UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE INFORMÁTICA
Departamento de Sistemas Informáticos y Computación

PROGRAMACIÓN DECLARATIVA CUALIFICADA CON RESTRICCIONES.

MEMORIA PARA OPTAR AL GRADO DE DOCTOR

PRESENTADA POR

Carlos Alberto Romero Díaz

Bajo la dirección de los doctores

Rafael Caballero
Mario Rodríguez Artalejo

Madrid, 2011

ISBN: 978-84-694-5111-3

PROGRAMACIÓN DECLARATIVA CUALIFICADA CON RESTRICCIONES

Tesis doctoral presentada por el doctorando

CARLOS • ALBERTO • ROMERO • DÍAZ

para la obtención del título de doctor en Ingeniería Informática en el departamento de ***Sistemas Informáticos y Computación*** *de la* UNIVERSIDAD COMPLUTENSE *de* MADRID

Dirigida por los profesores R. CABALLERO & M. RODRÍGUEZ ARTALEJO. Terminada en MADRID *el día viernes* 14 ***enero*** MMXI

Programación declarativa
cualificada con restricciones

Carlos Alberto Romero Díaz
DSIC ♣ **UCM**

Tesis doctoral en formato publicaciones presentada por el doctorando Carlos A. Romero Díaz en el *Departamento de Sistemas Informáticos y Computación* de la *Universidad Complutense de Madrid* para la obtención del título de doctor en ingeniería informática.

Terminada en Madrid el 14 de enero de 2011.

Título:

Programación declarativa cualificada con restricciones

Doctorando:

Carlos A. Romero Díaz (cromdia@fdi.ucm.es)
Departamento de Sistemas Informáticos y Computación, Universidad Complutense
Facultad de Informática, 28040 Madrid, España

Directores:

Rafael Caballero (rafa@sip.ucm.es)
Mario Rodríguez Artalejo (mario@sip.ucm.es)

Esta tesis doctoral ha sido realizada dentro del grupo de investigación sobre Programación Declarativa (grupo 910502 del catálogo de grupos reconocidos por la UCM) como parte de las actividades del proyecto de investigación *Foundations and Applications of Declarative Software Technologies – Software Tools and Multiparadigm Programming* (FAST-STAMP, referencia TIN2008-06622-C03-01).

Agradecimientos

La realización de esta tesis no hubiera sido posible sin la ayuda y colaboración de muchas personas e instituciones que han contribuido, de un modo u otro, a fomentar, facilitar y permitir mi trabajo de investigación.

No quisiera por ello perder esta oportunidad para mostrar mi agradecimiento a Mario y Rafa, mis directores, por su inestimable ayuda, por su colaboración en el trabajo y por la confianza depositada en mi;

a Paco por haberlo hecho posible;

al GPD y al DSIC por haberme permitido desarrollar mi labor;

a mis compañeros del 220 por el inmejorable ambiente de trabajo;

a mis amigos, especialmente a Laura, por haber estado ahí cuando hacían falta y cuando no;

a mis padres y mi familia, por haber sido los principales artífices de la persona que soy y sin los cuales no hubiera podido llegar hasta aquí;

y, finalmente, a Xavi.

Índice

(TR) **Informe técnico**
(RE) **Edición revisada**

Resumen

La incorporación del razonamiento con incertidumbre a la programación declarativa y, en especial, a la programación lógica ha sido objeto de investigación en las últimas décadas. En este periodo, se han aportado diversas propuestas con este propósito así como diferentes aplicaciones prácticas de estas propuestas. Al mismo tiempo, los esquemas CLP y CFLP de la programación lógica con restricciones y la programación lógico-funcional con restricciones, respectivamente, se han convertido en potentes marcos de programación con soporte para computaciones eficientes sobre dominios de restricciones especializados, el primero, y también con funciones perezosas al estilo de las de la programación funcional, el segundo.

Sin embargo, han sido escasas las extensiones con incertidumbre que han tomado como punto de partida alguno de los esquemas CLP o CFLP, a pesar de que la potencia de cómputo de los dominios de restricciones sí ha facilitado su implementación práctica.

Es por ello objeto de esta tesis la investigación de extensiones con incertidumbre de los esquemas anteriormente mencionados, y por tanto se desarrollan aquí principalmente dos esquemas paramétricos de programación declarativa con incertidumbre: una extensión con cualificación y proximidad del marco CLP; y una extensión con cualificación de programas CFLP de primer orden.

Para los esquemas de programación aquí desarrollados se aportan dos caracterizaciones equivalentes de la semántica declarativa — una de punto fijo a partir de un transformador de interpretaciones y otra basada en un cálculo lógico o una lógica de reescritura, según sea el esquema de partida. Se aportan también una noción declarativa de objetivo y solución, y diferentes métodos de resolución de objetivos basados en técnicas de transformación de programas, que conducen a la obtención de programas y objetivos equivalentes para los que pueden computarse respuestas adecuadas.

Estas técnicas de transformación permiten además implementar de manera sencilla y natural distintas instancias útiles de los esquemas propuestos en un prototipo que está públicamente disponible y que, sobre sistemas CLP actuales, hace posible la ejecución de ejemplos y programas propuestos a lo largo de la tesis y la resolución de objetivos arbitrarios para dichos programas.

Palabras clave: dominios de cualificación, incertidumbre, programación cualificada, programación lógica, programación lógica con restricciones, programación lógico-funcional con restricciones, relaciones de similaridad, relaciones de proximidad, transformación de programas.

1. Introducción

En la lógica clásica se ha utilizado de manera general dos valores veritativos para indicar cuando una determinada fórmula lógica es consecuencia o deducible a partir de un conjunto de fórmulas lógicas. En este sentido, cuando en una interpretación una fórmula φ toma el valor *cierto* indica que, en efecto, la fórmula es deducible en la interpretación del conjunto de fórmulas iniciales dado, y toma el valor *falso* en otro caso, i.e. cuando no es deducible.

Esta visión de la lógica tiene el inconveniente de que no permite tratar adecuadamente aquellos casos en los que existe una ausencia de información total, o cuando la información disponible no resulta del todo certera. Por ello, han surgido a lo largo de los últimos 30 años una serie de aproximaciones en distintas áreas de la lógica que intentan aportar herramientas adecuadas que permitan tratar con situaciones en las que o bien no se tenga toda la información posible para poder asegurar la veracidad de las informaciones en juego, o bien se acepte que la veracidad de las informaciones puede no ser completa.

Así, surgen en general, lógicas específicas y enfoques de programación en los que resulta posible representar distintos grados o niveles de certidumbre para las fórmulas o la información, de manera que permiten trabajar sobre unos niveles de información a partir de los cuales pueden extraerse conclusiones que, de otro modo, no serían posibles.

1.1. Programación declarativa

La programación declarativa es un paradigma de programación caracterizado por la independencia del orden de evaluación y la eliminación de la asignación y otras estructuras de control características de los lenguajes imperativos. Es común en los lenguajes declarativos evitar en los programas toda indicación a *cómo* debe procederse para alcanzar los objetivos deseados, limitándose a especificar, en una cierta lógica, *qué* objetivos han de perseguirse. Por lo tanto, mientras que en el caso de los lenguajes imperativos resulta estrictamente necesario aportar un *algoritmo* que indique cómo debe ejecutarse cada programa, para los lenguajes declarativos esto no es necesario puesto que el modo de proceder vendrá determinado a partir de las posibilidades de la lógica empleada para describir el programa.

A pesar de emplear de manera general el nombre de *programación declarativa* (DP) para toda una serie de lenguajes de programación no imperativos, resulta más conveniente, y apropiado, distinguir estos lenguajes según sus características en una serie de paradigmas de programación todos ellos declarativos. Entre los paradigmas más comunes encontramos la *programación funcional* (FP), con raíces en el *cálculo lambda* [8] y que incluye lenguajes como `Haskell` [36, 30]; la *programación lógica* (LP), basada en la lógica matemática aplicada a la computación y que incluye lenguajes como `Prolog`, para el que existen múltiples implementaciones y sistemas como pueden ser `SICStus Prolog` [67] o `SWI-Prolog` [70]; la *programación con restricciones* (CP), que surgió dentro de la programación lógica aunque ahora puede asociarse a la programación funcional o, incluso, a la programación imperativa, y que podemos encontrar tanto en `SICStus Prolog` como `SWI-Prolog` bajo diferentes librerías según sea el dominio de restricciones empleado; y toda una serie de lenguajes que combinan dos o más paradigmas como es el caso de la *programación lógica con restricciones* (CLP) o la *programación lógico-funcional* (FLP) y su versión *con restricciones* (CFLP). Son representantes de este último caso los lenguajes `Toy` [6] y `Curry` [29].

A modo de ilustración, considérese los dos siguientes ejemplos en los que se define la función matemática *factorial* en `Prolog` y `Haskell`, respectivamente.

Ejemplo 1.1 (Factorial; I)**.** Predicado que calcula el factorial de un número natural n en `Prolog`.

```
1  factorial(0, 1).
2  factorial(N, F) :- N>0, N1 is N-1, factorial(N1, F1), F is N*F1.
```

Ejemplo 1.2 (Factorial; II)**.** Función que calcula el factorial de un número natural n en `Haskell`.

```
1  factorial 0 = 1
2  factorial n = n * factorial (n-1)
```

Como ya se indicó, en el caso de los lenguajes declarativos no se hace explícito junto a los programa el *cómo* proceder en cada caso, sino el *qué* debe hacerse. Esto puede verse en los dos ejemplos anteriores por los motivos siguientes:

- En el caso de `Prolog`, y de la programación lógica, se dice que un átomo es derivable de un programa cuando es posible probarlo a partir del conjunto de hechos y cláusulas del programa. En el ejemplo anterior, *factorial*$(0, 1)$ es un hecho y nos dice que efectivamente el factorial de 0 es 1. La otra cláusula del programa, la cláusula 2, nos permite probar que el factorial de un número, representado por la variable N, es otro, representado por la variable F, siempre que ambas variables tomen valores que hagan posible probar todos los átomos de su cuerpo (sin un orden determinado ya que el cuerpo representa una conjunción lógica) para algún valor válido de las variables libres que en él aparezcan (en nuestro caso son libres las variables N_1 y F_1). Así, si tenemos que N es mayor que 0, que $N_1 = N - 1$, que el factorial de N_1 es F_1 y que $F = N * F_1$, entonces podremos efectivamente decir que F es el factorial de N.
- En el caso de `Haskell`, y de la programación funcional, es aún más claro puesto que lo escrito en el programa se limita a describir el comportamiento matemático de la función *factorial* asumiendo que el lenguaje será capaz de interpretar adecuadamente dicha definición y computar efectivamente el factorial de un número n.

Sin embargo, en la práctica ocurre que en ambos lenguajes aparecen elementos no declarativos puesto que, por ejemplo, el orden de evaluación de los átomos del cuerpo de las cláusulas `Prolog` es siempre de izquierda a derecha, y la comprobación de que la expresión a evaluar encaja con uno de los lados izquierdos de las reglas `Haskell` se realiza según el orden en el que se escriben dichas reglas, eligiendo siempre la primera que encaje y descartando el resto.

El trabajo de esta tesis se enmarca dentro de la programación declarativa y, más concretamente, dentro de los paradigmas de la programación lógica (véase [44, 2]), la programación lógica con restricciones (véase [33, 34]) y la programación lógico-funcional (véase [47, 1]). Por último, a pesar que se trabaja con el objetivo de aportar cuanta información

sea necesaria para el correcto entendimiento de la tesis, se espera que el lector esté especialmente familiarizado con los conceptos y la semántica de la programación lógica y la programación lógica con restricciones.

1.2. Programación declarativa con incertidumbre

La investigación sobre la incertidumbre en el marco de la programación declarativa se ha producido durante los últimos 25 años, y muy especialmente dentro del campo de la programación lógica. En este campo se han tratado diversas aproximaciones a la semántica declarativa y operacional, así como a la aplicación práctica de estos enfoques de la programación lógica con soporte para la incertidumbre y el razonamiento incierto.

Desde que L. A. Zadeh [76] definiera, en el año 1965, los *conjuntos borrosos* como una extensión de la teoría matemática de conjuntos, en la que la función de pertenencia de un conjunto indicaba el grado de pertenencia de un elemento al conjunto mediante la asignación de un número real, comprendido en el intervalo $[0, 1]$, la utilización de la incertidumbre en el ámbito de la lógica matemática y la programación ha visto numerosas aproximaciones desde diferentes enfoques que podemos englobar, de manera muy general, en cuatro grandes líneas de trabajo:

- *Enfoques anotados* como [71, 68, 69, 41], para los que los grados de certidumbre representan el nivel de confiabilidad sobre la certeza de las pruebas, y se aporta un procedimiento de cómputo para grados de certidumbre por medio de una interpretación para la implicación y otra para la conjunción. Opcionalmente, además, puede darse interpretación a la disyunción y/u otras conectivas lógicas.

- *Enfoques multi-valuados* o *borrosos* como [28, 75, 48], para los que los dos valores veritativos clásicos de *falso* y *cierto* se reemplazan por toda una serie de valores veritativos, y para los que se proponen multitud de conectivas y agregadores lógicos que permiten interpretar de manera diferente las cláusulas de un programa.

- *Enfoques posibilistas* como [78, 19], para los que el cómputo de los grados de certidumbre se realiza atendiendo a la teoría de la posibilidad [18], un sistema deductivo que, siguiendo ideas de Rescher [54], se basa en la idea de que la fuerza de una conclusión coincide con la fuerza de su argumento más débil. Estos guardan bastante relación con los enfoques anotados con los que comparten origen, y con los enfoques multi-valuados en que existen propuestas posibilistas que amplían el número de valores veritativos.

- *Enfoques probabilísticos* como [51, 56], para los que el cómputo de los grados de certidumbre se realiza atendiendo a las condiciones de la teoría matemática de la probabilidad.

Además, como ocurre de forma normal en la investigación, el enriquecimiento mutuo de estas líneas de trabajo hace que algunas propuestas encajen en más de una línea de trabajo según cómo se interpreten o qué elementos se consideren. Así tenemos *propuestas basadas en relaciones de similaridad* como [77, 4, 64, 45], o en *relaciones de proximidad* como [77, 17, 66, 38], que pueden ser anotadas o multi-valuadas según la interpretación de los grados de certidumbre y el número de conectivas y agregadores lógicos empleados.

En lo que respecta a esta tesis, puede decirse que el trabajo realizado se enmarca principalmente dentro de los enfoques anotados, no teniendo relación alguna con los enfoques probabilísticos dado que la teoría de la probabilidad no jugará papel alguno en las interpretaciones de implicaciones y conjunciones, y poca relación con los multi-valuados en el sentido de que la elección de la interpretación de las conectivas lógicas de implicación y conjunción está determinada y es única para un programa. En relación con los enfoques posibilistas, aún pudiendo decirse que resultan bastante cercanos a las propuestas de esta tesis, resultan bastante más restrictivos en el sentido de que exigen una serie de condiciones que no se cumplen, en general, en nuestras propuestas, e.g. que el retículo que contiene los valores de certidumbre sea completo y su orden total.

1.2.1. Programación lógica cuantitativa

La *programación lógica cuantitativa* comienza fundamentalmente con un trabajo de Shapiro [65] de 1983, en el que se proponía el uso del intervalo $[0, 1]$ de forma similar a como lo había usado Zadeh con anterioridad pero, esta vez, como *valores de certidumbre* que indicaban cuál era el grado con el que se podía confiar en la veracidad de la información proporcionada por las cláusulas de un programa lógico. Además, permitía utilizar diferentes *funciones de certidumbre* para propagar los factores de certidumbre que podían obtenerse para los cuerpos de las cláusulas a las cabezas.

En 1986, M. H. van Emden [71] consideró los programas lógicos cuantitativos (QLP por *quantitative logic program*) como conjuntos de cláusulas anotadas de la forma

$$A \leftarrow\!\!\textcircled{f}\!\!- B_1,\ \ldots,\ B_m$$

con un factor de atenuación f en su implicación, que tenían la característica fundamental de que empleaban una única función de certidumbre $\times$, ahora denominada *función de atenuación*, que debía emplearse para la propagación de los valores de certidumbre en todas las cláusulas del programa. Así, podía decirse que si b era el grado de certidumbre con el que podía probarse el cuerpo de una cláusula, calculado este a su vez como el mínimo de los grados de certidumbre con el que se probaban todos los átomos del cuerpo, $f \times b$ sería el grado de certidumbre para el que podía probarse la cabeza de dicha cláusula. Aunque el enfoque de van Emden resultaba menos general que el Shapiro, debido a la elección fija de una función de certidumbre particular, permitía al mismo tiempo probar resultados bastante más generales para la teoría de modelos y la semántica de punto fijo, similares incluso a aquellos obtenidos en [72, 3] para la programación lógica clásica. Además, van Emden aportaba, en el mismo trabajo, un procedimiento para el cómputo de los valores de certidumbre de los átomos en el menor modelo de Herbrand para un programa dado, mediante la aplicación de una heurística alfa-beta a los átomos de un árbol de búsqueda `and/or`. Sin embargo, este procedimiento únicamente funcionaba para átomos cerrados y para los que existiese un árbol de búsqueda finito.

Dos trabajos posteriores de V. S. Subrahmanian [68, 69] introdujeron un retículo especial $\mathcal{SL}$ en sustitución del retículo de los números reales en el intervalo $[0, 1]$ con el orden natural que había sido utilizado hasta el momento. $\mathcal{SL}$ incluía dos copias isomorfas de $[0, 1]$ cuyos elementos eran incomparables bajo el orden de $\mathcal{SL}$, y que podían utilizarse separadamente para representar el grado de *verdad* y de *falsedad*, respectivamente, permitiendo

así el uso de un sencillo tipo de negación en el cuerpo de las cláusulas. Otras contribuciones relevantes de [68, 69] fueron: a) la introducción de *cláusulas de programa anotadas* —generalizadas posteriormente en [41] a un marco mucho más expresivo y del que hablaremos a continuación— que relajaban la restricción impuesta por van Emden de emplear una única función de certidumbre; y b) la introducción de un procedimiento de resolución de objetivos mucho más conveniente y potente que el dado en [71].

1.2.2. Programación lógica anotada

La *programación lógica anotada* tiene su máximo exponente en la teoría de programas lógicos anotados generalizados (GAPs por *generalized annotated program*s) [41], una propuesta de Kifer y Subrahmanian del año 1992. En esencia, dado un semirretículo superior de valores de verdad, no necesariamente el formado por el intervalo $[0, 1]$ con el orden natural, se dice que un programa GAP es un conjunto de cláusulas anotadas de la forma

$$A : \rho \ \leftarrow \ B_1 : \mu_1 \ \& \ \ldots \ \& \ B_m : \mu_m$$

en la que $A : \rho$ es un átomo anotado, cada $B_i : \mu_i$ es un átomo c-anotado o v-anotado, según sea μ_i una constante o una variable, respectivamente, y en donde & representa la conjunción lógica.

Dado que la anotación ρ de la cabeza de las cláusulas puede ser una constante, i.e. un valor del semirretículo, una variable o un término de anotación complejo (que puede contener funciones definidas con los elementos del semirretículo), es fácil ver que las cláusulas de los programas QLP de van Emden pueden representarse en GAP con cláusulas anotadas de la siguiente forma

$$A : f \times \min\{\mu_1, \ldots, \mu_m\} \ \leftarrow \ B_1 : \mu_1, \ \& \ \ldots \ \& \ B_m : \mu_m$$

y por lo tanto GAP es, intuitivamente, más general que QLP. En relación a su semántica declarativa, se presentaron para GAP resultados más generales que los aportados por van Emden para QLP, y el procedimiento de resolución propuesto para GAP mejoraba de forma significativa aquel existente para QLP.

En todo caso, las interpretaciones en GAP se definen sobre el conjunto de átomos cerrados (la base de Herbrand), y el hecho de utilizar como anotación para un átomo cerrado del modelo mínimo de un programa el supremo (*lub*) de todas las anotaciones con la que es posible probarlo, obliga a introducir una compleja, y costosa en lo que a implementación se refiere, noción de *reductante* para permitir computar con una "unión" de cuerpos de cláusulas cuyas cabezas son todas unificables con el átomo en cuestión. Estos elementos limitan, en cierto modo, los resultados de la semántica declarativa y operacional de GAP ya que, por otro lado, su muy general sintaxis permitía, en efecto, englobar muchos de los enfoques conocidos, hasta el momento, sobre el razonamiento incierto o con incertidumbre.

