero Díaz
    - Fecha: 11 septiembre 2007
    - Descripción: Realiza en análisis léxico para los programas QLP.
*/
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:- module(qlptoken, [readInitialSection/4, readNextSection/4, readAll/2]).

:- load_files(token,[if(changed),imports([searchToken/3, readSection/6])]).
:- load_files(errortoy,[if(changed),imports([treatLexiconError/1])]).
:- load_files(tools,[if(changed),imports([endFile/2,append/3])]).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% ----------------------------------------------------------------------------
% Lee la primera sección del archivo.
% readInitialSection(+FileIn,-H,-ListOfTokens,-Future)
%   - FileIn       : Nombre del archivo para leer.
%   - H            : Handler del archivo desde donde leer.
%   - ListOfTokens : Lista de tokens de la sección leída.
%   - Future       : Carácter inicial de la siguiente sección.

readInitialSection(FileIn,H,ListOfTokens,Future) :-
    open(FileIn,read,H),
    beginSection(H,FirstChar,Error),
    readNextSection(H,FirstChar,ListOfTokens,Future).

% beginSection(+H,-FirstChar,-Error)
% Devuelve los datos que identifican el primer caracter de la primera seccion.
% si hay error, lo trata y devuelve Error = true. e.o.c. Error=false

beginSection(H,(Line,Pos,Ch),Error) :- !,
        searchToken(H,(1,0,32),(Line,Pos,Ch)),
        treatLexiconError(Error).

% ----------------------------------------------------------------------------
% Lee las siguientes secciones del archivo (una cada vez que se llama).
% readNextSection(+H,+Present,-ListOfTokens,-Future)
%   - H            : Handler del archivo desde donde leer.
%   - Present      : Primer carácter de la sección (y leído anteriormente).
%   - ListOfTokens : Lista de tokens de la sección leída.
%   - Future       : Carácter inicial de la siguiente sección.

readNextSection(H,Present,ListOfTokens,Future) :-
    readSection(H,Present,Future,[],32,ListOfTokens).
```

```
% ------------------------------------------------------------------------
% Lee el resultado de pasar todas las secciones por el analizador lexicográfico.
% readAll(+FileIn,-LO)
%   - FileIn : Nombre completo del fichero a leer.
%   - LO     : Lista de tokens leídos.

readAll(FileIn,LO) :- readInitialSection(FileIn,H,L,Future),
                      continueReadAll(H,Future,L,LO),
                close(H).

continueReadAll(H,(Line,Pos,Ch),LI,LI) :-
    endFile(H,Ch),!.

continueReadAll(H,Present,LI,LO) :-
    readNextSection(H,Present,LN,Future),
    append(LI,LN,LN2),
    continueReadAll(H,Future,LN2,LO).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% qlp_gramma_dcg.pl
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
/*
    - Autor: Carlos Alb. Romero Díaz
    - Fecha: 9 septiembre 2007
    - Descripción: Gramática del lenguaje QLP según se definió en el proyecto de
      máster "Programación Lógica Cuantitativa y su Implementación en TOY".
*/
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:- module(qlpgrammadcg, [seccion/3,qlpobjetivo/3]).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% ------------------------------------------------------------------------------
%     S E C C I O N E S
% ------------------------------------------------------------------------------
seccion(Rep) --> separador("{"),topdecls(Rep).

% Posibles declaraciones de 'primer nivel'. Al menos una
topdecls([One|Rest]) --> !, topdecl(One), resttopdecls(Rest).
resttopdecls([One|R]) --> separador(";"),!,topdecl(One),resttopdecls(R).
resttopdecls([]) --> !,separador("}").

% Las declaraciones de primer nivel pueden ser o bien declaraciones de tipos o
% datos (que copiaremos tal cual al programa TOY final) o cláusulas QLP que
% tendremos que traducir a TOY.

topdecl(S) --> qlpclausula(S).

% ------------------------------------------------------------------------------
%     C L Á U S U L A S   Q L P
% ------------------------------------------------------------------------------
%       1. Cláusulas del tipo A <-f- B1,..,Bn (con n >= 0).
%       2. Cláusulas del tipo A <- B1,..,Bn (con n >= 0 y f=1.0).
% ------------------------------------------------------------------------------

qlpclausula(cl(H,F,C)) --> qlpatomo(H), !, qlpclcuerpo(F,C).
qlpclcuerpo(F,C) --> separador("<-"),real(F),!,operador(-),qlpclatomos(C).
qlpclcuerpo(f(1.0),C) --> separador("<-"),!,qlpclatomos(C).
qlpclatomos(C) --> qlpatomos(C).
qlpclatomos([]) --> !,[].