1.2.3. Programación lógica multi-valuada

Una propuesta alternativa para tratar la incertidumbre a la programación cuantificada o anotada que se ha comentado hasta ahora, consiste en la utilización de lógicas multi-valuadas, como es el caso de los birretículos de Fitting [20], que son, intuitivamente, estructuras ordenadas mediante dos órdenes diferentes, uno de conocimiento $\leq_k$, al estilo

del utilizado en las propuestas anteriores, y otro de verdad $\leq_t$, en el que un elemento se dice menor o igual que otro cuando se prueba más falso o igual de falso que otro. Los birretículos son una generalización de la lógica multi-valuada $\mathcal{FOUR}$ de Belnap [9, 74] que, como puede verse en la figura 1, permite utilizar cuatro valores lógicos: 0 para la falsedad; 1 para la verdad; $\bot$ para la ausencia de información, i.e. ni falsedad ni verdad; y $\top$ para la contradicción, i.e. falsedad y verdad al mismo tiempo.

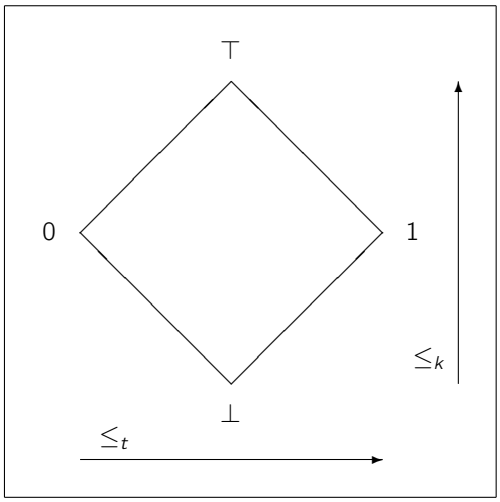

Figura 1: Birretículo de cuatro valores

Los dos órdenes que pueden definirse en el birretículo inducido por la lógica $\mathcal{FOUR}$ serían concretamente:

- El *orden de conocimiento* ($\leq_k$), que nos indica la cantidad de información que disponemos para un átomo, y ordena los valores de la lógica según la cantidad de información de la siguiente manera: $\bot$, el mínimo en este orden, representa la ausencia de información, 0 y 1 representan la misma cantidad de información pero el primero hacia la falsedad, y el segundo hacia la verdad; por último, $\top$, el máximo en este orden, representa que el átomo es falso y cierto a la vez o, lo que es lo mismo, que tenemos el máximo de información posible sobre un átomo.

- El *orden de verdad* ($\leq_t$), que nos indica cuánto de verdadero es un átomo determinado, y ordena los valores de la lógica según pueda deducirse la veracidad del átomo en cuestión. Así, 0, que representa el mínimo en este orden, indica falsedad; $\bot$ y $\top$ representan el mismo nivel de falsedad que de verdad; y 1, que representa el máximo de este orden, indica verdad.

A partir de otras lógicas multi-valuadas surgen enfoques basados en lógica borrosa como [75, 73, 27, 48, 49] y sistemas que implementan el principio de resolución de [42], presentado por Lee en 1972, como `Prolog-Elf` [32], `Fril-Prolog` [7] y `F-Prolog` [43]; cuya característica fundamental es la utilización de una serie de conectivas y agregadores lógicos multi-valuados que permiten variar la forma en que cada cláusula del programa debe interpretarse. Así tenemos, más recientemente, lenguajes como [75, 73, 27] que hacen uso de conectivas como las mostradas en la figura 2 y de los cuales la propuesta de [27] ha sido implementada sobre CLP($\mathcal{R}$). Por otro lado, basándose en la definición dada por Pavelka [52] de *par adjunto* ($\leftarrow, \&$) con respecto a un retículo dado, el marco de programación lógica multi-adjunta [48] (MALP por *multi-adjoint logic programming*) define programas lógicos con capacidad para utilizar diferentes pares adjuntos en un mismo programa, permitiendo interpretar cláusulas diferentes de un mismo programa de maneras muy diversas. La semántica procedural de MALP fue presentada en [49] y una transformación de programas MALP a programas `Prolog` equivalentes fue descrita posteriormente en [37].

Łukasiewicz
$\&_{Ł}(x,y) = \max(0, x+y-1)$
$\rightarrow_{Ł}(x,y) = \min(1, 1-x+y)$
$\vee_{Ł}(x,y) = \min(1, x+y)$

Gödel
$\&_{G}(x,y) = \min(x,y)$
$\rightarrow_{G}(x,y) = y$ if $x>y$ else 1
$\vee_{G}(x,y) = \max(x,y)$

Product
$\&_{P}(x,y) = x \cdot y$
$\rightarrow_{P}(x,y) = \min(1, y/x)$
$\vee_{P}(x,y) = x+y-x \cdot y$

Figura 2: Conectivas lógicas

Además, en la tesis doctoral de J. Penabad [53], presentada muy recientemente, se describen un conjunto de técnicas de transformación de programas borrosos, basadas en desplegado, para la optimización, especialmente, de programas multi-adjuntos. Se espera que estas técnicas contribuyan a la mejora y el desarrollo del sistema `FLOPER` [21], también implementado sobre CLP($\mathcal{R}$), y que permite ejecutar un subconjunto de programas multi-adjuntos.

1.2.4. Programación lógica con similaridad

Un enfoque más reciente que los anteriores es el de la programación lógica con incertidumbre basada en *relaciones de similaridad* (SLP por *similarity-based logic programming*) presentado por M. I. Sessa en [64] y otros trabajos anteriores como [26, 22, 63]. Este enfoque utiliza también el retículo $[0, 1]$ para tratar la incertidumbre al estilo de la programación lógica anotada o de la programación lógica multi-valuada. A diferencia de como ocurre en la programación lógica anotada o en la borrosa, los programas SLP son, sencillamente, un conjunto de cláusulas de Horn como aquellas de la programación lógica clásica. La diferencia con un programa lógico clásico radica, entonces, en la presencia de una *relación de similaridad*, i.e. el análogo borroso de una relación de equivalencia, que permite establecer similitudes, bien entre símbolos de función, bien entre símbolos de predicado, y que se proporciona junto al programa para permitir unificar términos que, de otro modo, no serían unificables en el sentido clásico. Esta unificación puede verse como si fuera estratificada, i.e. como si fuera de manera que a cada grado de certidumbre que descendemos desde el 1, se permitiera una mayor cantidad de unificaciones entre elementos no unificables en niveles superiores de certidumbre. El nivel con el que dos elementos son unificables se denomina, en este caso, *grado de similaridad*.

Para la semántica operacional de los enfoques basados en similaridad existen diferentes propuestas:

- Una posibilidad es transformar el programa original con similaridad en un programa equivalente sin similaridad y aplicar la resolución SLD clásica, como se propone en [26, 63, 64].
- Alternativamente, puede también desarrollarse una resolución SLD basada en relaciones de similaridad y aplicarla al programa original con similaridad, como se propone también en [64].

Las proposiciones [64](Prop. 7.1) y [64](Prop. 7.2) demuestran la equivalencia de las respuestas calculadas en ambos modos de proceder, y los sistemas `Likelog` [4, 5] y `SiLog` [45] se desarrollaron como implementaciones del esquema SLP sobre sistemas `Prolog` para dar soporte a aplicaciones en el ámbito de la búsqueda flexible de información.

Algunos trabajos posteriores han generalizado el enfoque SLP utilizando relaciones de proximidad en lugar de relaciones de similaridad porque, como ya dijeran Shenoi y Melton

[66], la transitividad requerida por las relaciones de similaridad puede resultar conflictiva con las intenciones de los usuarios en múltiples ocasiones. Así, intuitivamente, una relación de proximidad, originalmente propuesta por Dubois y Prade en [17], es el análogo borroso de una relación binaria reflexiva y simétrica, pero no necesariamente transitiva como en el caso de las relaciones de similaridad. El lenguaje `Bousi~Prolog` [39] supone una extensión de la programación lógica con relaciones de proximidad, y existen dos implementaciones del mismo: una implementación de bajo nivel [40] basada en la adaptación de la WAM clásica, denominada *Similarity WAM* e implementada en `JAVA` y capaz de ejecutar programas `Prolog` en el contexto de una relación de proximidad definida sobre el alfabeto de primer orden inducido por el programa; y una implementación de alto nivel [38] realizada sobre el sistema `SWI-Prolog` por medio de transformaciones de programas `Bousi~Prolog` a programas `BPL Traducidos` (así denominados) que pueden ser ejecutados en un metaintérprete de acuerdo con una resolución SLD basada en proximidad al estilo de la desarrollada en [64] para SLP, pero generalizada para el caso de las relaciones de proximidad.

1.2.5. Otros marcos de programación con incertidumbre

Como se ha podido ver, muchas de las extensiones de la programación lógica clásica con incertidumbre que han llegado a implementarse, han utilizado la programación con restricciones como método de implementación por las facilidades que aporta a la hora de calcular con los grados de certidumbre. De cualquier modo, son pocas las aproximaciones a la incertidumbre que toman como base el esquema clásico de CLP de Jaffar y Lassez [33].

En particular, podemos encontrar trabajos como [55, 56], que extienden la formulación de CLP de Höhfeld y Smolka [31] con programación lógica cuantitativa en el sentido de [71], y que están motivados por problemas en el ámbito del procesamiento del lenguaje natural. Por otro lado, [10] propuso la utilización de una aproximación a CLP, basada en semianillos en lugar de retículos, en el que las restricciones se resuelven de manera difusa por medio de unos niveles de consistencia, representados por valores del semianillo. Esta aproximación, motivada por problemas de satisfacción de restricciones (CSPs por *constraint satisfaction problem*s), fue implementada con `clp(FD,S)` en [25] para una clase particular de semianillos que permitían utilizar algoritmos de consistencia local.

En el campo más específico de la programación lógico-funcional, son muy pocas las aproximaciones con incertidumbre, y básicamente se concentran dentro del marco multi-adjunto de la programación lógica multi-valuada, para el que se ha investigado una unificación basada en similaridad para extender el procedimiento de estrechamiento necesario, que es el principal procedimiento de resolución de objetivos para lenguajes FLP [50]. Al igual que en [64], las relaciones de similaridad de [50] toman valores en el intervalo real $[0, 1]$.

1.3. Objetivos

El objetivo fundamental de esta tesis es tanto el estudio, investigación y desarrollo de marcos de programación declarativa de semántica rigurosa y con expresividad suficiente para el correcto tratamiento de la incertidumbre y del razonamiento incierto, como el desarrollo de la implementación práctica de dichos esquemas y la resolución de los problemas técnicos asociados a toda implementación práctica de un marco teórico de programación.

Más concretamente, se propone contribuir en dos áreas fundamentales: por un lado, en el área de la programación declarativa, aportando un marco teórico tanto para programación lógica como para programación lógico-funcional que incorpore incertidumbre, y que permita construir programas declarativos en los que la incertidumbre juegue un papel relevante; y por otro lado, en la implementación de un lenguaje con incertidumbre, mediante técnicas que aprovechen la programación con restricciones disponible en CLP o en CFLP para representar efectivamente programas con incertidumbre como programas CLP o CFLP equivalentes, y que permita, a su vez, la construcción de aplicaciones prácticas en el área de la incertidumbre, mejorando lenguajes existentes de enfoques relacionados al permitir la utilización de restricciones de un dominio de restricciones y funciones perezosas indeterministas.

De manera más esquemática, podemos englobar los objetivos de esta tesis en los siguientes puntos:

1. Generalizar el concepto de incertidumbre para permitir utilizar otros elementos distintos de los contenidos en el retículo $[0, 1]$, permitiendo representar otras características de los programas además del grado sobre la certidumbre de las cláusulas de los programas. En particular, se propone contribuir construyendo una nueva noción de *cualificación*, más general que las empleadas por enfoques relacionados, que satisfaga ciertos axiomas que garantizan la compatibilidad con la noción clásica de incertidumbre, y que permita razonar en los programas sobre otros elementos distintos de la incertidumbre como puede ser la complejidad de la demostración.

2. Presentar un marco teórico de semántica natural y rigurosa para la programación lógica con cualificación que sirva de base para la investigación posterior. En concreto, se pretende conseguir una semántica declarativa clara que permita, además de admitir caracterizaciones prácticas de interpretaciones y modelos, razonar adecuadamente sobre la corrección de los métodos de resolución de objetivos y de las técnicas de implementación.

3. Estudiar las posibles extensiones de la semántica y la expresividad del marco desarrollado en el punto anterior, así como proponer esquemas más generales que engloben o incorporen otras propuestas en el área, como por ejemplo la programación lógica basada en similaridad al estilo de [64] o la programación lógico-funcional con restricciones de [47].

4. Implementar instancias útiles de los esquemas de programación investigados para facilitar el desarrollo posterior de aplicaciones prácticas aportando herramientas útiles y sencillas de utilizar para los usuarios finales de estos marcos de programación. Más concretamente, estudiando la posibilidad de extender sistemas de programación existentes como el CFLP `Toy` [6] o los CLP `SICStus Prolog` [67] y `SWI-Prolog` [70].

1.4. Contribuciones principales

Las contribuciones principales de esta tesis se concretan en los resultados principales del conjunto de publicaciones que constituyen el contenido fundamental de esta tesis en formato de *tesis por publicaciones*, y pueden brevemente resumirse en los siguientes puntos:

1. La noción de *dominio de cualificación*, tal y como se presenta en la sección 2.1 y se define, principalmente, en las publicaciones [59](A.1, §2) y [61](A.5, §2.2), supone una generalización de la noción usual en enfoques relacionados, y permite razonar sobre diferentes maneras de asociar a cada respuesta calculada un valor de interés para el usuario que, como se verá más adelante, permite razonar tanto sobre la incertidumbre como sobre otros elementos.

2. El esquema **QLP** para la *programación lógica con cualificación*, presentado por primera vez en la publicación [59](A.1), y que sirve como base para el conjunto de marcos de programación desarrollados en el resto de publicaciones de la tesis. Este esquema de programación es una generalización de la programación lógica cuantitativa de van Emden [71], y para él se desarrollan dos caracterizaciones equivalentes de la semántica declarativa; una noción declarativa de objetivo y solución; y un procedimiento de resolución de objetivos, denominado *resolución SLD cualificada*, que se demuestra correcto y completo con respecto a la semántica declarativa. El esquema QLP presenta instancias interesantes que pueden ser fácilmente implementadas utilizando tecnología CLP, de manera que los QLP-programas pueden ser transformados a CLP-programas equivalentes que hacen uso de restricciones para computar con los valores de cualificación.

3. Un conjunto de *esquemas de programación declarativa con cualificación* que engloban a QLP y extienden su semántica y su expresividad. Estos esquemas desarrollados son:

   BQLP. Presentado en la sección 2 y en la publicación [60](A.2), es una pequeña extensión de QLP introduciendo predicados bivaluados —en el sentido de que es posible deducir átomos a partir de un programa que prueban veracidad o falsedad— y restricciones umbrales en los átomos del cuerpo de las cláusulas que limitan inferiormente la cantidad de información a partir de la cual es posible deducir otros elementos. Para este esquema, se extienden ambas caracterizaciones de la semántica de QLP y se adapta el procedimiento de resolución de objetivos.

   SQLP. Presentado en la sección 3 y en la publicación [11](A.3), es una extensión de QLP con una generalización de las relaciones de similaridad de [64], de forma que estas den valores sobre el conjunto soporte de un dominio de cualificación adecuado, en lugar del más estricto retículo de los números reales en el intervalo $[0, 1]$ con el orden estándar ($\leq$). Para este esquema se desarrolla una caracterización de la semántica declarativa y una técnica de transformación de SQLP-programas a QLP-programas equivalentes para los que puede utilizarse el procedimiento de resolución de objetivos existente. Además, se demuestra que los SLP-programas en el sentido de [64] son, en efecto, un caso particular de SQLP-programas que no hacen uso de factores de atenuación en sus cláusulas.

   QCFLP. Presentado en la sección 4 y en la publicación [13](A.4), es una extensión de la semántica y la expresividad de QLP con soporte para computaciones con funciones perezosas indeterministas y restricciones de un dominio de restricciones. En concreto, se considera un fragmento de primer orden de CFLP —un esquema general para la programación lógico-funcional con restricciones sobre un dominio de restricciones dado como parámetro presentado en [47]—, y se desarrolla una extensión del

mismo con un dominio de cualificación adecuado. Para este esquema se desarrollan dos caracterizaciones de la semántica declarativa y una técnica de transformación de QCFLP-programas en CFLP-programas equivalentes —al estilo de la desarrollada para SQLP pero con soporte para la pereza— que permite ejecutar los programas transformados sobre un sistema CFLP como `Toy` [6] o `Curry` [29].

SQCLP. Presentado en la sección 5 y en las publicaciones [61](A.5) y [15](B.5), es una extensión de SQLP que generaliza las relaciones de similaridad a relaciones de proximidad y que incorpora computaciones con restricciones de un dominio de restricciones. En concreto, se desarrollan dos caracterizaciones equivalentes de la semántica declarativa, inspiradas en la semántica para CLP basada en observables de [23], y una técnica de transformación de programas que se demuestra correcta y completa con respecto a la semántica declarativa. Esta técnica permite transformar SQCLP-programas en CLP-programas equivalentes en dos pasos: 1) eliminando la relación de proximidad — al estilo de la transformación presentada para SQLP; y 2) eliminando el dominio de cualificación — al estilo de la transformación presentada para QCFLP.

4. El sistema `(S)QCLP`, presentado en la sección 6, es un prototipo de implementación, basado en la técnica desarrollada en el informe técnico [15](B.5), de varias instancias útiles del esquema SQCLP — que engloba restricciones, cualificación y proximidad. Este sistema permite la ejecución de todos los ejemplos y programas de esta tesis a excepción de aquellos que hacen uso de funciones perezosas, i.e. los ejemplos específicos del esquema QCFLP. `(S)QCLP` está públicamente disponible en `http://gpd.sip.ucm.es/cromdia/qclp` y funciona indistintamente sobre los sistemas CLP `SICStus Prolog` y `SWI-Prolog`.

1.5. Organización de la tesis

Esta tesis sigue el formato de *tesis por publicaciones* de acuerdo con la normativa vigente de la Universidad Complutense de Madrid y se compone, por lo tanto, de una introducción — que incluye motivación, objetivo, resumen de contribuciones y conclusión— y de un apéndice con el conjunto de publicaciones asociadas a la tesis —que avalan la calidad de los resultados de la misma— en su formato y longitud original.

Las secciones 1 a 7 constituyen el resumen de las publicaciones asociadas a la tesis, e incluyen la motivación, los objetivos, las contribuciones principales de la tesis y sus conclusiones. Las publicaciones principales, y que avalan formalmente los resultados de la tesis, se presentan en el Apéndice A; el resto de publicaciones asociadas, correspondiente al conjunto de informes técnicos con versiones extendidas y con demostraciones completas, se presentan en el Apéndice B.

Las citas a referencias bibliográficas se realizan indicando únicamente con un número entre corchetes la publicación a la que hacen referencia, con el añadido de incluir opcionalmente y entre paréntesis la sección o enunciado específico que resulta de interés en dicha cita bibliográfica. Por ejemplo:

- [n] — Cita a la publicación n.
- [n](§X) — Cita a la sección X de n.

- [*n*](Def. *X*) — Cita a la definición *X* de *n*.

En el caso de las citas a las publicaciones asociadas de esta tesis, se incluye además dentro de los paréntesis opcionales el número del apéndice que contiene dicha publicación. Así:

- [*n*](A.*m*) — Cita a la publicación principal asociada *n* en el apéndice A.*m*.
- [*n*](B.*m*) — Cita a la publicación asociada *n* en el apéndice B.*m*.
- [*n*](B.*m*, §*X*) — Cita a la sección *X* de *n* en el apéndice B.*m*.
- [*n*](B.*m*, Def. *X*) — Cita a la definición *X* de *n* en el apéndice B.*m*.