qlpatomos([A|AS])--> qlpatomo(A),!,qlprestatomos(AS).
qlprestatomos([A|AS]) --> separador(","),!,qlpatomo(A),qlprestatomos(AS).
qlprestatomos([]) --> !,[].

qlpatomo(at(H,C)) --> identf(H),separador("("),!,qlpterminos(C),separador(")").
qlpatomo(at(H,[])) --> identf(H).

qlpterminos([T|TS]) --> qlptermino(T),!,qlprestterminos(TS).
qlprestterminos([T|TS]) --> separador(","),!,qlptermino(T),qlprestterminos(TS).
```

```
qlprestterminos([]) --> !,[].

qlptermino(T) --> qlpfuncion(T) | variable(T) | string(T).

qlpfuncion(fn(H,C)) --> identf(H),separador("("),!,qlpterminos(C),separador(")").
qlpfuncion(fn(H,[])) --> identf(H).

% ------------------------------------------------------------------------------
%     O B J E T I V O S   Q L P
% ------------------------------------------------------------------------------
%       1. A1#W1, ..., An#Wn | W1 >= m1, ..., Wn >= mn
%          Obtendremos una lista de átomos anotados y otra de pares (variable,
%          valor). En la traducción tomaremos el primer elemento de cada lista,
%          después los segundos, etc.
% ------------------------------------------------------------------------------

qlpobjetivo(goal(LA,LC)) --> separador("{"),!,
                    qlpaatomos(LA),separador("|"),
                    qlpconstraints(LC),separador("}").

qlpaatomos([AA|AAS]) --> qlpaatomo(AA),!,qlprestaatomos(AAS).
qlprestaatomos([AA|AAS]) --> separador(","),!,qlpaatomo(AA),qlprestaatomos(AAS).
qlprestaatomos([]) --> !,[].

qlpaatomo(aa(H,C,W)) --> qlpatomo(at(H,C)),operador(#),variable(W).

qlpconstraints([C|CS]) --> qlpconstraint(C),!,qlprestconstraints(CS).
qlprestconstraints([C|CS]) --> separador(","),!,qlpconstraints([C|CS]).
qlprestconstraints([]) --> !,[].

qlpconstraint(c(W,F)) --> variable(W),operador(>=),real(F).

% ------------------------------------------------------------------------------
%     C A D E N A S   D E   T E X T O
% ------------------------------------------------------------------------------

string(str(S)) --> separador("["),reststring(S).
reststring([]) --> separador("]").
reststring([C]) --> char(C),separador("]"),!.
reststring([C|CS]) --> char(C),separador(","),!,reststring(CS).

% ------------------------------------------------------------------------------
%     E L E M E N T O S   S I N T Á C T I C O S
% ------------------------------------------------------------------------------

real(f(F)) --> [float(F,_)].
identf(I) --> [id(I,_)].
separador(S) --> [sep(S,_)].
operador(O) --> [op(O,_)].
variable(v(V)) --> [var(V,_)].
char(C) --> [chr(C,_)].
reservada(R) --> [res(R,_)].
tipo_res(T) --> [tres(T,_)].
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% qlp_compil.pl
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
/*
    - Autor: Carlos Alb. Romero Díaz
    - Fecha: 11 septiembre 2007
    - Descripción: Compilador de programas QLP. Realiza la traducción de un
      programa en QLP a TOY para que pueda ser posteriormente empleado como
      un programa TOY.
*/
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

:- module(qlpcompil, [qlpcompil/1, qlpcompil/2, qlpcompilg/1, qlpcompilg/2]).

:- load_files(qlp_token,[if(changed),imports([readInitialSection/4,
    readNextSection/4, readAll/2])]).
:- load_files(qlp_gramma_dcg,[if(changed), imports([seccion/3,qlpobjetivo/3])]).
:- load_files(tools,[if(changed),imports([endFile/2])]).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% ----------------------------------------------------------------------------
%     C O M P I L A D O R
% ----------------------------------------------------------------------------
%  qlpcompil(+FileIn)
%    - FileIn  : Nombre del fichero a compilar SIN extensión ".qlp".
%
%  qlpcompil(+FileIn, +FileOut)
%    - FileIn  : Nombre completo del fichero a compilar (con extensión).
%    - FileOut : Nombre completo del fichero destino (con extensión).
% ----------------------------------------------------------------------------

% Compilamos a un fichero .toy con igual nombre.
qlpcompil(File) :- !,
    atom_concat(File,'.qlp',FileQLP),
    atom_concat(File,'.toy',FileTOY),
    qlpcompil(FileQLP,FileTOY).