De manera más específica, y excluyendo la introducción que finaliza con esta subsección, la tesis se organiza de la siguiente manera:

- La sección 2 desarrolla los esquemas QLP y BQLP que extienden la programación lógica clásica con un dominio de cualificación y otras mejoras de su expresividad mediante ampliación de la sintaxis de los programas. Los dominios de cualificación se presentan en la subsección 2.1.
- La sección 3 desarrolla el esquema SQLP de la programación lógica con cualificación y similaridad. Las relaciones de similaridad se presentan en la subsección 3.1.
- La sección 4 desarrolla el esquema QCFLP de la programación lógico-funcional de primer orden con restricciones y cualificación. Los dominios de restricciones se presentan en la subsección 4.1.
- La sección 5 desarrolla el esquema SQCLP de la programación lógica con restricciones, cualificación y proximidad. Las relaciones de proximidad se presentan, como generalización de las relaciones de similaridad, al principio de la subsección 5.1.
- La sección 6 presenta el sistema `(S)QCLP` como prototipo de implementación de varias instancias útiles del esquema SQCLP sobre los sistemas `SICStus Prolog` [67] y `SWI-Prolog` [70].
- Finalmente, la sección 7 concluye la introducción que conforma la primera parte de las tesis con formato por artículos, incidiendo en los principales resultados de la tesis y en las líneas de investigación futura.

2. Programación lógica con cualificación

La *programación lógica con cualificación* surge como una extensión de la programación lógica clásica que amplía la expresividad de los programas lógicos clásicos en el sentido de proporcionar una sintaxis y una semántica clara para permitir el modelado y uso de la incertidumbre y el razonamiento incierto, así como de otras maneras de asociar a cada respuesta calculada un valor de interés para el usuario, llamado *valor de cualificación*, en los programas lógicos. De esta manera, será posible probar no sólo que un átomo *A* es cierto, sino que dicho átomo *A* se prueba con (o para) *al menos* un determinado valor de cualificación *d*.

A nivel sintáctico, un programa lógico con cualificación es un conjunto de cláusulas o reglas de programa cualificadas. Una cláusula o regla de programa cualificada es una cláusula lógica clásica con un factor de atenuación en su implicación. Un factor de atenuación es el valor de cualificación asociado a la cláusula que, en el caso más conocido de la incertidumbre, representaría el grado de confianza en la información proporcionada por la cláusula. Al igual que ocurre en la programación lógica clásica, podemos distinguir entre hechos y cláusulas cualificadas. Así, asumiendo que A, $B_1, \ldots, B_m$ son átomos y α es un valor de cualificación, tenemos que

$$A \xleftarrow{\alpha}$$

es un hecho cualificado, y

$$A \xleftarrow{\alpha} B_1, \ \ldots, \ B_m$$

es una cláusula cualificada, en la que el átomo A es su cabeza y la conjunción de átomos $B_1, \ \ldots, \ B_m$ es su cuerpo.

Nota 1. En las publicaciones iniciales, el factor de atenuación α se representaba en las implicaciones de las cláusulas como "$\leftarrow\alpha-$" en lugar de "$\xleftarrow{\alpha}$".

A partir de una signatura universal —que proporciona símbolos de predicado y de función— y un conjunto de variables, se construyen *términos* y *átomos* de la manera usual. Así, un *término* es o bien una variable, o bien una constante (símbolo de función de aridad 0) o un símbolo de función de aridad n seguido, entre paréntesis, de n términos. Y un *átomo* es un símbolo de proposición (símbolo de predicado de aridad 0) o un símbolo de predicado de aridad n seguido, entre paréntesis, de n términos.

Si la intuición que hay detrás de una cláusula lógica como $A \leftarrow B_1, \ \ldots, \ B_m$ es que la cabeza de la cláusula debe ser cierta siempre que lo sea su cuerpo, i.e. algo equivalente a la fórmula lógica $B_1 \wedge \ldots \wedge B_m \Rightarrow A$, en el caso de la programación lógica con cualificación, la cabeza de una cláusula como $A \xleftarrow{\alpha} B_1, \ \ldots, \ B_m$ será cierta *para (o hasta)* un determinado valor de cualificación d, siempre que el cuerpo sea cierto *para (o hasta)* un valor de cualificación e y que el valor de cualificación resultante de la atenuación de e con el factor de atenuación α de la cláusula sea *igual o mejor* que d. Por lo tanto, en un marco de programación lógica con cualificación, no nos bastará con decir que un átomo A es cierto, sino que es cierto para *al menos* un determinado valor de cualificación d (en símbolos, $A\sharp d$) cuando sea posible probar el átomo A con un valor de cualificación *igual o mejor* que d, i.e. algo parecido a la fórmula $B_1\sharp e \wedge \ldots \wedge B_m\sharp e \Rightarrow A\sharp d$ con d acotado superiormente por la atenuación de e con α.

En estas circunstancias, estaremos interesados en probar la deducibilidad de átomos cualificados $A\sharp d$ con respecto a un programa lógico con cualificación dado.

El significado exacto de las expresiones "*A es cierto para (hasta) un valor de cualificación d*", "*el cuerpo es cierto para (hasta) un valor de cualificación e*", "*atenuación de e con el factor de atenuación α*" y "*un valor de cualificación es igual o mejor que otro*" quedará claro en la siguiente subsección.

2.1. Dominios de cualificación

Como es de esperar, el significado de un programa lógico con cualificación dependerá de la interpretación que se haga de los valores de cualificación y de la función de atenuación que

se utilicen en el programa. La interpretación de estos dos elementos es, precisamente, lo que proporcionan los *dominios de cualificación*.

Los *dominios de cualificación* se introdujeron por primera vez en la publicación [59](A.1) como generalización de los valores de certidumbre empleados por van Emden en [71]. El objetivo de esta generalización es no sólo permitir la utilización de la incertidumbre y el razonamiento incierto en los programas lógicos como parte de la inferencia lógica, como es el caso de los QLPs[1] de van Emden, sino también permitir utilizar otros elementos de interés para el usuario como puede ser el coste de la demostración de una respuesta calculada.

En esencia, un dominio de cualificación proporciona un conjunto de valores posibles de cualificación equipado con estructura de retículo, y una función de atenuación que se usa para calcular la propagación de valores de cualificación a través de las implicaciones lógicas de las cláusulas de los programas. La finalidad de un dominio de cualificación es aportar las herramientas necesarias para representar y razonar sobre una cualidad medible en los programas lógicos. Por lo tanto, los valores de cualificación de un dominio representan una medida específica de la cualidad sobre la que se pretende razonar. Un ejemplo de esto serían los grados de confianza o de certeza sobre la verosimilitud de la información proporcionada por un átomo, o también el coste —medido en base a la profundidad de la prueba— de obtener dicha información. Por otro lado, la finalidad de la función de atenuación es la de obtener un valor de cualificación a partir de la atenuación, o degradación, de otro valor de cualificación cuando es utilizado en un proceso de inferencia lógica con factor de atenuación. Así, la función de atenuación será la encargada de calcular el valor de cualificación con el que es posible probar la cabeza de una cláusula a partir del factor de atenuación de la cláusula y el valor de cualificación con el que se prueba su cuerpo. Por último, una conjunción de átomos podrá probarse con un valor de cualificación determinado siempre que sea posible probar cada átomo de la conjunción con dicho valor de cualificación. Esto implica, en la práctica, que el valor de cualificación con el que se prueba el cuerpo de una cláusula está acotado superiormente por el ínfimo en el dominio de cualificación de los valores de cualificación con los que se prueban los átomos del cuerpo.

Más concretamente, un dominio de cualificación $\mathcal{D}$ proporciona: un retículo $\langle D, \preccurlyeq, \mathbf{b}, \mathbf{t}\rangle$ de valores de cualificación con extremos, i.e. con un elemento mínimo $\mathbf{b}$ y otro máximo $\mathbf{t}$ con respecto al orden parcial $\preccurlyeq$; y una función de atenuación $\circ : D \times D \to D$ asociativa, conmutativa, monótona con respecto al orden del retículo ($\preccurlyeq$) y distributiva con respecto al ínfimo del retículo ($\sqcap$), que cumple además:

- $\forall d \in D : d \circ \mathbf{t} = d$ y $d \circ \mathbf{b} = \mathbf{b}$.
- $\forall d, e \in D : d \circ e \preccurlyeq e$.

Nota 2. En las publicaciones iniciales, los extremos del retículo se representaban por $\bot$ y $\top$ en lugar de $\mathbf{b}$ y $\mathbf{t}$, respectivamente; y el orden parcial se representaba por $\sqsubseteq$ en lugar de $\preccurlyeq$.

Los axiomas anteriores surgen de manera natural a partir de las características del producto ($\times$) en el retículo formado por el intervalo de los números reales entre el 0 y el 1 con el orden estándar ($\leq$).

[1] Aquí, *quantitative logic program* o programa lógico cuantitativo.

En este punto es conveniente realizar las siguientes observaciones sobre los axiomas mencionados. Como ya se ha dicho con anterioridad, un valor de cualificación aporta una medida sobre alguna característica cualitativa del programa sobre la que se desea poder razonar. En este sentido, un valor de cualificación aporta una cierta cantidad de información que puede considerarse mejor o peor según sea su posición en el orden del retículo, de manera que **b** es la peor información posible y **t** la mejor. Entonces:

1. $\forall d \in D : d \circ \mathbf{t} = d$, quiere decir que el resultado de atenuar un valor de cualificacion d con el máximo **t** del dominio es el propio d, o lo que es lo mismo, que atenuar con el máximo del dominio no empeora la cualificación conocida.

2. $\forall d \in D : d \circ \mathbf{b} = \mathbf{b}$, quiere decir que el resultado de atenuar un valor de cualificación d con el mínimo del dominio es el mínimo del dominio, o lo que es lo mismo, que no es posible extraer información mejor de una cláusula con factor de atenuación **b**, que la de átomos cualificados con **b** y, en consecuencia, triviales.

3. $\forall d, e \in D : d \circ e \preccurlyeq e$, quiere decir que no es posible mejorar la información con respecto a la proporcionada por los valores de cualificación d y e; es decir, al atenuar podremos, en el mejor de los casos, mantener la información dada por los valores de cualificación considerados pero en ningún caso mejorarla.

Nota 3. Este axioma ha sido relajado con respecto a como aparece originalmente en [59](A.1) para permitir que puedan ser iguales a e y no estrictamente peores que e. El requerir que fuera estrictamente peor que e coincide con el comportamiento de los valores de certidumbre, pero no resulta necesario para la semántica. Además, esta variación del axioma permite definir dominios de restricciones adicionales.

Los axiomas anteriores se plantearon partiendo de la base de que el mínimo del dominio representa la ausencia de información, por lo que probar un átomo con **b** resulta trivial, y por tanto la cualificación explícita de cláusulas con **b** no resulta útil. En esta misma línea y cómo veremos próximamente, el elemento mínimo de los dominios no es siempre fácilmente representable, por lo que evitar su uso explícito facilita la representación práctica de los dominios de cualificación.

Son ejemplos de dominios de cualificación los siguientes:

- **El dominio $\mathcal{B}$ de los valores booleanos.** En este dominio existen únicamente dos valores de cualificación, *falso* y *cierto*; el orden se define como *falso* $\preccurlyeq$ *cierto*, por lo que los extremos del dominio son **b** = *falso* y **t** = *cierto*; y tanto la función de atenuación, como el ínfimo del retículo, coinciden con la conjunción lógica ($\wedge$). Dado que el único valor de cualificación posible para un átomo que pueda probarse a partir de una cláusula es el de *cierto*, los programas lógicos cualificados con valores de este dominio de cualificación coinciden con los programas lógicos clásicos.

- **El dominio $\mathcal{U}$ de los valores de certidumbre.** En este dominio los valores de cualificación hacen referencia al grado de certidumbre o de confianza en la verosimilitud de la información contenida en un átomo. Los valores de cualificación posibles son los contenidos en el intervalo real $[0, 1]$; el orden parcial es el estándar ($\leq$), por lo que los extremos del dominio son **b** = 0 y **t** = 1; la función de atenuación es el

producto ($\times$); y el ínfimo del dominio coincide con la función mín. Por lo tanto, $A\sharp 1$ significa que A se demuestra con certidumbre, o confianza, máxima, y $A\sharp 0.5$ significa que A es cierto para un grado de certidumbre de *al menos* 0.5. Nótese que $A\sharp 0.5$ no significa que A tenga un 50 % de probabilidades de ser cierto —y por consiguiente, también un 50 % de probabilidades de ser falso— sino que A ha podido demostrarse para (o hasta) un grado de confianza de 0.5 según criterios heurísticos representados por las reglas del programa lógico cualificado que se haya utilizado. En general, dichas heurísticas pueden ser subjetivas y no está garantizado que obedezcan las leyes de la teoría matemática de la probabilidad.

- **El dominio $\mathcal{W}$ de la profundidad ponderada de la prueba.** En este dominio los valores de cualificación se refieren al coste de probar un átomo de acuerdo con la ponderación de cada nivel de profundidad de su demostración. Los valores de cualificación posibles son los números reales positivos y $+\infty$; el orden parcial es el inverso del estándar ($\geq$), por lo que $\mathbf{b} = +\infty$ y $\mathbf{t} = 0$; la función de atenuación es la suma ($+$); y el ínfimo del dominio coincide con la función máx. Por lo tanto, $A\sharp 0$ significa que podemos probar el átomo A sin coste alguno por su prueba (nótese que esto no tiene influencia sobre la forma de la demostración de A); $A\sharp 5$ significa que para probar A deberemos pagar un coste de orden *al menos* 5; y $A\sharp(+\infty)$ significa que puede probarse A siempre que paguemos un coste de orden infinito, lo que convierte al átomo A en un átomo trivial.

- **Los dominios producto cartesiano estrictos.** Una forma adicional de construir dominios de cualificación es mediante el producto cartesiano estricto de otros dos dominios de cualificación dados. El producto cartesiano estricto de dos dominios de cualificación se define de la manera natural por componentes. Sus valores de cualificación son parejas (d_1, d_2) en las que la primera componente d_1 es un valor de cualificación de $\mathcal{D}_1$ y la segunda componente d_2 es un valor de cualificación de $\mathcal{D}_2$. Estos dominios producto son la consecuencia de un refinamiento desarrollado en [61](A.5) del dominio producto tal y como fue definido en [59](A.1). La finalidad de dicho refinamiento fue la de evitar los pares problemáticos que contenían el mínimo de uno de los dominios de cualificación como una de sus componentes. Así, los elementos del dominio producto cartesiano estricto son, en la práctica, la pareja $(\mathbf{b}_1, \mathbf{b}_2)$ junto con todas las parejas (d_1, d_2) tales que $d_1 \neq \mathbf{b}_1$ y $d_2 \neq \mathbf{b}_2$.

2.2. Sintaxis

Para la formalización de la sintaxis completa nos referiremos a la programación lógica con cualificación como un esquema de programación paramétrico con un único parámetro: el dominio de cualificación. Así, llamaremos QLP($\mathcal{D}$) a la instancia específica del esquema QLP para el dominio de cualificación $\mathcal{D}$, y será QLP($\mathcal{U}$) la instancia correspondiente a programación lógica con incertidumbre; QLP($\mathcal{W}$) la instancia correspondiente a programación lógica con profundidades ponderadas de la prueba; y así sucesivamente.

Como ya se dijo anteriormente, un QLP($\mathcal{D}$)-programa $\mathcal{P}$ es un conjunto de cláusulas cualificadas de la forma

$$A \overset{\alpha}{\leftarrow} B_1,\ \ldots,\ B_m$$

donde $\alpha \in D \setminus \{\mathbf{b}\}$ es un factor de atenuación —distinto del mínimo del dominio de cualificación— y A, B_1, ..., B_m son átomos. Y nuestro interés radicará en la prueba de átomos cualificados $A\sharp d$, o átomos que pueden probarse para un valor de cualificación de al menos d, a partir de un QLP($\mathcal{D}$)-programa $\mathcal{P}$.

Considérese el siguiente QLP($\mathcal{W}$)-programa como ilustración.

Ejemplo 2.1 (QLP: Números de Peano)**.** Las dos cláusulas siguientes son una posible representación de los conocidos números de Peano en la instancia QLP($\mathcal{W}$):

$$c_1 \quad num(c) \overset{0}{\leftarrow}$$
$$c_2 \quad num(s(X)) \overset{1}{\leftarrow} num(X)$$

Y el predicado *suma*, tal y como se muestra a continuación, permite sumar números de Peano:

$$c_3 \quad suma(c, X, X) \overset{0}{\leftarrow}$$
$$c_4 \quad suma(s(X), Y, s(Z)) \overset{1}{\leftarrow} suma(X, Y, Z)$$

Como puede verse, el efecto de los factores de atenuación en este programa indica que las cláusulas C_1 y C_3 no requieren pagar coste alguno para su utilización, y las cláusulas C_2 y C_4 requieren que se pague al menos 1 por cada nivel de profundidad en el que se utilicen en una demostración. Por lo tanto, podemos ver de manera intuitiva que $num(c)\sharp 0$ debe tenerse debido a C_1 y que $num(s(X))\sharp(1+d)$ debe tenerse siempre que tengamos $num(X)\sharp d$ para alguna instanciación válida de la variable X. De manera similar ocurre para las cláusulas C_3 y C_4.

En la publicación posterior [60](A.2), se extendió la sintaxis de los QLP-programas dando lugar al esquema paramétrico BQLP de instancias BQLP($\mathcal{D}$). Esta extensión consistió:

1. Por un lado, en la definición de *restricciones umbral* en los cuerpo de las cláusulas para permitir imponer cotas inferiores a los valores de cualificación calculados para cada uno de los átomos del cuerpo. De esta forma se consigue evitar inferencias con premisas insuficientemente cualificadas.

2. Y por otro lado, en la incorporación de un sencillo tipo de negación mediante el uso de *átomos marcados* $A\sharp\mathtt{tt}$ y $A\sharp\mathtt{ff}$, donde `tt` y `ff` representan los dos valores veritativos clásicos de *cierto* y *falso*, respectivamente. Estos átomos marcados pueden verse como aserciones lógicas de predicados bivaluados que pueden cualificarse mediante valores de cualificación de un dominio $\mathcal{D}$. Cabe aquí hacer mención a que no hay relación entre átomos ciertos y falsos en un programa más allá de la interpretación subjetiva dada por el usuario a los predicados del programa, así puede ocurrir en un programa que un mismo átomo aparezca marcado con `tt` y con `ff` y que ambos sean deducibles al mismo tiempo.

El resultado de esta extensión de la sintaxis de los programas QLP queda reflejada en las cláusulas cualificadas al pasar a ser de la forma

$$A\sharp v \overset{\alpha}{\leftarrow} B_1\sharp(v_1, w_1), \ \ldots, \ B_m\sharp(v_m, w_m)$$

donde A, B_1, ..., B_m son átomos, $v, v_1, \ldots, v_m \in \{\texttt{tt}, \texttt{ff}\}$ indican si el átomo prueba verdad o falsedad y $w_1, \ldots, w_m \in (D \setminus \{\mathbf{b}\}) \uplus \{?\}$ establecen cotas inferiores para los valores de cualificación con los que se debe probar cada átomo del cuerpo. La notación "?" indica que no se establece cota inferior y que, por lo tanto, cualquier valor de cualificación es aceptable. Como convenio, se asume que un átomo $B\sharp\texttt{tt}$ o $B\sharp\texttt{ff}$, en el cuerpo de una cláusula, abrevia $B\sharp(\texttt{tt}, ?)$ o $B\sharp(\texttt{ff}, ?)$, respectivamente.

Finalmente, diremos ahora que probamos átomos cualificados de la forma $A\sharp(v, d)$ con $v \in \{\texttt{tt}, \texttt{ff}\}$ y $d \in D \setminus \{\mathbf{b}\}$, que deben entenderse como que *A se prueba cierto o falso (según sea v) para un valor de cualificación de al menos d*. Nótese que (v, d) no es un valor de cualificación en sí, sino una anotación compuesta por una marca y un valor de cualificación. Así, si d fuera un valor de un dominio producto cartesiano estricto, entonces sería de la forma (d_1, d_2) y, por lo tanto, $(v, d) = (v, (d_1, d_2))$.

Como ilustración de la sintaxis BQLP, considérense los dos siguientes ejemplos en los que se muestra la versión BQLP($\mathcal{W}$) del ejemplo 2.1, y un programa más completo para la instancia BQLP($\mathcal{U}\otimes\mathcal{W}$) del esquema.

Ejemplo 2.2 (BQLP: Números de Peano)**.** Las cuatro cláusulas siguientes son la representación directa y natural de los números de Peano y de su predicado *suma*, tal y como fueron presentados en el ejemplo 2.1:

c_1 $\quad num(c)\sharp\texttt{tt} \xleftarrow{0}$

c_2 $\quad num(s(X))\sharp\texttt{tt} \xleftarrow{1} num(X)\sharp(\texttt{tt}, ?)$

c_3 $\quad suma(c, X, X)\sharp\texttt{tt} \xleftarrow{0}$

c_4 $\quad suma(s(X), Y, s(Z))\sharp\texttt{tt} \xleftarrow{1} suma(X, Y, Z)\sharp(\texttt{tt}, ?)$

Nótese que en este ejemplo no se hace uso de restricciones umbral en los cuerpos de las cláusulas para imponer condiciones adicionales (podíamos también haber omitido la "?") y que las marcas empleadas corresponden todas a probar la veracidad (i.e. tt) del átomo en cuestión.

Ejemplo 2.3 (BQLP: Alimentación)**.** Sea $\mathcal{D} = \mathcal{U}\otimes\mathcal{W}$ el dominio de cualificación producto cartesiano estricto entre $\mathcal{U}$ y $\mathcal{W}$. Entonces el siguiente programa $\mathcal{P}_r$ es un BQLP($\mathcal{U}\otimes\mathcal{W}$)-programa válido.

1 $\quad persona(juan)\sharp\texttt{tt} \xleftarrow{(1,0)}$

2 $\quad persona(ana)\sharp\texttt{tt} \xleftarrow{(1,0)}$

3 $\quad persona(h(X))\sharp\texttt{tt} \xleftarrow{(1,1)} persona(X)\sharp(\texttt{tt}, ?)$

4 $\quad come(juan, insectos)\sharp\texttt{ff} \xleftarrow{(0.9,0)}$

5 $\quad come(juan, arroz)\sharp\texttt{tt} \xleftarrow{(0.85,0)}$

6 $\quad come(ana, carne)\sharp\texttt{ff} \xleftarrow{(0.6,0)}$

7 $\quad come(ana, pollo)\sharp\texttt{tt} \xleftarrow{(0.9,0)}$

8 $\quad come(h(X), Y)\sharp\texttt{ff} \xleftarrow{(0.8,1)} persona(X)\sharp(\texttt{tt}, ?),\ come(X, Y)\sharp(\texttt{ff}, ?)$

9 $\quad come(h(X), Y)\sharp\texttt{tt} \xleftarrow{(0.9,1)} persona(X)\sharp(\texttt{tt}, ?),\ come(X, Y)\sharp(\texttt{tt}, ?)$

En este programa las cualificaciones (d, e) indican, respectivamente, la certidumbre con

la que se confía en la veracidad de los predicados y el coste computacional de obtenerlos (o de demostrarlos). Como puede observarse, las cláusulas 1, 2 y 3 definen un conjunto de personas, en particular, *juan*, *ana* y todas aquellas personas que son descendientes de alguno de ellos. Las cláusulas 4, 5, 6 y 7 informan sobre los gustos alimenticios tanto de *juan* como de *ana*, indicando qué alimentos comen y cuáles no. Por último las cláusulas 8 y 9 permiten deducir los gustos alimenticios de sus descendientes a partir de los gustos alimenticios de los progenitores, partiendo de la base de que los descendientes se alimentarán de la misma forma que sus progenitores, aunque atenuando la certidumbre y aumentando el coste conforme aumenta el número de generaciones intermedias.

La exposición que resta de esta sección se centra en el esquema BQLP, dado que los resultados equivalentes para QLP pueden obtenerse muy fácilmente por simplificación.

2.3. Semántica declarativa

La semántica declarativa de los esquemas QLP y BQLP se desarrolló en [59](A.1) y en [60](A.2), respectivamente. Una versión extendida con las demostraciones de los principales resultados de [59](A.1) puede encontrarse en [58](B.1). A continuación se hace una exposición resumida de la semántica declarativa del esquema BQLP.

Como es costumbre en programación lógica, la semántica declarativa puede presentarse de varias maneras diferentes. En nuestro caso, presentamos dos caracterizaciones alternativas de la semántica declarativa: una semántica de punto fijo y una semántica basada en un cálculo lógico. En ambos casos coinciden las nociones de *interpretación*, *consecuencia inmediata* y *modelo* de un programa lógico con cualificación.

2.3.1. Interpretaciones y modelos

A diferencia de como resulta más común en las presentaciones de la semántica declarativa de los esquemas basados en programación lógica, se define una *interpretación* como un conjunto de átomos cualificados que satisfacen la condición de ser cerrados respecto a una relación denominada $\langle\mathcal{D}\rangle$*-implicación*. Las $\langle\mathcal{D}\rangle$-implicaciones de un átomo cualificado son, intuitivamente, aquellos átomos cualificados cuya información es más particular que la información dada por el primero. O, de otro modo, aquellos átomos cualificados que han de ser necesariamente ciertos cuando lo sea el primero. Formalmente, se dice que $A\sharp(v, d)$ $\langle\mathcal{D}\rangle$-implica $A'\sharp(v', d')$, en símbolos $A\sharp(v, d) \succcurlyeq_{\mathcal{D}} A'\sharp(v', d')$, sii existe una sustitución θ tal que A' es una instancia de A (i.e. $A' = A\theta$), v' es la misma marca que v (i.e. $v' = v$) y d' es un valor de cualificación acotado superiormente por d (i.e. $d' \preccurlyeq d$). Por lo tanto, una interpretación $\mathcal{I}$ es un conjunto de átomos cualificados cerrado bajo la relación $\succcurlyeq_{\mathcal{D}}$.

El siguiente ejemplo ilustra esta noción de interpetación.

Ejemplo 2.4 (Interpretaciones)**.** Sea $\mathcal{D} = \mathcal{U}$ el dominio de cualificación de los valores de incertidumbre, y sean p, a y b un símbolo de predicado unario y dos constantes, respectivamente.

Entonces:

1. La base de Herbrand abierta es el conjunto:

$$At_\Sigma = \left\{ \begin{array}{llll} p(X)\sharp(\mathtt{tt},1), & \ldots, & p(X)\sharp(\mathtt{tt},0.9), & \ldots, \\ p(X)\sharp(\mathtt{ff},1), & \ldots, & p(X)\sharp(\mathtt{ff},0.9), & \ldots, \\ p(a)\sharp(\mathtt{tt},1), & \ldots, & p(a)\sharp(\mathtt{tt},0.9), & \ldots, \\ p(a)\sharp(\mathtt{ff},1), & \ldots, & p(a)\sharp(\mathtt{ff},0.9), & \ldots, \\ p(b)\sharp(\mathtt{tt},1), & \ldots, & p(b)\sharp(\mathtt{tt},0.9), & \ldots, \\ p(b)\sharp(\mathtt{ff},1), & \ldots, & p(b)\sharp(\mathtt{ff},0.9), & \ldots \end{array} \right\}$$

2. Son interpretaciones: $\emptyset$, At_Σ y $\{p(a)\sharp(\mathtt{tt},d) \mid 0 < d \leq 1\}$, entre otras. Nótese que resulta muy sencillo comprobar que los tres conjuntos mencionados satisfacen que son cerrados con respecto a la relación de $\langle\mathcal{D}\rangle$-implicación, por lo que son, en efecto, interpretaciones.

Una noción auxiliar, pero que resulta interesante por facilitar la comprensión del resto de la semántica, es la de *consecuencia inmediata* de una interpretación dada con respecto a una cláusula también dada. Intuitivamente, son consecuencias inmediatas de una interpretación dada, con respecto a una cláusula lógica cualificada también dada, aquellos átomos cualificados que coinciden con la cabeza de la cláusula instanciada —para alguna sustitución—, y cuyos átomos del cuerpo pertenecen a la interpretación dada. Es decir, aquellos átomos cualificados que pueden probarse (en un único paso de inferencia lógica) a partir de los átomos contenidos en la interpretación y de acuerdo con la cláusula proporcionada. Formalmente, dadas una interpretación $\mathcal{I}$ y una cláusula C de la forma

$$H\sharp v \xleftarrow{\alpha} B_1\sharp(v_1, w_1),\ \ldots,\ B_m\sharp(v_m, w_m)$$

se dice que un átomo cualificado $A\sharp(v,d)$ es consecuencia inmediata de $\mathcal{I}$ con respecto a C, sii existe una sustitución θ tal que $A = H\theta$, y una serie de valores de cualificación $d_1, \ldots, d_m \in D \setminus \{\mathbf{b}\}$ que cumplen (1) y (2) como siguen:

1. $B_i\theta\sharp(v_i, d_i) \in \mathcal{I}$ con $d_i \trianglerighteq^? w_i$ para $i = 1 \ldots m$.
2. $d \trianglelefteq \alpha \circ \bigsqcap\{d_1, \ldots, d_m\}$.

La notación "$d \trianglerighteq^? w$" es trivialmente cierta si $w = ?$; y codifica "$d \trianglerighteq w$" en otro caso.

A continuación, resulta sencillo definir las nociones de *modelo de una cláusula* y *modelo de un programa*:

- una interpretación $\mathcal{I}$ es *modelo de una cláusula* C de $\mathcal{P}$, sii todo átomo cualificado, que es consecuencia inmediata de $\mathcal{I}$ con respecto a C, pertenece a $\mathcal{I}$; y
- una interpretación $\mathcal{I}$ es *modelo de un programa* $\mathcal{P}$, sii $\mathcal{I}$ es modelo de cada cláusula C de $\mathcal{P}$.

2.3.2. Semántica de punto fijo

Como es de sobra conocido, una manera de caracterizar los modelos y modelos mínimos de un programa lógico es mediante el uso de operadores de consecuencia inmediata que actúan como transformadores de interpretaciones. Para comenzar, es sencillo comprobar que el conjunto de todas las interpretaciones forma un retículo completo con respecto al orden de inclusión ($\subseteq$), donde el elemento mínimo coincide con la interpretación vacía (o conjunto vacío, $\emptyset$) y el elemento máximo con el conjunto de todos lo átomos cualificados (o base de Herbrand abierta, At_Σ). Además, para cualquier conjunto de interpretaciones I, su supremo (*lub*) $\bigsqcup I$ es la unión de las interpretaciones del conjunto I ($\bigcup_{\mathcal{I}\in I}\mathcal{I}$); y su ínfimo (*glb*) $\bigsqcap I$ es la intersección de las interpretaciones del conjunto I ($\bigcap_{\mathcal{I}\in I}\mathcal{I}$).

A continuación, el *transformador de interpretaciones* $\mathsf{T}_\mathcal{P}$ se define con la intención de obtener todas las posibles consecuencias inmediatas de una interpretación $\mathcal{I}$, con respecto a las cláusulas de un programa $\mathcal{P}$. Su definición formal es la que sigue:

$$\mathsf{T}_\mathcal{P}(\mathcal{I}) =_{\mathsf{def}} \{A\sharp(v,d) \mid A\sharp(v,d) \text{ es consecuencia inmediata de } \mathcal{I} \text{ vía alguna } C \in \mathcal{P}\}$$

donde la noción de consecuencia inmediata es tal y como se definió en la subsección anterior. El siguiente ejemplo ilustra el funcionamiento del transformador de interpretaciones $\mathsf{T}_\mathcal{P}$.

Ejemplo 2.5 (Transformador de interpretaciones)**.** Considérese $\mathcal{D} = \mathcal{U}$ y los mismos p, a y b del ejemplo 2.4. Dado entonces el programa compuesto por las dos siguientes cláusulas:

1 $p(X)\sharp\mathtt{tt} \xleftarrow{0.75}$
2 $p(b)\sharp\mathtt{ff} \xleftarrow{0.95} p(a)\sharp(\mathtt{tt},?)$

Se tiene:

$$\mathsf{T}_\mathcal{P}\uparrow^1(\emptyset) = \mathsf{T}_\mathcal{P}(\emptyset) = \left\{ \begin{array}{llll} p(X)\sharp(\mathtt{tt},0.75), & \ldots, & p(X)\sharp(\mathtt{tt},0.7), & \ldots, \\ p(a)\sharp(\mathtt{tt},0.75), & \ldots, & p(a)\sharp(\mathtt{tt},0.7), & \ldots, \\ p(b)\sharp(\mathtt{tt},0.75), & \ldots, & p(b)\sharp(\mathtt{tt},0.7), & \ldots \end{array} \right\}$$

a partir de $\emptyset$ con la cláusula 1 (que es la única con cuerpo vacío); y

$$\mathsf{T}_\mathcal{P}\uparrow^2(\emptyset) = \mathsf{T}_\mathcal{P}(\mathsf{T}_\mathcal{P}(\emptyset)) = \mathsf{T}_\mathcal{P}(\emptyset) \cup \{\, p(b)\sharp(\mathtt{ff},d) \mid 0 < d \leq 0.7125 \,\}$$

cuando se tiene que puede probarse también el cuerpo de la cláusula 2 a partir de los átomos cualificados contenidos en la interpretación $\mathsf{T}_\mathcal{P}(\emptyset)$. Finalmente, como $\mathsf{T}_\mathcal{P}\uparrow^3(\emptyset) = \mathsf{T}_\mathcal{P}(\mathsf{T}_\mathcal{P}\uparrow^2(\emptyset)) = \mathsf{T}_\mathcal{P}\uparrow^2(\emptyset)$, se tiene que $\mathsf{T}_\mathcal{P}\uparrow^2(\emptyset)$ es el menor punto fijo.

Como cabría esperar, son propiedades del transformador de interpretaciones $\mathsf{T}_\mathcal{P}$ las de ser una función bien definida —i.e. de interpretaciones en interpretaciones—, monótona y continua. Además, sus puntos prefijos coinciden con los modelos de $\mathcal{P}$ — i.e. para toda interpretación $\mathcal{I}$, $\mathcal{I}$ es modelo de $\mathcal{P}$ sii se tiene que $\mathsf{T}_\mathcal{P}(\mathcal{I}) \subseteq \mathcal{I}$.

Por último, la caracterización del modelo mínimo $\mathcal{M}_\mathcal{P}$ de $\mathcal{P}$ coincide con el menor punto fijo (*lfp*) del operador $\mathsf{T}_\mathcal{P}$, y entonces:

$$\mathcal{M}_\mathcal{P} = \mathit{lfp}(\mathsf{T}_\mathcal{P}) = \bigcup_{k\in\mathbb{N}} \mathsf{T}_\mathcal{P}\uparrow^k(\emptyset)$$

Ejemplo 2.6 (Modelo mínimo)**.** Considérese el programa del ejemplo 2.2. Por la definición del transformador de interpretaciones $\mathsf{T}_{\mathcal{P}}$ sabemos que:

$$\mathsf{T}_{\mathcal{P}}\uparrow^1(\emptyset) = \{\ num(c)\sharp(\mathtt{tt},0),\ \ldots,\ num(c)\sharp(\mathtt{tt},1),\ \ldots\ \}$$
$$\mathsf{T}_{\mathcal{P}}\uparrow^2(\emptyset) = \mathsf{T}_{\mathcal{P}}\uparrow^1(\emptyset) \cup \{\ num(s(c))\sharp(\mathtt{tt},1),\ \ldots,\ num(s(c))\sharp(\mathtt{tt},2),\ \ldots\ \}$$
$$\mathsf{T}_{\mathcal{P}}\uparrow^3(\emptyset) = \mathsf{T}_{\mathcal{P}}\uparrow^2(\emptyset) \cup \{\ num(s(s(c)))\sharp(\mathtt{tt},2),\ \ldots,\ num(s(s(c)))\sharp(\mathtt{tt},3),\ \ldots\ \}$$

Y por la caracterización del modelo mínimo $\mathcal{M}_{\mathcal{P}}$ dada anteriormente, se tiene:

$$\mathcal{M}_{\mathcal{P}} = \left\{ \begin{array}{llll} num(c)\sharp(\mathtt{tt},0), & \ldots, & num(c)\sharp(\mathtt{tt},1), & \ldots, \\ num(s(c))\sharp(\mathtt{tt},1), & \ldots, & num(s(c))\sharp(\mathtt{tt},2), & \ldots, \\ num(s(s(c)))\sharp(\mathtt{tt},2), & \ldots, & num(s(s(c)))\sharp(\mathtt{tt},3), & \ldots, \\ \ldots & & & \\ num(s^k(c))\sharp(\mathtt{tt},k), & \ldots, & num(s^k(c))\sharp(\mathtt{tt},k{+}1), & \ldots, \\ \ldots & & & \end{array} \right\}$$

que coincide con el modelo mínimo esperado para el programa.

2.3.3. Semántica basada en un cálculo lógico

La semántica basada en un cálculo lógico es otra manera de caracterizar los modelos y modelos mínimos de un programa lógico. En este caso, la caracterización viene dada a partir de un sistema de deducción lógica, que denominaremos *lógica de Horn cualificada y bivaluada* sobre $\mathcal{D}$, i.e. BQHL($\mathcal{D}$), que contiene una única regla de inferencia denominada *Modus ponens cualificado y bivaluado*, i.e. BQMP. En la figura 3 puede verse el cálculo lógico BQHL($\mathcal{D}$).

BQMP $\dfrac{B_1\theta\sharp(v_1,d_1) \quad \cdots \quad B_m\theta\sharp(v_m,d_m)}{A\theta\sharp(v,d)}$

si $(A\sharp v \xleftarrow{\alpha} B_1\sharp(v_1,w_1),\ldots,B_m\sharp(v_m,w_m)) \in \mathcal{P}$, θ sust.,
$d_i \trianglerighteq^? w_i \ \ (1 \leq i \leq m)$ y $d \trianglelefteq \alpha \circ \bigsqcap\{d_1,\ldots,d_m\}$.

Figura 3: Cálculo lógico BQHL($\mathcal{D}$)

Nota 4. Téngase en cuenta que el nombre del cálculo lógico presentado en la publicación [60](A.2) es también QHL, y no BQHL como se le ha denominado aquí. Con este cambio de denominación se tiene que QHL denomina, en exclusiva, el cálculo lógico que puede obtenerse eliminando la parte correspondiente a las marcas de los átomos y las restricciones umbral, y que da lugar a la semántica basada en un cálculo lógico del esquema QLP presentado en la publicación [59](A.1).

La notación $\mathcal{P} \vdash_{\mathcal{D}} A\sharp(v, d)$ indica que el átomo cualificado $A\sharp(v, d)$ puede inferirse a partir de las cláusulas del programa $\mathcal{P}$ en un número finito de pasos de inferencia BQMP. Para hacer explícito el número exacto n de inferencias BQMP utilizadas en la demostración, empleamos la notación $\mathcal{P} \vdash^{n}_{\mathcal{D}} A\sharp(v, d)$. Nótese que es usual representar pruebas en BQHL($\mathcal{D}$) como árboles invertidos en el que la raíz contiene el átomo cualificado objetivo de la demostración, y cada nodo contiene un átomo cualificado que se corresponde con una inferencia que utiliza como premisas los hijos del nodo en cuestión.

Como ilustración, considérese el siguiente ejemplo:

Ejemplo 2.7 (Derivación en BQHL)**.** Considérese el programa $\mathcal{P}_{\mathrm{r}}$ del ejemplo 2.3. La siguiente demostración prueba que el átomo cualificado

$$come(h(ana), carne)\sharp(\mathtt{ff}, (0.45, 3))$$

es derivable a partir de las cláusulas de $\mathcal{P}_{\mathrm{r}}$:

$$\dfrac{\dfrac{}{persona(ana)\sharp(\mathtt{tt}, (1, 0))}\ (2) \quad \dfrac{}{come(ana, carne)\sharp(\mathtt{ff}, (0.6, 0))}\ (3)}{come(h(ana), carne)\sharp(\mathtt{ff}, (0.45, 3))}\ (1)$$

donde cada paso de inferencia es como sigue:

(1) Inferencia con cláusula 8 y donde $(0.45, 3) \trianglelefteq (0.8, 1) \circ \bigsqcap\{(1, 0), (0.6, 0)\}$.

(2) Inferencia con cláusula 2 y donde $(1, 0) \trianglelefteq (1, 0) \circ \mathbf{t}$.

(3) Inferencia con cláusula 6 y donde $(0.6, 0) \trianglelefteq (0.6, 0) \circ \mathbf{t}$.

Finalmente, la caracterización del modelo mínimo $\mathcal{M}_{\mathcal{P}}$ de un programa $\mathcal{P}$ dado, a partir del cálculo lógico BQHL($\mathcal{D}$), es

$$\mathcal{M}_{\mathcal{P}} = \{A\sharp(v, d) \mid \mathcal{P} \vdash_{\mathcal{D}} A\sharp(v, d)\}$$

i.e. el conjunto de todos los átomos $A\sharp(v, d)$ que se deducen de las cláusulas del programa $\mathcal{P}$ en el cálculo BQHL($\mathcal{D}$).