% Compilamos al fichero destino indicado.
qlpcompil(FileIn, FileOut) :- !,
    open(FileOut, write, Hout),                          % Escritura
    addHeader(Hout),                                     % Ponemos header
    write('(QLPcompil) Translating to TOY'),
    readInitialSection(FileIn,Hin,ListOfTokens,CharOut), % Lexicográfico
    qlpsyntax(ListOfTokens,S),                           % Sintáctico
    translateToTOY(Hout,S),                              % Traducción
    write('.'),
    compilation(Hin, Hout, CharOut),                     % Bucle compilación
    close(Hin),
    close(Hout),
    nl,write('(QLPcompil) Translation complete.').
```

```
% ------------------------------------------------------------------------
% Bucle de compilación (termina cuando termine el archivo).
% compilation(+Hin, +Hout, +CharIn)
%  - Hin    : Handler del archivo para lectura.
%  - Hout   : Handler del archivo para escritura.
%  - CharIn : Carácter desde el que comenzar.

compilation(Hin, Hout, (Line, Pos, Ch)) :-
    endFile(Hin, Ch), !.

compilation(Hin, Hout, CharIn) :-
    qlplexicon(Hin,CharIn,TokensOfSection,CharOut),      % Lexicográfico
    qlpsyntax(TokensOfSection,S),                        % Sintáctico
    translateToTOY(Hout,S),                              % Traducción
    write('.'),
    compilation(Hin, Hout, CharOut).                     % Bucle compilación

% ------------------------------------------------------------------------
% Cabecera de los archivos traducidos.
% addHeader(+Hout)
%  - Hout : Handler del archivo para escritura.

addHeader(Hout) :-
    write(Hout,'% Traducido por QLPcompil (v0.0.1)'),nl(Hout),
    write(Hout,'min1 [] = 1'),nl(Hout),
    write(Hout,'min1 [X|Xs] = min2 X (min1 Xs)'),nl(Hout),
    write(Hout,'min2 W1 W2 = if W1 <= W2 then W1 else W2'),nl(Hout),
    nl(Hout).

% ------------------------------------------------------------------------
%     C O M P I L A D O R   D E   O B J E T I V O S
% ------------------------------------------------------------------------
%  qlpcompilg(+FileIn)
%    - FileIn  : Nombre del fichero a compilar SIN extensión ".qlp".
%
%  qlpcompilg(+FileIn, +FileOut)
%    - FileIn  : Nombre completo del fichero a compilar (con extensión).
%    - FileOut : Nombre completo del fichero destino (con extensión).
% ------------------------------------------------------------------------

% Compilamos a un fichero .tmp.toy con igual nombre.
qlpcompilg(File) :- !,
    atom_concat(File,'.tmp.qlp',FileQLP),
    atom_concat(File,'.tmp.toy',FileTOY),
    qlpcompilg(FileQLP,FileTOY).

% Compilamos al fichero destino indicado.
qlpcompilg(FileIn, FileOut) :- !,
    qlpalllexicon(FileIn,ListOfTokens),                  % Lexicográfico
    qlpgoalsyntax(ListOfTokens,goal(As,Cs)),             % Sintáctico
    open(FileOut, write, Hout),                          % Escritura
    translateToTOYg(Hout,As,Cs),                         % Traducción
    close(Hout).
```

```
% ------------------------------------------------------------------------
%     L E X I C O G R Á F I C O
% ------------------------------------------------------------------------
%  qlplexicon(+H,+CharIn,-TokensOfSection,-CharOut)
%    - H               : Handle del fichero de lectura.
%    - CharIn          : Primer carácter de la sección (y que se obtuvo
%                        anteriormente a partir de CharOut).
%    - TokensOfSection : Lista de Tokens de la sección.
%    - CharOut         : Primer carácter de la sección siguiente (y que se
%                        empleará una llamada posterior a esta regla).
%  qlpalllexicon(+File,-ListOfTokens)
%    - File            : Nombre del archivo del que obtener la lista de tokens.
%    - ListOfTokens    : Lista de tokens leída.
% ------------------------------------------------------------------------

% Le pedimos la siguiente sección.
qlplexicon(H, CharIn, TokensOfSection, CharOut) :- !,
    readNextSection(H, CharIn, TokensOfSection, CharOut).

% Le pedimos todas las secciones.
qlpalllexicon(File,ListOfTokens) :- !,readAll(File,ListOfTokens).