2.3.4. Objetivos y soluciones

En el caso de la programación lógica clásica, un objetivo se presenta como la conjunción de una serie de átomos que se espera puedan ser probados a partir del programa. Sin embargo, en el caso que nos ocupa, y dado que los átomos deben probarse para un determinado valor de cualificación, se consideran *átomos anotados abiertos* $A\sharp(v, W)$ donde $v \in \{\mathtt{tt}, \mathtt{ff}\}$ y W es una variable de cualificación, perteneciente a un conjunto de variables $\mathcal{W}ar$, disjunto de la signatura y del conjunto de variables $\mathcal{V}ar$. Además, los objetivos podrán incluir una serie de *restricciones umbral* con la finalidad de limitar, por medio de cotas inferiores, los posibles valores de cualificación con los que se desea probar cada átomo. Estas restricciones umbral son de la forma $W \trianglerighteq^{?} \beta$ donde $W \in \mathcal{W}ar$ y β es: a) un valor de cualificación, i.e. $\beta = d \in D \setminus \{\mathbf{b}\}$, y entonces la restricción debe entenderse como $W \trianglerighteq \beta$; o bien b) $\beta = ?$,

y entonces la restricción es trivialmente cierta, y equivale a no establecer una restricción umbral para el átomo anotado con la variable W.

Formalmente, un objetivo BQLP tiene la forma

$$A_1 \sharp (v_1, W_1),\ \ldots,\ A_m \sharp (v_m, W_m) \,[\!]\, W_1 \trianglerighteq^? \beta_1,\ \ldots,\ W_m \trianglerighteq^? \beta_m$$

que puede abreviarse como $(A_i \sharp (v_i, W_i),\ W_i \trianglerighteq^? \beta_i)_{i=1\ldots m}$. Y una solución para dicho objetivo, será una pareja $\langle \theta, \rho \rangle$ donde θ es una substitución de variables a términos y ρ es una sustitución de variables de cualificación a valores de cualificación, y tal que $\mathcal{P} \vdash_{\mathcal{D}} A_i\theta \sharp (v_i, W_i\rho)$ con $W_i\rho \trianglerighteq^? \beta_i$ para $i = 1 \ldots m$. Es decir, que pueden probarse, a partir del programa $\mathcal{P}$, todos los átomos del objetivo una vez instanciados por θ y ρ, y ρ asigna valores de cualificación aceptables a las variables de cualificación del objetivo — entendiendo por aceptables aquellos valores que satisfagan la restricción umbral impuesta en el objetivo y que permitan probar el átomo cualificado a partir del programa.

Nota 5. La noción de solución de un objetivo BQLP aquí presentada difiere de la realizada en [60](A.2, Def. 4.2) — y en [59](A.1, Def. 2) para objetivos QLP. El motivo es que la noción que se presenta aquí resulta independiente del procedimiento de resolución de objetivos escogido, y se acerca más a la noción declarativa de solución dada en los trabajos posteriores. En cualquier caso, es fácil ver que la noción de solución tal y como aquí se presenta resulta equivalente a la dada en aquella definición, si se asume que el objetivo BQLP es inicial, en el sentido de [60](A.2, Def. 4.1).

El siguiente ejemplo ilustra la noción de objetivo y solución.

Ejemplo 2.8 (Objetivo y solución)**.** Siguiendo con el programa $\mathcal{P}_\mathsf{r}$ del ejemplo 2.3, un posible objetivo G_r para preguntar qué alimentos no comería un hijo de *ana* con al menos una cualificación de $(0.45, 3)$, es:

$$come(h(ana), A) \sharp (\mathtt{ff}, W) \,[\!]\, W \trianglerighteq^? (0.45, 3)$$

y una posible solución válida para G_r con respecto a $\mathcal{P}_\mathsf{r}$ es:

$$\langle \{A \mapsto carne\}, \{W \mapsto (0.48, 1)\} \rangle$$

que satisface ambas condiciones para ser solución porque $(0.48, 1) \trianglerighteq^? (0.45, 3)$ y $\mathcal{P}_\mathsf{r} \vdash_{\mathcal{U}\otimes\mathcal{W}} come(h(ana), carne) \sharp (\mathtt{ff}, (0.48, 1))$.

2.4. Resolución de objetivos

Para resolver BQLP-objetivos se presenta, a continuación, un procedimiento paramétrico de resolución de objetivos inspirado en la resolución SLD clásica que llamaremos BQSLD($\mathcal{D}$).

Nota 6. Este procedimiento de resolución se presentó originalmente en [59](A.1, §4) como "SLD($\mathcal{D}$)" para programas y objetivos QLP; y en [60](A.2, §4) como "QSLD($\mathcal{D}$)" para programas y objetivos BQLP. Este último es el que se presenta en esta sección con el nombre de "BQSLD($\mathcal{D}$)" por considerarlo más acertado.

La resolución BQSLD($\mathcal{D}$) parte de un BQLP-objetivo, al que nos referiremos como *objetivo inicial*, y procede a través de una serie de *objetivos intermedios* mediante *pasos de resolución* hasta alcanzar un *objetivo final* del que se podrá extraer una *solución asociada*. Por tanto, resulta conveniente extender ligeramente la sintaxis dada para un objetivo con la finalidad de poder representar el estado intermedio de una computación de resolución. Así, diremos ahora que un BQLP-objetivo es de la forma:

$$\overline{A} \parallel \sigma \parallel \Delta$$

donde $\overline{A}$ es una conjunción de átomos anotados abiertos, σ es una sustitución de variables a términos y Δ una conjunción de *restricciones de cualificación*. Adicionalmente, Δ deberá cumplir una serie de condiciones de admisibilidad, detalladas en [60](A.2, Def. 4.1), para garantizar la integridad del procedimiento de resolución.

Las *restricciones de cualificación* que pueden aparecer en Δ, son de dos tipos: o bien son de la forma "$W \geqslant^{?} \beta$", y entonces decimos que se trata de una *restricción umbral para la variable W*; o bien son de la forma "$W = \alpha \circ \prod\{W_1, \ \ldots, \ W_m\}$", y entonces decimos que se trata de una *restricción definitoria para la variable W*.

Las restricciones umbral tienen la finalidad de acotar inferiormente los valores de cualificación con los que podrán probarse los átomos del objetivo, y vendrán determinadas por el usuario en el objetivo inicial según sean sus expectativas o intereses, y por el programa según sean los cuerpos de las cláusulas empleadas en la resolución. Por otro lado, las restricciones definitorias tienen la finalidad de determinar cómo debe calcularse el valor de una variable de cualificación en particular —la que define— y esta restricción dependerá de la cláusula que haya sido empleada en el paso específico de resolución en el que se introdujo, ya que esta determinará el factor de atenuación α a aplicar y el número de variables de cualificación nuevas de las que dependerá.

Concretando un poco más la forma de los objetivos:

- Un *objetivo inicial* será de la forma $\overline{A} \parallel \varepsilon \parallel \Delta$ (o sencillamente $\overline{A} \parallel \Delta$), donde ε es la sustitución vacía y Δ contiene únicamente restricciones umbral para las variables de cualificación que aparecen en $\overline{A}$.

- Un *objetivo intermedio* será de la forma $\overline{A} \parallel \sigma_1\sigma_2 \cdots \sigma_l \parallel \Delta$, donde $\sigma_1\sigma_2 \cdots \sigma_l$ corresponde a las sustituciones aplicadas a $\overline{A}$ y calculadas en cada uno de los l pasos de resolución dados hasta el momento; y Δ contiene tanto restricciones umbral como restricciones definitorias.

- Un *objetivo final* será de la forma $\parallel \sigma_1\sigma_2 \cdots \sigma_l \parallel \Delta$ (o sencillamente $\sigma_1\sigma_2 \cdots \sigma_l \parallel \Delta$), donde la conjunción de átomos es vacía y Δ sólo contiene restricciones definitorias. Cabe aquí destacar que todo objetivo final $\sigma \parallel \Delta$ tiene una *solución asociada* $\langle \theta, \rho \rangle$ en la que $\theta = \sigma$ y ρ asigna valores de cualificación a variables de cualificación según indica el conjunto de restricciones definitorias Δ. La construcción de ρ está garantizada por las condiciones adicionales de admisibilidad expuestas en [60](A.2, Def. 4.1) y que no se detallan en esta exposición.

De la forma de los objetivos se extrae que los pasos de resolución deberán encargarse de dos elementos principales: a) obtener las sustituciones σ_i que unifican cada átomo de la conjunción $\overline{A}$ seleccionado en el paso de resolución con la cabeza de una cláusula del programa;

y b) actualizar las restricciones umbral para cada una de las variables de cualificación que aparecen en la conjunción $\overline{A}$ por su correspondiente restricción definitoria determinada por la cláusula empleada en el paso de resolución.

La idea general consiste, por tanto, en que dados un programa $\mathcal{P}$ y un objetivo G, un *paso de resolución* haga lo siguiente:

1. Seleccionar un átomo anotado abierto del objetivo para el paso de resolución — sin presuponer ninguna función de selección determinada.
2. Obtener un u.m.g. σ entre el átomo seleccionado y la cabeza de una cláusula C del programa $\mathcal{P}$. Esta cláusula será la seleccionada para dar el paso de resolución.
3. Añadir al objetivo los átomos del cuerpo de la cláusula C, anotándolos con variables de cualificación nuevas.
4. Aplicar σ al objetivo resultante, tanto a la conjunción de átomos como a la sustitución del objetivo.
5. Variar el conjunto de restricciones de cualificación de forma que se añadan restricciones umbral para cada una de las variables de cualificación nuevas introducidas en el objetivo, y se sustituya la restricción umbral de la variable de cualificación del átomo seleccionado por la correspondiente restricción definitoria, utilizando para ello el factor de atenuación α de C, y las variables de cualificación nuevas introducidas para cada uno de los átomos del cuerpo de C.

Procediendo de forma reiterada, y suponiendo que el cómputo termine, se llega hasta que: o bien no puede darse ningún paso de resolución adicional y falla el cómputo con respecto al programa; o bien no queden átomos en el objetivo y haya podido probarse con respecto al programa para una solución $\langle \theta, \rho \rangle$ que se obtiene como solución asociada al objetivo final de la computación.

De manera más formal se escribirá

$$G_0 \Vdash_{C_1,\sigma_1} G_1 \Vdash_{C_2,\sigma_2} \cdots \Vdash_{C_n,\sigma_n} G_n$$

para indicar una computación en n pasos de resolución desde el objetivo inicial G_0 hasta el objetivo final G_n, y donde casa paso de resolución tiene la forma $G \Vdash_{C',\sigma'} G'$ con

$$\begin{array}{l} G : \overline{L},\ A\sharp(v,W),\ \overline{R} \,[\!]\, \sigma \,[\!]\, W \trianglerighteq^? \beta,\ \Delta \\ G' : (\overline{L}, (B_i\sharp(v_i,W_i))_{i=1\ldots m}, \overline{R})\sigma' \,[\!]\, \sigma\sigma' \,[\!]\, (\ W_i \trianglerighteq^? \beta_i\ )_{i=1\ldots m},\ W = \alpha \circ \textstyle\prod_{i=1}^m W_i,\ \Delta \end{array}$$

cuando

- $A\sharp(v,W)$ es el átomo seleccionado;
- $C' : H\sharp v \overset{\alpha}{\leftarrow} B_1\sharp(v_1,w_1),\ \ldots,\ B_m\sharp(v_m,w_m)$ es una variante de una cláusula del programa $\mathcal{P}$ con variables nuevas y tal que $\alpha \trianglerighteq^? \beta$;
- σ' es un u.m.g. entre A y H;
- $W_1,\ \ldots,\ W_m$ son variables de cualificación nuevas;

- y los umbrales β_i para las variables de cualificación W_i se calculan en función del umbral β para W, del factor de atenuación α de C' y del umbral w_i del átomo correspondiente del cuerpo de C'.

Nota 7. En este momento es donde se hace patente la diferencia en la definición presentada de los dominios de cualificación en [59](A.1, §2) y en [60](A.2, §2). La presentación de los dominios de cualificación en este último incluye una función $\oslash$ que se define como la inversa de $\circ$ (o de $\otimes$ como se denomina por coherencia con $\oslash$ en en dicho trabajo), y que es la que nos permite calcular el valor final de los umbrales β_i mediante la función *newThreshold*(β, α, w_i) que aparece en [60](A.2, Def. 4.4) y que aquí se ha omitido por ser la definición de la función $\oslash$ un detalle técnico que no ha tenido continuidad en la presentación axiomática de los dominios de cualificación utilizada en posteriores publicaciones.

El siguiente ejemplo muestra una posible computación por resolución del BQLP-objetivo $come(h(ana), A)\sharp(\mathtt{ff}, W) \,[\!]\, W \trianglerighteq^{?} (0.4, 3)$ para el programa $\mathcal{P}_{\mathsf{r}}$ del ejemplo 2.3.

Ejemplo 2.9 (Resolución de objetivos)**.** Sea $\mathcal{P}_{\mathsf{r}}$ el programa del ejemplo 2.3 y G_{r} el objetivo del ejemplo 2.8. Entonces:

G_0 $come(h(ana), A)\sharp(\mathtt{ff}, W) \,[\!]\, W \trianglerighteq^{?} (0.4, 3)$ $\Vdash_{8, \{ X \mapsto ana,\ A \mapsto Y \}}$

G_1 $persona(ana)\sharp(\mathtt{tt}, W_1),\ come(ana, Y)\sharp(\mathtt{ff}, W_2)$
$\quad [\!]\ \{ X \mapsto ana,\ A \mapsto Y \}$
$\quad [\!]\ W_1 \trianglerighteq^{?} (0.5, 2),\ W_2 \trianglerighteq^{?} (0.5, 2),\ W = (0.8, 1) \circ \prod\{W_1, W_2\}$ $\Vdash_{2, \varepsilon}$

G_2 $come(ana, Y)\sharp(\mathtt{ff}, W_2)$
$\quad [\!]\ \{X \mapsto ana,\ A \mapsto Y\}\ \varepsilon$
$\quad [\!]\ W_1 = (1, 0) \circ \mathbf{t},\ W_2 \trianglerighteq^{?} (0.5, 2),\ W = (0.8, 1) \circ \prod\{W_1, W_2\}$ $\Vdash_{6, \{ Y \mapsto carne \}}$

G_3 $\{ X \mapsto ana,\ A \mapsto Y \}\ \varepsilon\ \{ Y \mapsto carne \}$
$\quad [\!]\ W_1 = (1, 0),\ W_2 = (0.6, 0),\ W = (0.48, 1)$

y $\langle \{A \mapsto carne\}, \{W \mapsto (0.48, 1)\} \rangle$ es la restricción a las variables de G_{r} de la solución asociada al objetivo final G_3 de la resolución.

Nota 8. Nótese que las restricciones umbral $W_1 \trianglerighteq^{?} (0.5, 2)$ y $W_2 \trianglerighteq^{?} (0.5, 2)$ del objetivo intermedio G_1 surgen de la utilización de la función *newThreshold* según se define en [60](A.2, Def. 4.4) mediante el cálculo:

$$newThreshold((0.4, 3), (0.8, 1), ?) = (0.4, 3) \oslash (0.8, 1) = (0.5, 2)$$

3. Programación lógica con cualificación y similaridad

Como se ha visto hasta ahora, la cualificación supone una generalización de la noción de incertidumbre empleada en la programación lógica cuantitativa o anotada, permitiendo, de hecho, modelar programas lógicos que hagan uso de la incertidumbre en sus razonamientos.

La *programación lógica con similaridad* (SLP) surge con la misma finalidad que la programación lógica cuantitativa y, evidentemente, también que la programación lógica cualificada: permitir la utilización de la incertidumbre en los programas lógicos. Sin embargo, su aproximación difiere en la forma en la que se emplea esta incertidumbre como parte del razonamiento lógico. Si bien en la programación lógica cualificada la incertidumbre (o la cualificación) interviene directamente en el grado con el que confiamos en la veracidad de la implicación de las cláusulas lógicas; en la programación lógica con similaridad, la incertidumbre es la que permite que algunos símbolos del programa jueguen el papel de otros símbolos *similares* según la *relación de similaridad* empleada en el programa. Por lo tanto, en los esquemas de programación lógica basada en similaridad, la incertidumbre juega el papel de permitir probar átomos similares, según la relación de similaridad considerada, a aquellos que resultan consecuencia lógica, en el sentido clásico, de un programa lógico cualquiera.

En esta sección, se extiende con relaciones de similaridad al estilo de [64] el esquema QLP objeto de la sección anterior —i.e. BQLP sin predicados bivaluados ni restricciones umbrales en los cuerpos de las cláusulas—, obteniendo como resultado el esquema general SQLP de la programación lógica con cualificación y similaridad con la expresividad necesaria para escribir programas lógicos que hagan uso de la cualificación y la similaridad al mismo tiempo. Asimismo, y en lugar de proponer una semántica operacional para programas SQLP, se propone un mecanismo de transformación de programas SQLP a programas QLP equivalentes que puedan hacer uso del procedimiento de resolución de objetivos descrito en la subsección 2.4 de la sección anterior. En particular, estos resultados demuestran que los SLP-programas pueden ser, en efecto, reducidos a QLP-programas equivalentes.

3.1. Relaciones de similaridad

Las relaciones de similaridad sobre un conjunto de elementos S fueron definidas en [64] y otros trabajos relacionados como funciones $\mathcal{S} : S \times S \to [0, 1]$ que satisfacen tres axiomas análogos a los requeridos en las relaciones de equivalencia clásicas — i.e. reflexividad, simetría y transitividad. Cada valor $\mathcal{S}(x, y)$ computado por la relación de similaridad $\mathcal{S}$ se denomina *grado de similaridad* entre x e y. En nuestro caso, se utiliza una extensión natural de la definición dada en [64] para permitir que elementos arbitrarios de un dominio de cualificación $\mathcal{D}$ sirvan como grados de similaridad. Así, se tiene que una relación de similaridad puede establecer el grado de certidumbre sobre el parecido de dos símbolos (si esta diera valores en el dominio de cualificación $\mathcal{U}$), el precio a pagar para que un símbolo juegue el papel de otro en una demostración (si esta diera valores en el dominio de cualificación $\mathcal{W}$), etc. De la misma manera que en [64], se estará interesado en relaciones de similaridad sobre un conjunto S cuyos elementos sean variables y símbolos de la signatura de un programa dado.

Nota 9. En la publicación [11](A.3) las relaciones de similaridad se representaban con la notación $\mathcal{R}$ y no con $\mathcal{S}$. Aquí se ha optado por utilizar $\mathcal{S}$ —al igual que se hace en las publicaciones posteriores— para reservar la notación $\mathcal{R}$ para el dominio de restricciones reales que se empleará en las próximas secciones.

Formalmente, dados un dominio de cualificación $\mathcal{D}$ con conjunto soporte D y un conjunto S, se define una relación de similaridad *$\mathcal{D}$-valuada* sobre el conjunto S como cualquier

función $\mathcal{S} : S \times S \to D$ tal que los tres axiomas siguientes se satisfacen para todo x, y, z del conjunto S:

- **Reflexividad.** $\mathcal{S}(x, x) = \mathbf{t}$.
- **Simetría.** $\mathcal{S}(x, y) = \mathcal{S}(y, x)$.
- **Transitividad.** $\mathcal{S}(x, z) \trianglerighteq \mathcal{S}(x, y) \sqcap \mathcal{S}(y, z)$.

Como consecuencia de esta definición puede trivialmente definirse una relación de similaridad $\mathcal{S}_{\text{id}}$ especial, la *identidad*, que cumple: (1) $\forall x \in S$, $\mathcal{S}_{\text{id}}(x, x) = \mathbf{t}$; y (2) $\forall x, y \in S$ tal que $x \neq y$, $\mathcal{S}_{\text{id}}(x, y) = \mathbf{b}$.

A modo de ilustración, considérese la siguiente relación de similaridad:

Ejemplo 3.1 (Relación de similaridad)**.** Sean *roble*, *haya*, *castaño*, *mármol* y *granito* cinco constantes de la signatura que representan diferentes materiales. Entonces, la relación de similaridad $\mathcal{S}$ definida por las siguientes ecuaciones:

$$\begin{aligned}
\mathcal{S}(\textit{roble}, \textit{roble}) &= 1 & \mathcal{S}(\textit{roble}, \textit{haya}) &= 0.85 = \mathcal{S}(\textit{haya}, \textit{roble}) \\
\mathcal{S}(\textit{haya}, \textit{haya}) &= 1 & \mathcal{S}(\textit{roble}, \textit{castaño}) &= 0.75 = \mathcal{S}(\textit{castaño}, \textit{roble}) \\
\mathcal{S}(\textit{castaño}, \textit{castaño}) &= 1 & \mathcal{S}(\textit{haya}, \textit{castaño}) &= 0.75 = \mathcal{S}(\textit{castaño}, \textit{haya}) \\
\mathcal{S}(\textit{mármol}, \textit{mármol}) &= 1 & & \\
\mathcal{S}(\textit{granito}, \textit{granito}) &= 1 & \mathcal{S}(\textit{mármol}, \textit{granito}) &= 0.85 = \mathcal{S}(\textit{granito}, \textit{mármol})
\end{aligned}$$

es una relación de similaridad $\mathcal{U}$-valuada que satisface los tres axiomas de reflexividad, simetría y transitividad. Nótese que entre las parejas de símbolos no especificadas se tiene un grado de similaridad de 0; i.e. $\mathcal{S}(\textit{roble}, \textit{granito}) = 0$, etc.

En adelante, las relaciones de similaridad se especificarán mediante un conjunto de ecuaciones sin ecuaciones reflexivas y no necesariamente cerrado bajo simetría ni transitividad, por lo que la relación de similaridad definida será en realidad la que resultase del cierre reflexivo, simétrico y transitivo del conjunto de ecuaciones dadas.

Hasta al momento, una relación de similaridad ha sido definida como una función tal que dados dos elementos de un conjunto S, correspondiente al conjunto de variables y símbolos de un programa, les asigna un valor de cualificación de un dominio $\mathcal{D}$ determinado, y por el que se dice que la relación de similaridad es $\mathcal{D}$-valuada. Sin embargo, cumpliendo lo anterior es posible definir relaciones de similaridad cuya utilidad en un marco de programación lógica fuera bastante discutible, es decir, ¿qué cabría esperar de una relación de similaridad de la que pudiera concluirse que un determinado símbolo, e.g. una constante, fuera parecido a otro de distinta naturaleza, e.g. un símbolo de predicado? Por esta razón, se hace necesario establecer unas condiciones adicionales de *admisibilidad* para que una relación de similaridad pueda formar parte de una instancia del esquema SQLP.

En esencia, diremos que una relación $\mathcal{S}$ de similaridad $\mathcal{D}$-valuada es *admisible* sii el conjunto S es la unión del conjunto de variables y los conjuntos de símbolos de constructora y de predicado, y además:

1. La restricción de $\mathcal{S}$ al conjunto de las variables se comporta como la identidad — i.e. $\mathcal{S}(X, X) = \mathbf{t}$ para toda variable X, y $\mathcal{S}(X, Y) = \mathbf{b}$ cuando $X \neq Y$.

2. $\mathcal{S}(x, y) \neq \mathbf{b}$ se cumple únicamente si x e y son: a) la misma variable; b) símbolos de constructora de igual aridad; o c) símbolos de predicado de igual aridad.

Por último, se asume de ahora en adelante que es posible extender cualquier relación admisible de similaridad $\mathcal{S}$ para actuar sobre términos y átomos. La extensión, que se denominará también $\mathcal{S}$, puede definirse de manera recursiva como en [64], y tal y como se hace para el caso de los términos en [11](A.3, Def. 3). Básicamente, esta extensión calcula el ínfimo de los grados de similaridad de cada símbolo de un término con respecto al símbolo del otro término que ocupa la misma posición. El siguiente ejemplo ilustra esta extensión:

Ejemplo 3.2 (Extensión de una relación de similaridad)**.** Dada una constructora unaria m, y la relación de similaridad $\mathcal{U}$-valuada del ejemplo 3.1, se tiene:

$$\begin{aligned} \mathcal{S}(m(\mathit{roble}), m(\mathit{haya})) &= \mathcal{S}(m, m) \sqcap \mathcal{S}(\mathit{roble}, \mathit{haya}) \\ &= 1 \sqcap 0.85 \\ &= 0.85 \end{aligned}$$

Por lo que $m(\mathit{roble})$ y $m(\mathit{haya})$ son términos similares con grado 0.85.

Para el caso de los átomos, así como de cualquier otro elemento estructurado, se procedería de manera análoga.

3.2. Sintaxis

El esquema SQLP se define también como un esquema paramétrico de programación lógica con dos parámetros: un dominio de cualificación $\mathcal{D}$ arbitrario y una relación $\mathcal{S}$ de similaridad $\mathcal{D}$-valuada admisible. La instancia del esquema SQLP para la pareja $\langle \mathcal{S}, \mathcal{D} \rangle$ será SQLP($\mathcal{S}, \mathcal{D}$), y por tanto SQLP($\mathcal{S}, \mathcal{U}$) es la instancia para valores de incertidumbre (con $\mathcal{S}$ relación de similaridad $\mathcal{U}$-valuada), que puede verse también como SLP con cualificación; SQLP($\mathcal{S}, \mathcal{W}$) es la instancia para costes valorados de la profundidad de la prueba (con $\mathcal{S}$ relación de similaridad $\mathcal{W}$-valuada); y así sucesivamente para otros dominios de cualificación cualesquiera. Nótese que el esquema SQLP subsume QLP cuando se toma $\mathcal{S} = \mathcal{S}_{\mathsf{id}}$, la identidad; y también subsume la programación lógica basada en similaridad al comportarse como las aproximaciones descritas en [64], y otros trabajos relacionados, cuando se toma $\mathcal{D} = \mathcal{U}$ y se usan exclusivamente cláusulas con factor de atenuación 1.

A nivel sintáctico, SQLP presenta muy pocas novedades con respecto a QLP. Las diferencias se harán patentes en la definición de la semántica declarativa que se verá en la siguiente subsección, ya que esta deberá ser extendida de manera adecuada para que se tenga en cuenta la relación de similaridad dada. En definitiva, la sintaxis de un SQLP-programas es bastante sencilla: se trata de QLP-programas ampliados con una relación de similaridad que permitirá a unos símbolos jugar el papel de otros *pagando un coste*, e.g. disminuyendo la certidumbre del átomo probado o aumentando el coste mínimo de la prueba. Un programa para la instancia SQLP($\mathcal{S}, \mathcal{U}$) es el que muestra el siguiente ejemplo.

Ejemplo 3.3 (SQLP: Piezas de roble)**.** Supóngase que disponemos de una serie de piezas de diferentes materiales identificadas por un número. Una representación válida de esta serie de piezas es la colección de hechos que sigue:

c_1 $\quad pieza(1, granito) \overset{1}{\leftarrow}$
c_2 $\quad pieza(2, roble) \overset{1}{\leftarrow}$
c_3 $\quad pieza(3, haya) \overset{1}{\leftarrow}$
c_4 $\quad pieza(4, castaño) \overset{1}{\leftarrow}$
c_5 $\quad pieza(5, mármol) \overset{1}{\leftarrow}$

Construimos ahora un predicado capaz de seleccionar las piezas de *roble* y dar, uno a uno, sus identificadores:

c_6 $\quad sel\_roble(X) \overset{1}{\leftarrow} pieza(X, roble)$

Asumiendo además la relación de similaridad $\mathcal{S}$ del ejemplo 3.1, un usuario esperaría obtener, para el objetivo

$$sel\_roble(Pieza) \sharp W \, [] \, W \geq^{?} 0.8$$

las piezas 2 y 3. El caso de la pieza 2 es claro por ser una pieza de *roble*. El de la pieza 3, es debido a que el límite de certidumbre establecido por el objetivo es inferior al grado de similaridad entre los materiales *roble* y *haya*, por lo que a un nivel de certidumbre suficiente (en este caso mayor o igual que 0.8), es posible utilizar el material *haya* en lugar del material *roble*. Por este motivo, la pieza 3 debe ser también una solución válida al objetivo planteado. Nótese que la pieza 4, aún siendo de un material similar al *roble*, no lo es a un grado suficiente (al menos 0.8) como para poder jugar el papel del *roble* en este caso.

La diferencia entre la solución 2 y la solución 3 vendrá dada por el valor máximo de la variable de cualificación W en cada caso: mientras que para la pieza 2 tendríamos que $W \mapsto 1$; para la pieza 3, $W \mapsto 0.85$.

Como es fácil observar, el ejemplo anterior no saca partido de la cualificación, ya que los factores de atenuación de las seis cláusulas del programa coinciden con el máximo en el dominio de cualificación $\mathcal{U}$. Así que, el ejemplo anterior podría considerarse como un SLP-programa si se sustituyen las implicaciones por "$\leftarrow$", al estilo de los programas de [64].

3.3. Semántica declarativa

La semántica declarativa del esquema SQLP se basa en una extensión de la semántica de QLP para el tratamiento adecuado de las relaciones de similaridad. Nótese que en esta sección partiremos de la semántica del esquema QLP, y no de la del esquema BQLP, que fue la realmente desarrollada en la sección 2, por lo que las nociones de interpretación, consecuencia inmediata y modelo, y la caracterización del modelo mínimo, se harán a partir de la simplificación de las presentadas en dicha sección una vez eliminada la parte correspondiente a la bivaluación de los predicados y las restricciones umbrales de los cuerpos de las cláusulas.

3.3.1. Interpretaciones y modelos

Antes de comenzar con la definición de las interpretaciones y modelos en el esquema SQLP, resulta conveniente exponer un problema que surge precisamente por la utilización de relaciones de similaridad. Para situarnos, considérese la relación de similaridad $\mathcal{S}$ del ejemplo 3.1 y el átomo $d(X, X)$ donde d es una predicado binario de la signatura. Una instanciación cualquiera de dicho átomo, daría lugar a átomos que son: o bien de la forma $d(Y, Y)$ donde Y es una variable del conjunto de variables; o más en general de la forma $d(t, t)$ donde t es un término construido de acuerdo con la signatura. Sin embargo, y dado que ahora nos encontramos bajo el influjo una relación de similaridad, será interesante poder instanciar el átomo $d(X, X)$ de manera que los dos argumentos de d fueran no necesariamente iguales, sino similares de acuerdo con la relación de similaridad $\mathcal{S}$ empleada. Para ello, esperaríamos que tanto $d(\mathit{roble}, \mathit{roble})$ como $d(\mathit{roble}, \mathit{haya})$ fueran "instancias similares" del átomo $d(X, X)$ —y este último porque $\mathcal{S}(\mathit{roble}, \mathit{haya}) \neq \mathbf{b}$—, y por lo tanto instancias aceptables para el átomo $d(X, X)$. Para conseguir esto, tenemos dos alternativas:

1. variar la noción clásica de sustitución de variables por términos, de modo que se permitiera instanciar una misma variable con términos construidos de una clase de equivalencia inducida por la relación de similaridad; o bien

2. *linealizar* el átomo estableciendo restricciones adicionales sobre la manera en que el átomo lineal resultante debiera ser instanciado.

Mientras que la primera opción parece mejor desde el punto de vista de la definición formal de un esquema de programación basado en similaridad, resulta más compleja de implementar. El motivo es que los sistemas `Prolog` disponibles implementan, obviamente, la noción estándar de sustitución y para conseguir el efecto deseado, se necesitaría implementar un sistema completo desde el principio, cosa que no resulta trivial si se espera conseguir un rendimiento comparable al de los sistemas `Prolog` existentes. Por este motivo, se desarrollará la segunda opción, de manera que se conseguirá obtener átomos equivalentes al original pero que permitan, realizando una instanciación clásica condicionada —dado que deberá cumplir condiciones adicionales—, obtener todos los átomos que son "instancias similares" del original. Así, dado un átomo A, diremos que

$$\mathrm{lin}(A) = (A_\ell, \mathcal{S}_\ell)$$

es su versión *lineal*, en la que:

- A_ℓ es un átomo lineal; y
- $\mathcal{S}_\ell$ es un conjunto de ecuaciones de la forma $X \sim X_i$ que indican que el término que resulte de la instanciación de la variable X debe ser similar, según la relación de similaridad $\mathcal{S}$, al término que resulte de la instanciación de la variable X_i.

La manera de construir la versión lineal de un átomo A cualquiera es la siguiente: A_ℓ se construye sustituyendo las n apariciones adicionales de una variable X cualquiera por variables nuevas X_i (con $1 \leq i \leq n$); y $\mathcal{S}_\ell$ es el conjunto de *ecuaciones de similaridad* $X \sim X_i$ (con $1 \leq i \leq n$) condicionando las variables de A_ℓ a que tomen valores similares en los casos en que correspondieran a la misma variable X del átomo A. Un ejemplo de

este procedimiento es el siguiente. Nótese, en particular, qué ocurre cuando se tiene que el átomo original es lineal.

Ejemplo 3.4 (Linealización)**.** Sean $H_1 = p(c(X), Y)$ y $H_2 = p(c(X), X, Y)$ dos átomos. Entonces:

- $\mathrm{lin}(H_1) = (\ p(c(X), Y),\ \{\}\ )$.
- $\mathrm{lin}(H_2) = (\ p(c(X), X_1, Y),\ \{X \sim X_1\}\ )$.

A partir de la linealización $\mathrm{lin}(A) = (A_\ell, \mathcal{S}_\ell)$ de un átomo A cualquiera, resulta sencillo definir cuáles son las "instancias similares" a nivel δ de A. Se dirá que A' es una *instancia similar a nivel* δ de A, o más formalmente que A' *es una* $\mathcal{S}$*-instancia a nivel* δ *de* A, en símbolos $(A', \delta) \in [A]_{\mathcal{S}}$, cuando A' sea una instancia de un átomo similar —con grado de similaridad δ diferente de **b**— a A_ℓ con una sustitución que satisfaga el conjunto de ecuaciones de similaridad $\mathcal{S}_\ell$ a nivel δ. Para la definición formal de $\mathcal{S}$-instancia de un átomo, véase [11](A.3, Def. 4).

Nota 10. *En la publicación [11](A.3) se denominaba* $\mathcal{R}$-instancia de un átomo *a las* $\mathcal{S}$*-instancias de un átomo.*

Una vez explicada la razón de por qué es necesario linealizar, y definida la noción de $\mathcal{S}$-instancia de un átomo, puede continuarse con la definición de interpretación en el esquema SQLP. Las interpretaciones en SQLP se definen también como conjuntos de átomos cualificados cerrados, esta vez, bajo una relación denominada $\langle \mathcal{S}, \mathcal{D} \rangle$*-implicación*, cuya finalidad es la de asegurar que si una interpretación $\mathcal{I}$ incluye un átomo cualificado $A \sharp d$, entonces incluya también todos aquellos átomos cualificados que sean $\mathcal{S}$-instancias de $A \sharp d$. Formalmente, se dice que un átomo cualificado $A \sharp d$ $\langle \mathcal{S}, \mathcal{D} \rangle$-implica de otro átomo cualificado $A' \sharp d'$, en símbolos $A \sharp d \succcurlyeq_{\mathcal{S},\mathcal{D}} A' \sharp d'$, cuando ocurra $(A', \delta) \in [A]_{\mathcal{S}}$ y $d' \trianglelefteq \delta \circ d$ para algún grado de similaridad δ.

Para la definición de la noción de *modelo* de un SQLP-programa $\mathcal{P}$, se empleará también una noción auxiliar parecida a la de $\mathcal{S}$-instancia de un átomo, la $\mathcal{S}$*-instancia de una cláusula*. Esta noción permitirá unificar átomos con cabezas de cláusulas del programa $\mathcal{P}$ siempre que el átomo sea una $\mathcal{S}$-instancia de la cabeza de una cláusula, y no sólo la instancia como en el caso de la programación lógica clásica.

Se dirá que una cláusula C' es una $\mathcal{S}$*-instancia a nivel* δ de otra cláusula C de la forma $A \overset{\alpha}{\leftarrow} B_1,\ \ldots,\ B_m$ de $\mathcal{P}$, en símbolos $(C', \delta) \in [C]_{\mathcal{S}}$, cuando A' sea una $\mathcal{S}$-instancia —con sustitución θ— a nivel δ de A y C' sea de la forma $A' \overset{\alpha}{\leftarrow} B_1\theta,\ \ldots,\ B_m\theta$. Para la definición formal de $\mathcal{S}$-instancia de una cláusula, véase [11](A.3, Def. 5).

Nota 11. En la publicación [11](A.3) se denominaba $\mathcal{R}$*-instancia de una cláusula* a las $\mathcal{S}$-instancias de una cláusula.

Finalmente, dado un SQLP-programa $\mathcal{P}$, se dice que una interpretación $\mathcal{I}$ es *modelo de una cláusula* C de $\mathcal{P}$, sii para toda $\mathcal{S}$-instancia $C' : A' \overset{\alpha}{\leftarrow} B'_1,\ \ldots,\ B'_m$ a nivel δ de C y

para cualesquiera valores de cualificación $d_1, \ldots, d_m$ tales que $B'_i \sharp d_i \in \mathcal{I}$ con $1 \leq i \leq m$, entonces $A' \sharp d \in \mathcal{I}$ donde $d \trianglelefteq \alpha \circ \bigsqcap \{\delta,\ d_1,\ \ldots,\ d_m\}$. Y se dice que $\mathcal{I}$ es *modelo de* $\mathcal{P}$ sii $\mathcal{I}$ es modelo de toda cláusula C de $\mathcal{P}$.

3.3.2. Semántica basada en un cálculo lógico

Como ya se ha visto para el caso de la semántica declarativa del esquema QLP, existen dos posibles alternativas para caracterizar el modelo mínimo de un programa SQLP, bien mediante un transformador de interpretaciones, bien mediante un cálculo de inferencia lógica. Para este caso, y siguiendo la propuesta de la publicación [11](A.3), se propone únicamente una semántica basada en un cálculo de inferencia lógica, dado que ambas aproximaciones producen caracterizaciones equivalentes, y la extensión del transformador de interpretaciones presentado en la subección 2.3.2 resulta claramente sencilla.

Se define, por tanto, un cálculo lógico denominado *lógica de Horn cualificada basada en similaridad* sobre $\langle \mathcal{S}, \mathcal{D} \rangle$ —i.e. SQHL$(\mathcal{S}, \mathcal{D})$— compuesto también en este caso por una única regla de inferencia denominada *Modus Ponens cualificado basado en similaridad* —i.e. SQMP— tal y como puede verse en la figura 4.

SQMP $\dfrac{B'_1 \sharp d_1 \quad \cdots \quad B'_m \sharp d_m}{A' \sharp d}$

si $(A' \xleftarrow{\alpha} B'_1, \ldots, B'_m, \delta) \in [C]_{\mathcal{S}}$ para alguna cláusula $C \in \mathcal{P}$
y $d \trianglelefteq \alpha \circ \bigsqcap \{\delta,\ d_1,\ \ldots,\ d_m\}$.

Figura 4: Cálculo lógico SQHL$(\mathcal{S}, \mathcal{D})$

De manera similar a como se hizo para el esquema QLP, la notación $\mathcal{P} \vdash_{\mathcal{S},\mathcal{D}} A \sharp d$ indica que el átomo cualificado $A \sharp d$ puede inferirse a partir de las cláusulas de $\mathcal{P}$ en un número finito de pasos de inferencia. En el caso de querer hacer explícito el número n de pasos de inferencia, se escribirá $\mathcal{P} \vdash^n_{\mathcal{S},\mathcal{D}} A \sharp d$. Nótese la diferencia en la notación $\vdash_{\mathcal{S},\mathcal{D}}$ con respecto a la notación $\vdash_{\mathcal{D}}$ correspondiente a la inferencia QHL (y también BQHL) del esquema QLP (o BQLP).

Como ilustración del proceso de inferencia lógica en el cálculo SQHL$(\mathcal{S}, \mathcal{D})$ considérese el siguiente ejemplo.

Ejemplo 3.5 (Derivación en SQHL)**.** Sea $\mathcal{S}$ la relación de similaridad del ejemplo 3.1, y sea $\mathcal{D}$ el dominio de cualificación $\mathcal{U}$ de los valores de incertidumbre. Entonces el átomo $sel\_roble(3) \sharp 0.8$ se deduce en SQHL a partir del programa del ejemplo 3.3 con el siguiente árbol de prueba:

$$\dfrac{\dfrac{}{pieza(3, roble) \sharp 0.85}\ (2)}{sel\_roble(3) \sharp 0.8}\ (1)$$

donde:

(1) Inferencia con cláusula $(sel\_roble(3) \overset{1}{\leftarrow} pieza(3, roble),\ 1) \in [C_6]_{\mathcal{S}}$ y sustitución $\{X \mapsto 3\}$ donde $0.8 \leq 1 \times \min\{1, 0.85\}$.