% ------------------------------------------------------------------------
%     S I N T Á C T I C O
% ------------------------------------------------------------------------
%  qlpsyntax(+ListOfTokens,-S)
%    - ListOfTokens : Lista de tokens que componen una sección del programa.
%    - S            : Representación sintáctica del programa para poder
%                     trabajar con él.
%
%  qlpgoalsyntax(+ListOfTokens,-S)
%    - ListOfTokens : Lista de tokens que componen un objetivo.
%    - S            : Representación sintáctica del objetivo.
% ------------------------------------------------------------------------

qlpsyntax(ListOfTokens,S) :- seccion(S,ListOfTokens,[]).

qlpgoalsyntax(ListOfTokens,S) :- qlpobjetivo(S,ListOfTokens,[]).

% ------------------------------------------------------------------------
%     T R A D U C C T O R   A   T O Y
% ------------------------------------------------------------------------
% La traducción se realiza por secciones del código original.
%
%  translateToTOY(+H,+Section)
%    - H       : Handler del archivo donde escribir la traducción.
%    - Section : Sección obtenida del analizador sintáctico para traducir a TOY.
% ------------------------------------------------------------------------

translateToTOY(H,[cl(Head,Factor,Body)|R]) :- !,
    translateQLPClause(H,Head,Factor,Body),
    nl(H),
```

```
    translateToTOY(H,R).
translateToTOY(H,[]) :- !.

% ------------------------------------------------------------------------
% Traduce una cláusula QLP a TOY.
% translateQLPClause(+H,+Head,+Factor,+Body)
%   - H      : Handler del archivo donde escribir.
%   - Head   : Cabeza de la cláusula (de la forma "at(...)").
%   - Factor : Factor de certidumbre asociado a la cláusula (0.0 < F <= 1.0).
%   - Body   : Cuerpo de la cláusula (de la forma "[at(...), at(...), ...]").

translateQLPClause(H,Head,f(Factor),Body) :-
    translateQLPAtom(H,Head,Hf,Hw,Hm), write(H,' :- '),
    writeQLPVar(H,Hf),write(H,'*'),
    write(H,Factor),write(H,'>='), writeQLPVar(H,Hm),
    translateQLPBody(H,Body,Factor,Hf,Hw,Hm,ListWs),
    writeQLPVar(H,Hw), write(H,' == '), write(H,Factor),
    write(H,' * min1 '), write(H,'['),translateQLPWarList(H,ListWs).

% ------------------------------------------------------------------------
% Traduce un Átomo QLP a TOY.
% translateQLPAtom(+H,+Atom,F,W,M[,+VarF])
%   - H      : Handler del archivo donde escribir.
%   - Atom   : Átomo QLP (de la forma "at(...)").
%   - F      : Certidumbre calculada hasta el momento (permite la poda).
%   - W      : Certidumbre máxima para la que se demuestra el átomo.
%   - M      : Valor mínimo de certidumbre (permite la poda).
%   - (VarF) : Variación en el factor (esta variante de translateQLPAtom se
%              emplea sólo cuando el Átomo pertenece al cuerpo para escribir
%              correctamente "Hf * Factor" por "F").

% Átomo es símbolo de proposición
translateQLPAtom(H,at(Head,[]),F,W,M) :-
    !,write(H,Head),
    write(H,'('),
    writeQLPVar(H,F),write(H,', '),
    writeQLPVar(H,W),write(H,', '),
    writeQLPVar(H,M),
    write(H,')').

translateQLPAtom(H,at(Head,Body),F,W,M) :-
    write(H,Head),
    write(H,'('),
    translateQLPTerms(H,Body), write(H,','),
    writeQLPVar(H,F),write(H,', '),
    writeQLPVar(H,W),write(H,', '),
    writeQLPVar(H,M),
    write(H,')').

% Átomo es símbolo de proposición (con variación)
translateQLPAtom(H,at(Head,[]),F,W,M,VarF) :-
    !,write(H,Head),
    write(H,'('),
```

```
    writeQLPVar(H,F),write(H,'*'),write(H,VarF),write(H,', '),
    writeQLPVar(H,W),write(H,', '),
    writeQLPVar(H,M),
    write(H,')').

translateQLPAtom(H,at(Head,Body),F,W,M,VarF) :-
    write(H,Head),
    write(H,'('),
    translateQLPTerms(H,Body), write(H,','),
    writeQLPVar(H,F),write(H,'*'),write(H,VarF),write(H,', '),
    writeQLPVar(H,W),write(H,', '),
    writeQLPVar(H,M),
    write(H,')').