(2) Inferencia con cláusula $(pieza(3, roble) \overset{1}{\leftarrow},\ 0.85) \in [C_3]_{\mathcal{S}}$ y sustitución ε donde $0.85 \leq 1 \times \min\{0.85\}$.

Finalmente, la caracterización del modelo mínimo $\mathcal{M}_{\mathcal{P}}$ de un programa $\mathcal{P}$ dado, a partir del cálculo lógico SQHL$(\mathcal{S}, \mathcal{D})$, es

$$\mathcal{M}_{\mathcal{P}} = \{A\sharp d \mid \mathcal{P} \vdash_{\mathcal{S},\mathcal{D}} A\sharp d\}$$

i.e. el conjunto de todos los átomos $A\sharp d$ que se deducen de las cláusulas del programa $\mathcal{P}$ en el cálculo SQHL$(\mathcal{S}, \mathcal{D})$.

3.3.3. Objetivos y soluciones

Tanto los objetivos como las soluciones en SQLP presentan una forma idéntica a la de ambas nociones en el esquema QLP, dado que la relación de similaridad no influye en la definición declarativa de ambas nociones. Por lo tanto, únicamente se recordará la forma de un SQLP-objetivo y la definición de solución.

Un SQLP-objetivo es una conjunción de átomos anotados abiertos $A\sharp W$ y un conjunto de restricciones umbral $W \trianglerighteq \beta$, con $\beta \in D \setminus \{\mathbf{b}\}$, para las variables de cualificación que aparecen en la conjunción de átomos.

Nota 12. En la publicación [11](A.3) no se consideraba la posibilidad de que β tomara el valor "?" al estar basada, la noción de objetivo y solución, en aquella desarrollada en la publicación [59](A.1) para el esquema QLP, que aún no tenía en cuenta dicha posibilidad.

Más concretamente, un objetivo es de la forma:

$$A_1\sharp W_1,\ \ldots,\ A_m\sharp W_m \,[\!]\, W_1 \trianglerighteq \beta_1,\ \ldots,\ W_m \trianglerighteq \beta_m$$

o también, de manera más corta, $(\ A_i\sharp W_i,\ W_i \trianglerighteq \beta_i\ )_{i=1\ldots m}$. Por último, se dice que la pareja $\langle \theta, \rho \rangle$, donde σ es una sustitución de variables por términos y μ es una sustitución de variables de cualificación por valores de cualificación, es una solución a un objetivo $(\ A_i\sharp W_i,\ W_i \trianglerighteq \beta_i\ )_{i=1\ldots m}$ para un SQLP-programa $\mathcal{P}$ sii se tiene $\mathcal{P} \vdash_{\mathcal{S},\mathcal{D}} A_i\theta\sharp W_i\rho$ con $W_i\rho \trianglerighteq \beta_i$ para $i = 1 \ldots m$.

El siguiente ejemplo ilustra esta noción de objetivo y solución utilizando para ello el SQLP-programa del ejemplo 3.3:

Ejemplo 3.6 (SQLP: Objetivo y solución)**.** Sea $\mathcal{S}$ la relación de similaridad del ejemplo 3.1 y $\mathcal{P}$ el SQLP$(\mathcal{S}, \mathcal{U})$-programa del ejemplo 3.3. Entonces:

$$G\ :\ sel\_roble(P)\sharp W \,[\!]\, W \geq 0.8$$

es un objetivo para $\mathcal{P}$, y $\langle \{P \mapsto 3\}, \{W \mapsto 0.85\} \rangle$ una solución para G.

3.4. Transformación de SQLP-programas en QLP-programas

Dado un SQLP($\mathcal{S}, \mathcal{D}$)-programa $\mathcal{P}$, existen dos alternativas a la hora de resolver objetivos G para el programa $\mathcal{P}$:

1. adaptar el procedimiento de resolución SLD para SQLP-programas;
2. transformar los programas y objetivos SQLP en otros programas y objetivos equivalentes para los que exista un procedimiento de resolución de objetivos, y resolver sobre dichos programas y objetivos equivalentes.

En esta sección, y partiendo de la existencia del procedimiento de resolución QSLD($\mathcal{D}$) para QLP($\mathcal{D}$) —que resulta de la simplificación de la resolución BQSLD($\mathcal{D}$) presentada en la subsección 2.4, y que fue presentado en [59](A.1, §4.2)—, se desarrolla un método de transformación de programas y objetivos SQLP en programas y objetivos QLP equivalentes. En cualquier caso, en [64] se desarrolla una adaptación de la resolución SLD estándar para programas lógicos con similaridad, y también hubiera sido posible desarrollar una adaptación de dicha resolución SLD con similaridad para SQLP-programas.

La finalidad será, por tanto, la siguiente: dado un SQLP($\mathcal{S}, \mathcal{D}$)-programa $\mathcal{P}$, y un objetivo G para $\mathcal{P}$, se obtendrá un QLP($\mathcal{D}$)-programa $S_{\mathcal{S}}(\mathcal{P})$, semánticamente equivalente a $\mathcal{P}$, sobre el que ejecutar el objetivo G. Nótese que el objetivo G no requiere de transformación alguna dado que es, sin cambios, tanto un SQLP($\mathcal{S}, \mathcal{D}$)-objetivo para $\mathcal{P}$, como un QLP($\mathcal{D}$)-objetivo para $S_{\mathcal{S}}(\mathcal{P})$.

La intuición que guiará la transformación de SQLP-programas en QLP-programas equivalentes es la de extender el conjunto de cláusulas que componen el SQLP-programa para que incluya toda la información contenida en la relación de similaridad $\mathcal{S}$. De esta manera, puede eliminarse la relación de similaridad manteniéndose el sentido del programa original. Un pequeño ejemplo de esta idea es el siguiente:

Ejemplo 3.7 (Intuición sobre la transformación; I)**.** Sea $\mathcal{D}$ el dominio de cualificación $\mathcal{U}$ de los valores de certidumbre, y sea $\mathcal{S}$ la relación de similaridad del ejemplo 3.1. Entonces, el siguiente conjunto de cláusulas conforma un SQLP($\mathcal{S}, \mathcal{U}$)-programa.

c_1 $\quad pieza(1, granito) \overset{1}{\leftarrow}$
c_2 $\quad pieza(2, roble) \overset{1}{\leftarrow}$
c_6 $\quad sel\_roble(X) \overset{1}{\leftarrow} pieza(X, roble)$

Nótese que es un subconjunto de las cláusulas del programa del ejemplo 3.3.

Intuitivamente, es fácil ver que tanto $pieza(1, granito)\sharp 1$ como $pieza(2, roble)\sharp 1$ pertenecen al modelo mínimo, y por lo tanto también $sel\_roble(2)\sharp 1$ pertenece al modelo mínimo (inferencia SQHL con la cláusula C_6 y la sustitución $\{X \mapsto 2\}$). Sin embargo, debido a la relación de similaridad $\mathcal{S}$, también es cierto que $pieza(1, mármol)\sharp 0.85$, $pieza(2, haya)\sharp 0.85$ y $pieza(2, castaño)\sharp 0.75$ pertenecen al modelo mínimo (♠), y por lo tanto tenemos también que $sel\_roble(2)\sharp 0.85$ y $sel\_roble(2)\sharp 0.75$ pertenecen al modelo mínimo (♣). Los pasos (♠) y (♣) se explican a continuación.

(♠) Por un lado sabemos que el átomo $pieza(1, mármol)$ es una $\mathcal{S}$-instancia a nivel $\delta = 0.85$ de $pieza(1, granito)$. Por lo tanto,

$$pieza(1, granito)\sharp 1 \succcurlyeq_{\mathcal{S},\mathcal{D}} pieza(1, mármol)\sharp 0.85$$

y si el primero está en el modelo mínimo, el segundo lo está también por ser este cerrado bajo $\langle \mathcal{S}, \mathcal{D} \rangle$-implicación.

Por otro lado, tenemos que $pieza(1, mármol)\sharp 0.85$ se prueba directamente en SQHL a partir de la cláusula C_1 con sustitución ε porque $0.85 \leq 1 \times \min\{0.85\} = 1 \times 0.85 = 0.85$.

Los casos de $pieza(2, haya)\sharp 0.85$ y $pieza(2, castaño)\sharp 0.75$ son análogos.

(♣) En este caso tenemos trivialmente que $sel\_roble(2)\sharp 0.85$ y $sel\_roble(2)\sharp 0.75$ pertenecen al modelo mínimo porque $sel\_roble(2)\sharp 1$ pertenece y tanto 0.85 como 0.75 son menores o iguales que 1.

Por otro lado, $sel\_roble(2)\sharp 0.85$ se prueba en SQHL como sigue:

$$\dfrac{\dfrac{}{pieza(2, roble)\sharp 0.85}\ (2)}{sel\_roble(2)\sharp 0.85}\ (1)$$

(1) Cláusula C_6, que es $\mathcal{S}$-instancia de sí misma con $\delta = 1$, con sustitución $\{X \mapsto 2\}$ y donde $0.85 \leq 1 \times \min\{1, 0.85\}$.

(2) Cláusula $C_2' : (pieza(2, haya) \overset{1}{\leftarrow})$, que es $\mathcal{S}$-instancia de C_2 con $\delta = 1$, con sustitución ε y donde $0.85 \leq 1 \times \{0.85\}$.

y $sel\_roble(2)\sharp 0.75$ puede probarse también variando el paso (2) de la derivación anterior para utilizar la $\mathcal{S}$-instancia de C_2 que incluye *castaño*.

De modo que, si quisiéramos ampliar el conjunto de cláusulas del programa con la información contenida en la relación de similaridad $\mathcal{S}$, necesitaríamos transformar el programa al siguiente QLP($\mathcal{U}$)-programa equivalente:

c_1 $\quad pieza(1, granito) \overset{1}{\leftarrow}$
c_1' $\quad pieza(1, mármol) \overset{1}{\leftarrow} pay_{0.85}$
c_2 $\quad pieza(2, roble) \overset{1}{\leftarrow}$
c_2' $\quad pieza(2, haya) \overset{1}{\leftarrow} pay_{0.85}$
c_2'' $\quad pieza(2, castaño) \overset{1}{\leftarrow} pay_{0.75}$
c_6 $\quad sel\_roble(X) \overset{1}{\leftarrow} pieza(X, roble)$
c_7 $\quad pay_{0.85} \overset{0.85}{\longleftarrow}$
c_8 $\quad pay_{0.75} \overset{0.75}{\longleftarrow}$

Donde los átomos *pay* introducidos se encargan de calcular los δ que resultarían de la relación de similaridad en el cálculo de los valores de cualificación finales para las cabezas de las cláusulas, de este modo serían posibles también en QHL las inferencias equivalentes a las mostradas en (♠) y (♣)(2), aunque en QHL requerirían de una inferencia más con la cláusula C_7.

A partir del ejemplo anterior, puede verse que los QLP-programas transformados estarán compuestos por dos conjuntos de cláusulas: a) aquellas que vienen del conjunto de cláusulas original, ya fueran ellas mismas como en el caso de las cláusulas C_1, C_2 y C_6, ya las que resultan de ser cláusulas con una cabeza similar —según $\mathcal{S}$— a alguna cláusula original, como es el caso de C_1', C_2' y C_2''; y b) aquellas que se encargan de computar el grado de similaridad entre la cabeza de una cláusula del primer conjunto y la cabeza de la cláusula SQLP original de la que proviene dicha cláusula, y que en la transformación se codifican en átomos *pay*, como las cláusulas C_7 y C_8, con la que se resolverán los átomos *pay* introducidos en los cuerpos de las cláusulas del primer conjunto.

En este punto, es importante hacer notar que en el ejemplo anterior las cabezas de las cláusulas eran lineales, i.e. se dice que el programa es *lineal por la izquierda*, y por lo tanto las ecuaciones de similaridad que aparecen al linealizar un átomo no lineal no han jugado papel alguno. Para poder ver qué ocurre en el caso de tener un programa que no sea lineal por la izquierda, considérese el siguiente ejemplo:

Ejemplo 3.8 (Intuición sobre la transformación; y II)**.** Sea $\mathcal{D}$ el dominio de cualificación $\mathcal{U}$ de los valores de certidumbre, y sea $\mathcal{S}$ la relación de similaridad del ejemplo 3.1. Entonces, la siguiente cláusula conforma un SQLP$(\mathcal{S},\mathcal{U})$-programa no lineal por la izquierda.

$$_{D_1} \quad p(X,X) \xleftarrow{1}$$

Sin ánimo de ser exhaustivos aquí, nos centraremos en la prueba de dos posibles átomos cualificados que pueden deducirse de la cláusula D_1 en el cálculo SQHL y que resultarán ilustrativos acerca de la problemática de la linealidad en las cabezas. El primero de ellos será $p(\mathit{roble},\mathit{roble})\sharp 1$ ($\heartsuit$), y el segundo $p(\mathit{granito},\mathit{mármol})\sharp 0.85$ ($\diamondsuit$).

En primer lugar, nótese que $\mathrm{lin}(p(X,X)) = (p(X,X_1),\{X \sim X_1\})$. Y por lo tanto tenemos que:

$$D_1' \ : \ p(\mathit{roble},\mathit{roble}) \xleftarrow{1}$$
$$D_1'' \ : \ p(\mathit{granito},\mathit{mármol}) \xleftarrow{1}$$

son $\mathcal{S}$-instancias de la cláusula D_1 con $\delta = 1$ y sustitución $\{X \mapsto \mathit{roble}, X_1 \mapsto \mathit{roble}\}$ para D_1', y $\delta = 0.85$ y sustitución $\{X \mapsto \mathit{granito}, X_1 \mapsto \mathit{mármol}\}$ para D_1''. Nótese además que, en efecto, ($\heartsuit$) se deduce del programa en SQHL con la cláusula D_1' y con $1 \leq 1 \times \min\{1\} = 1$; y que ($\diamondsuit$) se deduce del programa en SQHL con la cláusula D_2'' y con $0.85 \leq 1 \times \min\{0.85\} = 0.85$.

Para conseguir hacer dos inferencias en QHL equivalentes a estas, parece claro que es necesario que el programa transformado tenga una versión lineal de la cláusula D_1 en lugar de la propia D_1, de este modo será posible realizar instanciaciones como la requerida en el caso de ($\diamondsuit$). Sin embargo, escribiendo una cláusula como

$$p(X,X_1) \xleftarrow{1}$$

no obligará en la lógica QHL a que la sustitución θ escogida para el paso de inferencia con dicha cláusula cumpla que $\mathcal{S}(X\theta, X_1\theta) \neq \mathbf{b}$. Por lo tanto, será necesario introducir un predicado especial adicional "$\sim$" que se encargue de obligar a que las sustituciones

cumplan ciertas condiciones de similaridad que, en particular, serán aquellas que resulten de la linealización de la cabeza de la cláusula. En el caso que nos ocupa, tendríamos sólo una condición de similaridad, i.e. $X \sim X_1$, por lo que deberemos añadirla a la versión linealizada de la cláusula, de manera que esta quedaría como sigue

$$D_1''' \quad p(X, X_1) \xleftarrow{1} X \sim X_1$$

Finalmente, añadiendo también las cláusulas apropiadas para el predicado "$\sim$" obtendríamos el comportamiento deseado en las inferencias QHL que utilizaran esta cláusula. En nuestro caso, para poder probar tanto ($\heartsuit$) como ($\diamondsuit$), las cláusulas para "$\sim$" necesarias serían:

$$D_2 \quad X \sim X \xleftarrow{1}$$
$$D_3 \quad \mathit{granito} \sim \mathit{mármol} \xleftarrow{1} \mathit{pay}_{0.85}$$

Y el programa QLP equivalente transformado sería:

$$D_1''' \quad p(X, X_1) \xleftarrow{1} X \sim X_1$$
$$D_2 \quad X \sim X \xleftarrow{1}$$
$$D_3 \quad \mathit{granito} \sim \mathit{mármol} \xleftarrow{1} \mathit{pay}_{0.85}$$
$$D_4 \quad \mathit{pay}_{0.85} \xleftarrow{0.85}$$

permitiendo, en efecto:

($\heartsuit$) probar $p(\mathit{roble}, \mathit{roble})\sharp 1$ en QHL con la siguiente demostración:

$$\dfrac{\dfrac{}{X\theta_1 \sim X_1\theta_1\sharp 1}\,(2)}{p(\mathit{roble}, \mathit{roble})\sharp 1}\,(1)$$

donde

(1) Cláusula D_1''' con $\theta_1 = \{X \mapsto \mathit{roble}, X_1 \mapsto \mathit{roble}\}$ y donde $1 \leq 1 \times \min\{1, 1\}$.

(2) Cláusula D_2 con $\theta_2 = \varepsilon$ y donde $1 \leq 1 \times \min\{\}$.

($\diamondsuit$) probar $p(\mathit{granito}, \mathit{mármol})\sharp 0.85$ en QHL con la siguiente demostración:

$$\dfrac{\dfrac{\dfrac{}{\mathit{pay}_{0.85}\theta_3\sharp 0.85}\,(3)}{X\theta_1 \sim X_1\theta_1\sharp 0.85}\,(2)}{p(\mathit{granito}, \mathit{mármol})\sharp 0.85}\,(1)$$

donde

(1) Cláusula D_1''' con $\theta_1 = \{X \mapsto \mathit{granito}, X_1 \mapsto \mathit{mármol}\}$ y donde $0.85 \leq 1 \times \min\{1, 0.85\}$.

(2) Cláusula D_3 con $\theta_2 = \varepsilon$ y donde $0.85 \leq 1 \times \min\{0.85\}$.

(3) Cláusula D_4 con $\theta_3 = \varepsilon$ y donde $0.85 \leq 0.85 \times \min\{\}$.

Tras este ejemplo, pueden verse dos cosas:

1. Las cláusulas que se utilizan para obtener el primer conjunto de cláusulas de la transformación, i.e. aquellas que vienen de las cláusulas originales, deberán ser las versiones lineales de las cláusulas originales, y no las originales. En cualquier caso, nótese que estas coinciden cuando el programa es lineal por la izquierda como ocurría en el caso del ejemplo 3.7.

2. Deberá añadirse un conjunto adicional de cláusulas que implementen el predicado "$\sim$", de manera que puedan resolverse los átomos correspondientes a dicho predicado y que obligarán a instanciar adecuadamente las cláusulas. Este conjunto de cláusulas para el predicado "$\sim$" trasladarán a QLP la información contenida para las constructoras en la relación de similaridad $\mathcal{S}$. Nótese que la información referida a los símbolos de predicado se utilizará en la generación de las cláusulas adicionales del primer conjunto.

Así se tendrá, formalmente, que dado un SQLP$(\mathcal{S}, \mathcal{D})$-programa $\mathcal{P}$, el QLP$(\mathcal{D})$-programa transformado $S_{\mathcal{S}}(\mathcal{P})$ equivalente será

$$S_{\mathcal{S}}(\mathcal{P}) = \mathcal{P}_{\mathcal{S}} \uplus \mathcal{P}_{\sim} \uplus \mathcal{P}_{\mathsf{pay}}$$

donde

- $\mathcal{P}_{\mathcal{S}}$ es el conjunto de cláusulas con cabezas parecidas a alguna cláusula linealizada del programa original $\mathcal{P}$ que contienen además un átomo *pay* con el grado de similaridad entre la cabeza de dicha cláusula y la cláusula linealizada original de la que proviene;

- $\mathcal{P}_{\sim}$ es el conjunto de cláusulas que implementan el predicado $\sim$ con la información contenida en la relación de similaridad $\mathcal{S}$ para las constructoras de la signatura; y

- $\mathcal{P}_{\mathsf{pay}}$ es el conjunto de cláusulas que permiten deducir los átomos *pay* necesarios que hayan sido introducidos en los cuerpos de las cláusulas presentes en cualquier de los dos conjuntos anteriores.

Para una definición formal y exhaustiva de la transformación, véase [11](A.3, Def. 6).