% ------------------------------------------------------------------------
% Traduce el cuerpo de una cláusula QLP a TOY.
% translateQLPBody(+H,+ListAtoms,+Factor,Hf,Hw,Hm,-ListWs)
%   - H         : Handler del archivo donde escribir.
%   - ListAtoms : Cabeza de la cláusula (de la forma "at(...)").
%   - Factor    : Factor de certidumbre asociado a la cláusula (0.0 < F <= 1.0).
%   - Hf        : Certidumbre calculada de la cabeza de la cláusula.
%   - Hw        : Certidumbre de la cabeza de la cláusula.
%   - Hm        : Valor mínimo que interesa para demostrar el objetivo.
%   - ListWs    : Lista de los Ws del cuerpo.

translateQLPBody(H,[Atom|R],Factor,Hf,Hw,Hm,[W|WR]) :-
    write(H,','),
    writeQLPVar(H,W),write(H,'>0, '),writeQLPVar(H,W),write(H,'<=1.0,'),
    translateQLPAtom(H,Atom,Hf,W,Hm,Factor),
    translateQLPBody(H,R,Factor,Hf,Hw,Hm,WR).

translateQLPBody(H,[],Factor,Hf,Hw,Hm,[]) :-
    write(H,',').

% ------------------------------------------------------------------------
% Traduce una conjunción de términos QLP a TOY.
% translateQLPTerms(+H,+ListOfTerms)
%   - H           : Handler del archivo donde escribir.
%   - ListOfTerms : Lista de términos QLP.

translateQLPTerms(H,[T1,T2|R]) :-
    translateQLPTerm(H,T1),
    write(H,','),
    translateQLPTerms(H,[T2|R]).

translateQLPTerms(H,[T]) :-
    translateQLPTerm(H,T).

translateQLPTerms(H,[]).

% Cadena de texto (string)
translateQLPTerm(H,str(S)) :- writeQLPString(H,S).
```

```
% Variable
translateQLPTerm(H,v(Var)) :- write(H,Var).

% Función sin parámetros
translateQLPTerm(H,fn(Functor,[])) :- !, write(H,Functor).

% Función con parámetros
translateQLPTerm(H,fn(Functor,Body)) :-
    write(H,Functor),
    write(H,'('),
    translateQLPTerms(H,Body),
    write(H,')').

% ------------------------------------------------------------------------------
% Escribe una lista de variables W
% translateQLPWarList(+H,+ListWs)

translateQLPWarList(H,[]) :-
    !,write(H,']').

translateQLPWarList(H,[W]) :-
    !,writeQLPVar(H,W),write(H,']').

translateQLPWarList(H,[W1,W2|Ws]) :-
    !,writeQLPVar(H,W1),write(H,','),translateQLPWarList(H,[W2|Ws]).

% ------------------------------------------------------------------------------
% Escribe una variable con nombre desconocido
% writeQLPVar(+H,+Var)

writeQLPVar(H,Var) :- write(H,'V'),write(H,Var).

% ------------------------------------------------------------------------------
% Escribe un string (escribe también las comillas [34])
% writeQLPString(+H,+Str)

writeQLPString(H,Str) :- put_code(H,34),writeQLPChars(H,Str).
writeQLPChars(H,[C|CS]) :- put_code(H,C),writeQLPChars(H,CS).
writeQLPChars(H,[]) :- put_code(H,34).

% ------------------------------------------------------------------------------
%     T R A D U C C T O R   D E   O B J E T I V O S
% ------------------------------------------------------------------------------
% La traducción se realiza por secciones del código original.
%
%  translateToTOYg(+H,+As,+Cs)
%    - H  : Handler del archivo donde escribir la traducción.
%    - As : Lista de átomos anotados del objetivo.
%    - Cs : Lista de restricciones del objetivo.
% ------------------------------------------------------------------------------

translateToTOYg(H,[],Cs) :- !.
translateToTOYg(H,[A],Cs) :- trGoalAAtom(H,A,Cs),!.
```

```
translateToTOYg(H,[A1,A2|R],Cs) :-
    trGoalAAtom(H,A1,Cs),
    write(H,','),
    translateToTOYg(H,[A2|R],Cs).

% Traduce un átomo anotado sin cuerpo de un objetivo.
trGoalAAtom(H,aa(Head,[],v(W)),Cs) :- floatForWar(W,Cs,F),
    write(H,Head),
    write(H,'(1.0,'),
    write(H,W),write(H,','),
    write(H,F),
    write(H,')').

% Traduce un átomo anotado de un objetivo.
trGoalAAtom(H,aa(Head,Body,v(W)),Cs) :- floatForWar(W,Cs,F),
    write(H,Head),
    write(H,'('),
    translateQLPTerms(H,Body), write(H,',1.0,'),
    write(H,W),write(H,','),
    write(H,F),
    write(H,')').

% ------------------------------------------------------------------------
% Escribe el real asociado a una variable en la lista de restricciones. Si la
% variable no aparece en la lista, devolverá 0.0.
% floatForWar(+War,+Constraints,-Float)
%   - W           : Variable para la que queremos obtener su real asociado.
%   - Constraints : Lista de restricciones.
%   - F           : Número real asociado a la variable en War.

floatForWar(W,[c(v(Wx),f(F))|R],F) :- W == Wx,!.
floatForWar(W,[c(v(Wx),f(Fx))|R],F) :- W \== Wx,!,floatForWar(W,R,F).
floatForWar(W,[],0.0).
```

A.4. Extensiones al sistema $\mathcal{TOY}$

`initToy.pl`

Para instalar estos comandos es necesario copiar el código siguiente en el archivo `initToy.pl` tras la declaración de los predicados `processC`.

```
% ------------------------------------------------------------------------------
% (QLP) Inicio

% qlpload - Carga el compilador de QLP
processC("qlpload") :-
    !,
    load_files(qlp_compil,[if(changed),imports([qlpcompil/1,qlpcompilg/1])]),
    nl,write('QLP Compiler loaded.'),nl,nl.

% qlptotoy - Compila un programa QLP.
processC(Command) :-
    append("qlptotoy(",Rest,Command),
    !,
    (
        append(ExprStr,")",Rest), name(Expr,ExprStr),
        qlpcompil(Expr),nl,nl
    ;
    nl,write('Error: qlptotoy needs one argument.'), nl
    ),
    !.

% qlpgoal - Lanza un objetivo QLP.
processC(Command) :-
    append("qlpgoal(",Rest,Command),
    !,
    (
        append(ExprStr,")",Rest), name(Expr,ExprStr),
        % Escribimos el objetivo en un archivo
        open('qlpgoal.tmp.qlp',write,Hout),
        write(Hout,Expr),
        close(Hout),
        % Traducimos el objetivo
        qlpcompilg('qlpgoal'),
        % Lanzamos el objetivo
        open('qlpgoal.tmp.toy',read,H),
        read_line(H,S),
        close(H),
        process(S)
    ;
    nl,write('Error: qlpgoal needs one argument.'), nl
    ),
    !.

% (QLP) Fin
% ------------------------------------------------------------------------------
```

Bibliografía

[1] Krzysztof R. Apt. Logic programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, pages 493–574. Elsevier and The MIT Press, 1990.

[2] Krzysztof R. Apt and M. Gabbrielli. Declarative interpretations reconsidered. In P. van Hentenryck, editor, *Proceedings of the 11th International Conference on Logic Programming (ICLP'94)*, pages 74–89. The MIT Press, 1994.

[3] Krzysztof R. Apt and M.H. van Emden. Contributions to the theory of logic programming. *Journal of the Association for Computing Machinery (JACM)*, 29(3):841–862, 1982.

[4] F. Arcelli and F. Formato. Likelog: a logic programming language for flexible data retrieval. In *Proceedings of the 1999 ACM symposium on Applied computing (SAC'99)*, pages 260–267, New York, NY, USA, 1999. ACM Press.

[5] P. Arenas, A.J. Fernández, A. Gil, F.J. López-Fraguas, M. Rodríguez-Artalejo, and F. Sáenz-Pérez. $\mathcal{TOY}$, a multiparadigm declarative language. version 2.2.3, 2006. R. Caballero and J. Sánchez (Eds.), Available at `http://toy.sourceforge.net`.

[6] Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[7] J.F. Baldwin. Support logic programming. *International Journal of Intelligent Systems*, 1:73–104, 1986.

[8] J.F. Baldwin. Evidential support logic programming. *Fuzzy Sets and Systems*, 24(1):1–26, 1987.

[9] M. Ben-Ari. *Mathematical Logic for Computer Science*. Springer Verlag, 2001.

[10] V. Biazzo, R. Giugno, T. Lukasiewicz, and V.S. Subrahmanian. Temporal probabilistic object bases. *IEEE Transactions on Knowledge and Data Engineering*, 15(4):921–939, 2003.

[11] H.A. Blair and V.S. Subrahmanian. Paraconsistent logic programming. *Theoretical Computer Science*, 68(2):35–54, 1987.

[12] H.A. Blair and V.S. Subrahmanian. Paraconsistent foundations for logic programming. *Journal of Non-Classical Logic*, 5(2):45–73, 1988.

[13] K.L. Clark. Predicate logic as a computational formalism (res. report doc 79/59). Technical report, Imperial College, Dept. of Computing, London, 1979.

[14] A. Dekhtyar, R. Ross, and V.S. Subrahmanian. Probabilistic temporal databases, I:Algebra. *ACM Transactions on Database Systems*, 26(1):41–95, 2001.

[15] R. del Vado-Vírseda. Declarative constraint programming with definitional trees. In Bernhard Gramlich, editor, *Proceedings of the 5th International Conference on Frontiers of Combining Systems (FroCoS'05)*, volume 3717 of *LNCS*, pages 184–199. Springer Verlag, 2005.

[16] J. Dix, S. Kraus, and V.S. Subrahmanian. Heterogeneous temporal probabilistic agents. *ACM Transactions on Computational Logic*, 7(1):151–198, 2006.

[17] J. Dix, M. Nanni, and V.S. Subrahmanian. Probabilistic agent programs. *ACM Transactions on Computational Logic*, 1(2):208–246, 2000.

[18] H.D. Ebbinghaus, J. Flum, and W. Thomas. *Mathematical Logic*. Springer Verlag, 1994.

[19] T. Eiter, J.J. Lu, T. Lukasiewicz, and V.S. Subrahmanian. Probabilistic object bases. *ACM Transactions on Database Systems*, 26(3):264–312, 2001.

[20] T. Eiter, V.S. Subrahmanian, and G. Pick. Heterogeneous active agents, I:Semantics. *Artificial Intelligence*, 108(1&2):179–255, 1999.

[21] R. Fagin, J.Y. Halperin, and N. Megiddo. A logic for reasoning about probabilities. *Information and Computation*, 87(11):78–128, 1990.

[22] M. Falaschi, G. Levi, M. Martelli, and C. Palamidessi. A model-theoretic reconstruction of the operational semantics of logic programs. *Information and Computation*, 102(1):86–113, 1993.

[23] J.E. Fenstad. The structure of probabilities defined on first-order languages. In R. C. Jeffrey, editor, *Studies in Inductive Logic and Probabilities*, volume 2, pages 251–262. University of California Press, 1980.

[24] Melvin Fitting. Bilattices and the semantics of logic programming. *Journal of Logic Programming*, 11:91–116, 1991.

[25] Jean H. Gallier. *Logic for Computer Science: Foundations of Automatic Theorem Proving*. Harper & Row, 1986.

[26] S. Guadarrama, S. Muñoz, and C. Vaucheret. Fuzzy prolog: A new approach using soft constraint propagation. *Fuzzy Sets and Systems*, 144(1):127–150, 2004.

[27] Carl A. Gunter and Dana S. Scott. Semantic domains. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, pages 633–674. Elsevier and The MIT Press, 1990.

[28] T. Hailperin. Probability logic. *Notre Dame Journal of Formal Logic*, 25(3):198–212, 1984.

[29] M. Hanus. The integration of functions into logic programming: From theory to practice. *Journal of Logic Programming*, 9&20(1-3):583–628, 1994.

[30] M. Hanus. Curry: an integrated functional logic language, version 0.8.2, 2006. M. Hanus (Ed.), Available at `http://www.informatik.uni-kiel.de/~curry/report.html`.

[31] M. T. Hortalá González, J. Leach Albert, and M. Rodríguez Artalejo. *Matemática Discreta y Lógica Matemática (Segunda Edición)*. Editorial Complutense, 2001.

[32] M. Ishizuka and N. Kanai. Prolog-ELG incorporating fuzzy logic. *New Generation Computing*, 3(4):479–486, 1986.

[33] J. Jaffar and M. Maher. Constraint logic programming: a survey. *Journal of Logic Programming*, 19&20:503–581, 1994.

[34] J. Jaffar, M. Maher, K. Marriott, and P.J. Stuckey. Semantics of constraints logic programs. *Journal of Logic Programming*, 37(1-3):1–46, 1998.

[35] J. Jaffar, S. Michaylov, P.J. Stuckey, and R.H.C. Yap. The CLP(R) language and system. *ACM Transactions on Programming Languages and Systems*, 14(3):339–395, 1992.

[36] M. Kifer and V.S. Subrahmanian. Theory of generalized annotated logic programs and their applications. *Journal of Logic Programming*, 12(3&4):335–367, 1992.

[37] L.V.S. Lakshmanan, N. Leone, R. Ross, and V.S. Subrahmanian. Probview: A flexible probabilistic database system. *ACM Transactions on Database Systems*, 22(3):419–469, 1997.

[38] L.V.S. Lakshmanan and F. Sadri. Probabilistic deductive databases. In *Proceedings of the International Symposium on Logic Programming*, pages 254–268, 1994.

[39] L.V.S. Lakshmanan and F. Sadri. Modeling uncertainty in deductive databases. In *Proceedings of the 5th International Conference on Databases and Expert Systems Applications*, volume 856 of *LNCS*, pages 724–733. Springer Verlag, 1996.

[40] John Wylie Lloyd. *Foundations of Logic Programming, Second Edition.* Springer, 1987.

[41] V. Loia, S. Senatore, and M. I. Sessa. Similarity-based SLD resolution and its role for web knowledge discovery. *Fuzzy Sets and Systems*, 144(1):151–171, 2004.

[42] F.J. López-Fraguas, M. Rodríguez-Artalejo, and R. del Vado-Vírseda. A new generic scheme for functional logic programming with constraints. *Journal of Higher-Order and Symbolic Computation*, 20(1&2):73–122, 2007.

[43] R. Mittu and R. Ross. Building upon the coalitions agent experiment (coax)-integration of multimedia information in gccs-m. In *Proceedings of the 9th International Workshop on Multimedia*, Ischia, Italy, 2003.

[44] Raymond T. Ng and V.S. Subrahmanian. A semantical framework for supporting subjective and conditional probability in deductive databases. In *Proceedings of the International Conference on Logic Programming*, pages 565–580, 1991.

[45] Raymond T. Ng and V.S. Subrahmanian. Probabilistic logic programming. *Information and Computation*, 101(2):150–201, 1992.

[46] Raymond T. Ng and V.S. Subrahmanian. A semantical framework for supporting subjective and conditional probability in deductive databases. *Journal of Automated Reasoning*, 10(2):191–235, 1993.

[47] Liem Ngo and Peter Haddawy. Probabilistic logic programming and bayesian networks. In Kanchana Kanchanasut and Jean-Jacques Lévy, editors, *Proceedings of the Asian Computing Science Conference*, pages 286–300. Springer-Verlag, 1995.

[48] N. Nilsson. Probabilistic logic. *Artificial Intelligence Journal*, 28(1):71–87, 1986.

[49] S. Riezler. Quantitative constraint logic programming for weighted grammar applications. In C. Retoré, editor, *Proceedings of the Logical Aspects of Computational Linguistics (LACL'96)*, volume 1328 of *LNCS*, pages 346–365. Springer Verlag, 1996.

[50] S. Riezler. *Probabilistic Constraint Logic Programming.* PhD thesis, Neuphilologischen Fakultät del Universität Tübingen, 1998.

[51] John Alan Robinson. A machine-oriented logic based on the resolution principle. *Journal of the Association for Computing Machinery (ACM)*, 12(1):23–41, 1965.

[52] M. Rodríguez-Artalejo. Functional and constraint logic programming. In C. Marché H. Comon and R. Treinen, editors, *Constraints in Computational Logics, Theory and Applications*, volume 2002 of *Lecture Notes in Computer Science*, pages 202–270. Springer Verlag, 2001.

[53] U. Schöning. *Logic for Computer Scientists.* Birkhäuser Verlag, 1989.

[54] Dana S. Scott. Domains for denotational semantics. In *Proceedings of the 9th Colloquium on Automata, Languages and Programming (ICALP'82)*, volume 140 of *LNCS*, pages 577–613. Springer Verlag, 1982.

[55] Ehud Y. Shapiro. Logic programs with uncertainties: A tool for implementing rule-based systems. In Alan Bundy, editor, *Proceedings of the 8th International Joint Conference on Artificial Intelligence (IJCAI'83)*, pages 529–532, Karlsruhe, Germany, 1983.

[56] Robert F. Stärk. A direct proof for the completeness of SLD-resolution. In Egon Börger, Hans Kleine Büning, and Michael M. Richter, editors, *Proceedings of the 3rd Workshop on Computer Science Logic (CSL'89)*, volume 440 of *LNCS*, pages 382–383. Springer Verlag, 1990.

[57] V.S. Subrahmanian. On the semantics of quantitative logic programs. In *Proceedings of the 4th IEEE Symposium on Logic Programming*, pages 173–182, San Francisco, 1987.

[58] V.S. Subrahmanian. Query processing in quantitative logic programming. In *Proceedings of the 9th International Conference on Automated Deduction*, volume 310 of *LNCS*, pages 81–100, London, UK, 1988. Springer-Verlag.

[59] V.S. Subrahmanian. Uncertainty in logic programming: Some recollections. *Association for Logic Programming Newsletter*, 20(2), 2007.

[60] Alfred Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5(2):285–309, 1955.

[61] M.H. van Emden. Quantitative deduction and its fixpoint theory. *Journal of Logic Programming*, 3(1):37–53, 1986.

[62] M.H. van Emden and R.A. Kowalski. The semantics of predicate logic as a programming language. *Journal of the Association for Computing Machinery (JACM)*, 23(4):733–742, 1976.