Por último, y como resultado principal de la trasformación de programas propuesta, se demuestra que dado un SQLP-programa $\mathcal{P}$ cualquiera, y su equivalente QLP-programa transformado $S_{\mathcal{S}}(\mathcal{P})$, se tiene:

$$\mathcal{P} \vdash_{\mathcal{S},\mathcal{D}} A\sharp d \iff S_{\mathcal{S}}(\mathcal{P}) \vdash_{\mathcal{D}} A\sharp d$$

para todo átomo cualificado $A\sharp d$ cuyo símbolo de predicado sea $\mathcal{S}$-similar a un símbolo de predicado que aparezca en alguna cláusula de $\mathcal{P}$. Este resultado prueba que todo átomo cualificado $A\sharp d$ de la signatura de $\mathcal{P}$ que es deducible en SQHL a partir del programa $\mathcal{P}$, puede también deducirse en QHL a partir del programa $S_{\mathcal{S}}(\mathcal{P})$. Y también, que todo átomo cualificado $A\sharp d$ de la signatura de $\mathcal{P}$ que es deducible en QHL a partir del programa $S_{\mathcal{S}}(\mathcal{P})$, puede también deducirse en SQHL a partir del programa $\mathcal{P}$. Esto prueba, de hecho, la equivalencia semántica de ambos programas en sus respectivos esquemas. Este resultado está demostrado en [11](A.3, T.1).

Antes de terminar esta sección, resulta conveniente hacer dos comentarios que serán de relevancia en las secciones que siguen a continuación:

- La ausencia de igualdad con similaridad en el esquema hace necesario aplicar una técnica como la de la linealización de los lados izquierdos de las cláusulas, y al desarrollo de las nociones de $\mathcal{S}$-instancia que se han presentado. Aunque resulta posible variar las técnicas y nociones empleadas, es evidente que sin igualdad, se hace necesario algún mecanismo que permita obtener instancias "no tradicionales" de términos, átomos y cláusulas debidas a la relación de similaridad.
- El procedimiento de transformación seguido presenta la ventaja de evitar el uso del predicado "$\sim$" cuando el programa SQLP es un programa lineal por la izquierda. Esto es así, porque los parecidos entre símbolos de predicado, o constructora, que aparezcan en los lados izquierdos de las cláusulas se resuelven, durante el proceso de transformación, añadiendo cláusulas adicionales con su respectivo átomo *pay* que asegura que se tenga en cuenta el grado de similaridad de ambas cabezas. Sin embargo, en esquemas de programación que permitan el uso de la igualdad es posible simplificar tanto la noción de instancia utilizada en la semántica de programas con similaridad como la técnica de transformación de programas con similaridad en programas sin similaridad semánticamente equivalentes. Así se hará en los esquemas de programación declarativa cualificada presentados en el resto de la tesis.

4. Programación lógico-funcional con restricciones y cualificación

Las extensiones de la programación lógica que se han desarrollado hasta ahora incluyen la utilización de los valores de cualificación de un dominio de cualificación dado como generalización de los valores de incertidumbre utilizados por van Emden [71], y la utilización de las relaciones de similaridad al estilo de [64] como una forma de modelar la incertidumbre más cercana a la intuición.

En esta sección, se extiende el esquema QLP de la programación lógica cualificada con dos elementos que no habían aparecido hasta ahora: por un lado, se incorpora un dominio de restricciones como los empleados en la programación lógica con restricciones (CLP); y por otro lado, se incorpora también la posibilidad de utilizar funciones perezosas de primer orden al estilo de la programación funcional (FP), dando lugar a un esquema paramétrico que se denomina QCFLP para la programación lógico-funcional con restricciones y cualificación. A pesar de que podría también hacerse uso de las relaciones de similaridad que se presentaron en la sección anterior, se optará ahora por dejarlas fuera del esquema para evitar las dificultades técnicas propias de la similaridad y, así, centrarse en el desarrollo de un marco de programación lógico-funcional básico con cualificación.

Más concretamente, se parte de un fragmento de primer orden del esquema CFLP($\mathcal{C}$) para la programación lógico-funcional con restricciones sobre un dominio paramétrico $\mathcal{C}$ de restricciones presentado en [47]. Se extenderá dicho fragmento a un esquema QCFLP($\mathcal{D},\mathcal{C}$) en el que el parámetro adicional $\mathcal{D}$ representa un dominio de cualificación. Un QCFLP($\mathcal{D},\mathcal{C}$)-programa $\mathcal{P}$ será, entonces, un conjunto de reglas condicionales de reescritura de la forma

$$f(\overline{t}_n) \xrightarrow{\alpha} r \Leftarrow \Delta$$

donde la condición Δ es una conjunción de $\mathcal{C}$-restricciones que pueden involucrar funciones definidas por el usuario, y α es el factor de atenuación de la regla. Como ocurría en los

esquemas de programación lógica con cualificación, el factor α representa la confianza del usuario en la inferencia indicada por la regla en cuestión: cualquier aplicación satisfactoria de la regla aplica al valor devuelto una cualificación que no puede exceder de la computada por medio del cálculo $\alpha \circ d$, donde d es el mínimo de los valores de cualificación computados para la expresión r y las $\mathcal{C}$-restricciones pertenecientes a la conjunción Δ. Es sencillo observar que cualquier cláusula de un QLP($\mathcal{D}$)-programa puede ser formulada como caso particular de regla de un QCFLP($\mathcal{D}, \mathcal{C}$)-programa.

Dos sistemas para la programación lógico-funcional con restricciones que pueden servir de ilustración sobre la expresividad del marco utilizado como base para la extensión que se propone en esta sección son los sistemas `Toy` [6] y `Curry` [29]. Por último, el informe técnico [12](B.3) es una versión ampliada con demostraciones completas de los resultados de la publicación [13](A.4) en la que se basa esta sección de la tesis.

4.1. Dominios de restricciones

Los dominios de restricciones se emplean en CLP y sus extensiones como un mecanismo para proporcionar valores de datos, funciones primitivas y restricciones específicas para aplicaciones orientadas a un área también específica. Existen varias formalizaciones de esta noción dadas por diferentes autores. Aquí emplearemos dominios de restricciones relacionados con signaturas Σ que incluyen un conjunto universal de símbolos de constructora de datos, un conjunto universal de símbolos de función definida y un conjunto de símbolos de función primitiva determinado por el dominio de restricciones considerado. Además, dada una signatura, un símbolo $\bot$ para representar el *valor indefinido*, un conjunto B de valores básicos y un conjunto de variables $\mathcal{V}ar$, definimos las siguientes nociones donde $\overline{o}_n$ abrevia $o_1, \ldots, o_n$.

- *Expresiones* e. Son: o bien el valor indefinido $\bot$; o bien una variable $X \in \mathcal{V}ar$; o bien un valor básico $u \in B$; o bien de la forma $h(\overline{e}_n)$ donde h es un símbolo de constructora, de función definida o de función primitiva. Cuando $n = 0$, $h(\overline{e}_0)$ se escribe simplemente como h.

- *Términos construidos* t. Son: o bien el valor indefinido $\bot$; o bien una variable $X \in \mathcal{V}ar$; o bien un valor básico $u \in B$; o bien de la forma $c(\overline{e}_n)$ donde c es un símbolo de constructora. Cuando $n = 0$, $c(\overline{t}_0)$ se escribe simplemente c, y en adelante, se denominarán sencillamente *términos*.

- Expresiones o términos *totales*. Una expresión o un término se dice que es *total* cuando no es, ni contiene, el valor indefinido $\bot$.

- Expresiones o términos *cerrados*. Una expresión o un término (totales o no) es *cerrado* si no es una variable, ni contiene variables $X \in \mathcal{V}ar$.

- *Orden de información* $\sqsubseteq$. Se define como el menor orden parcial sobre las expresiones compatible con contextos y que verifica que $\bot \sqsubseteq e$ para toda expresión e.

- *Sustitución*. Se definen como funciones de variables a términos (no necesariamente totales), y se representan, tal y como hemos hecho hasta ahora, como conjuntos de vínculos $X \mapsto t$ extendidos para actuar sobre otros objetos sintácticos o.

Adaptando ahora la definición [47](§2.2, Def. 1) a primer orden, se formaliza un dominio de restricciones de signatura Σ como una estructura algebraica de la forma $\mathcal{C} = \langle C, \{p^{\mathcal{C}} \mid p \text{ es símbolo de función primitiva}\}\rangle$ tal que:

1. El conjunto soporte C es el conjunto de los términos cerrados sobre la signatura Σ y el conjunto de valores básicos B.

2. $p^{\mathcal{C}}$ es la interpretación de la primitiva p en el dominio de restricciones $\mathcal{C}$.

Para una formalización más rigurosa, véase [12](B.3, Def. 2).

Se asumirá, de ahora en adelante, la existencia de dos constructoras *true* y *false* para los valores booleanos, y un símbolo de función primitiva binaria $==$ (que se usará en notación infija) para la *igualdad estricta*; véase [47] para más detalles.

Para los ejemplos de esta sección, y la siguiente, y como ejemplo representativo de los dominios de restricciones, se considera el dominio de restricciones $\mathcal{R}$ de las restricciones reales cuyo conjunto de elementos básicos es el conjunto $\mathbb{R}$ de los números reales y cuyas funciones primitivas se corresponden con las operaciones aritméticas $+$, $\times$, ... y las operaciones de comparación $\leq$, $<$, ... usuales sobre $\mathbb{R}$. Otros ejemplos de dominios de restricciones pueden encontrarse en [47].

Finalmente, se definen las tres nociones siguientes:

- *$\mathcal{C}$-restricción atómica*. Son de la forma $p(\overline{e}_n) == v$ donde p es un símbolo de función primitiva y v es o bien una variable; o bien una constante (constructora de aridad 0); o bien un valor básico. Las $\mathcal{C}$-restricciones atómicas de la forma $p(\overline{e}_n) == \mathit{true}$ se abrevian como $p(\overline{e}_n)$; en particular, $(e_1 == e_2) == \mathit{true}$ se abrevia como $e_1 == e_2$. Y las $\mathcal{C}$-restricciones atómicas de la forma $(e_1 == e_2) == \mathit{false}$ se abrevian como $e_1 /= e_2$.

- *$\mathcal{C}$-restricción compuesta*. Se construyen a partir de $\mathcal{C}$-restricciones atómicas empleando conjunciones lógicas, cuantificaciones existenciales y, a veces, otras operaciones lógicas. Las $\mathcal{C}$-restricciones sin apariciones de símbolos f de función definida se denominan *primitivas*. A lo largo de esta sección, denotaremos δ a las $\mathcal{C}$-restricciones atómicas, Δ a los conjuntos de $\mathcal{C}$-restricciones atómicas, π a las $\mathcal{C}$-restricciones atómicas primitivas y Π a los conjuntos de $\mathcal{C}$-restricciones atómicas primitivas. Cuando se interpreten conjuntos de $\mathcal{C}$-restricciones, se tratarán como la conjunción de sus $\mathcal{C}$-restricciones atómicas miembro.

- *Valoración de variables sobre $\mathcal{C}$*. Son sustituciones de variables por términos cerrados, i.e. tales que a cada variable de su dominio le asignan un término cerrado, o lo que es lo mismo, que no contiene variables.

Por último, para las posibles instancias del esquema QCFLP objeto de esta sección, interesará que sus dos parámetros estén relacionados de una manera similar a como lo estaba la relación de similaridad $\mathcal{S}$ con respecto al dominio de cualificación $\mathcal{D}$ en las instancias del esquema SQLP($\mathcal{S}, \mathcal{D}$). En este caso, se dice que una pareja $\langle \mathcal{D}, \mathcal{C}\rangle$ es *admisible* cuando el dominio de cualificación $\mathcal{D}$ es *expresable* en el dominio de restricciones $\mathcal{C}$. Y se dice que $\mathcal{D}$ es expresable en $\mathcal{C}$ cuando el conjunto soporte de $\mathcal{C}$ incluye todos los elementos del conjunto $\mathcal{D}$ (excepto quizás **b**) y además es posible construir dos $\mathcal{C}$-restricciones primitivas especiales que sirven para calcular con valores de cualificación.

Nota 13. En posteriores publicaciones sobre el esquema SQCLP, la condición de que el conjunto soporte de $\mathcal{C}$ incluya a $D \setminus \{\mathbf{b}\}$ se ha relajado a la condición más general de que C incluya una copia isomorfa de $D \setminus \{\mathbf{b}\}$.

Estas dos $\mathcal{C}$-restricciones primitivas son:

1. **qVal**(X). Que se verifica cuando la variable X toma un valor de cualificación válido, i.e. distinto de **b**.
2. **qBound**(X, Y, Z). Que se verifica cuando las tres variables toman valores de cualificación válidos (distintos de **b**) y tales que $X \preccurlyeq Y \circ Z$ se verifica también, cuando $\preccurlyeq$ y $\circ$ se sustituyen por las operaciones necesarias dependiendo del dominio de cualificación $\mathcal{D}$ que se desee expresar en $\mathcal{C}$ (y asumiendo que estas fueran expresables en $\mathcal{C}$).

Un ejemplo de la construcción de estas dos $\mathcal{C}$-restricciones es el siguiente.

Ejemplo 4.1 (Expresando $\mathcal{W}$ en $\mathcal{R}$)**.** Tomando el dominio de cualificación $\mathcal{W}$ de las profundidades ponderadas de la prueba, y el dominio $\mathcal{R}$ de las restricciones reales, se construyen las $\mathcal{R}$-restricciones primitivas **qVal**(X) y **qBound**(X, Y, Z) de la siguiente manera:

- **qVal**$(X) = X > 0$.
- **qBound**$(X, Y, Z) =$ **qVal**$(X) \wedge$ **qVal**$(Y) \wedge$ **qVal**$(Z) \wedge (X \leq Y + Z)$.

4.2. Sintaxis

Para la presentación de la sintaxis del esquema QCFLP, interesan instancias QCFLP$(\mathcal{D}, \mathcal{C})$ tales que la pareja $\langle \mathcal{D}, \mathcal{C} \rangle$ sea admisible en el sentido de que $\mathcal{D}$ sea expresable en $\mathcal{C}$, tal y como se ha visto al final de la subsección anterior. Son ejemplos de parejas admisibles las parejas $\langle \mathcal{B}, \mathcal{R} \rangle$, $\langle \mathcal{U}, \mathcal{R} \rangle$, $\langle \mathcal{W}, \mathcal{R} \rangle$ y cualquier otra compuesta por un producto estricto de $\mathcal{B}$, $\mathcal{U}$ y/o $\mathcal{W}$ y el dominio de restricciones $\mathcal{R}$.

Nota 14. Esto último sólo funciona si se permite que el conjunto soporte de $\mathcal{R}$ incluya una copia isomorfa de $D \setminus \{\mathbf{b}\}$ porque los elementos (d_1, d_2) de un dominio producto no son, en general, elementos del conjunto soporte de $\mathcal{R}$.

En la introducción de esta sección ya se indicó que un QCFLP$(\mathcal{D}, \mathcal{C})$-programa $\mathcal{P}$ es un conjunto de reglas de programa condicionales de la forma

$$f(\overline{t}_n) \xrightarrow{\alpha} r \Leftarrow \Delta$$

En este punto se dirá, además, que f es un símbolo de función definida, $\overline{t}_n$ es una secuencia lineal de términos, i.e. en la que cada variable aparece una única vez, α es un factor de atenuación del dominio $\mathcal{D}$ de cualificación distinto de **b**, r es una expresión y Δ una secuencia de $\mathcal{C}$-restricciones atómicas $\delta_1, \ldots, \delta_m$ interpretada como su conjunción. Por último, debe

tenerse en cuenta que el símbolo indefinido ($\bot$), que podía formar parte de términos y expresiones, no puede aparecer en una regla de programa por lo que todos los términos t_i, la expresión r y las posibles $\mathcal{C}$-restricciones δ_i que conforman una regla de programa deberán ser totales.

A modo de ejemplo, considérese la versión lógico-funcional del ejemplo 2.1:

Ejemplo 4.2 (QCFLP: Números de Peano)**.** Sea $\mathcal{D} = \mathcal{W}$ y $\mathcal{C} = \mathcal{R}$. Entonces, el siguiente conjunto de reglas de programa es un QCFLP($\mathcal{W}, \mathcal{R}$)-programa para representar los números naturales según la representación de Peano y sumar con ellos:

1. $num \xrightarrow{0} c$
2. $num \xrightarrow{1} s(num)$
3. $suma(c, X) \xrightarrow{0} X$
4. $suma(s(X), Y) \xrightarrow{1} s(suma(X, Y))$

Nótese que tanto c como s son símbolos de constructora de datos, y que tanto *num* como *suma* son símbolos de función definida.

Otro posible ejemplo es el siguiente, en el que define una función para estimar el nivel adecuado de lectura de un libro para estudiantes de lengua, a partir del nivel de su vocabulario, género y número de páginas.

Ejemplo 4.3 (Nivel de lectura)**.** Sea $\mathcal{D} = \mathcal{U}$ y $\mathcal{C} = \mathcal{R}$. Entonces el siguiente conjunto de reglas de programa es un QCFLP($\mathcal{U}, \mathcal{R}$)-programa capaz de estimar el nivel de dificultad de un libro dado, teniendo en cuenta su vocabulario, su género y su número de páginas:

1. $estima\_nivel(fácil, G, Págs) \xrightarrow{1} básico \Leftarrow Págs < 50$
2. $estima\_nivel(fácil, G, Págs) \xrightarrow{0.8} intermedio \Leftarrow Págs \geq 50$
3. $estima\_nivel(V, infantil, Págs) \xrightarrow{0.9} básico$
4. $estima\_nivel(difícil, G, Págs) \xrightarrow{0.9} proficiencia \Leftarrow Págs \geq 200$
5. $estima\_nivel(difícil, G, Págs) \xrightarrow{0.8} alto \Leftarrow Págs < 200$
6. $estima\_nivel(medio, G, Págs) \xrightarrow{0.8} intermedio$
7. $estima\_nivel(medio, G, Págs) \xrightarrow{0.7} alto$

Nótese que este ejemplo está basado en el subconjunto de las reglas de programa para la función definida `guessReaderLevel` del ejemplo presentado en [11](A.3, Fig. 1).

4.3. Semántica declarativa

Fijándonos en los ejemplos 4.2 y 4.3 que acaban de verse, resulta evidente que, aún sin tener en cuenta los factores de atenuación, se trata claramente de sistemas de reescritura de términos condicionales no confluentes, i.e. viendo únicamente las reglas de programa 1 y 2 del primero de los ejemplos está claro que pueden obtenerse resultados diferentes para una misma llamada a la función *num*. Las funciones diseñadas para comportarse de esa manera

se dice que son *funciones no deterministas*, y tal y como se argumenta en [57], la semántica para funciones no deterministas no puede describirse adecuadamente mediante la reescritura clásica. Por ello, y para la definición de la semántica declarativa del esquema QCFLP, se opta aquí por diseñar un sistema formal de inferencia denominado *lógica de reescritura con restricciones condicional* o, más brevemente, QCRWL($\mathcal{D}, \mathcal{C}$). Nótese que también es posible caracterizar la semántica declarativa de QCFLP-programas como semántica de punto fijo, a pesar de que se opte únicamente aquí por una caracterización de la semántica basada en una lógica de reescritura. Una caracterización de este tipo, mediante un transformador de interpretaciones, se desarrolla con todo detalle en la primera parte de [12](B.3, §3.2).

4.3.1. Semántica basada en una lógica de reescritura

El primer paso es el de definir el tipo de enunciados declarativos que podrán inferirse en la lógica QCRWL($\mathcal{D}, \mathcal{C}$). De manera resumida, podrán inferirse *qc-declaraciones* φ de las dos formas siguientes:

- $\varphi : (e \to t)\sharp d \Leftarrow \Pi$, i.e. φ es una *qc-producción*, donde e es una expresión, t un término parcial, d un valor de cualificación distinto de **b** y Π un conjunto de $\mathcal{C}$-restricciones primitivas atómicas. Intuitivamente, significa que t es uno de los valores posibles que se pueden obtener evaluando e con respecto a las reglas del programa, bajo el supuesto de que Π se satisface. Si t es un término parcial con apariciones de $\bot$, se deberá entender como una aproximación de un valor total de e. Además, debido al indeterminismo, $(e \to t)\sharp d \Leftarrow \Pi$ se podrá cumplir para varias elecciones diferentes de t. Las qc-producciones de la forma $(f(\overline{t}_n) \to t)\sharp d \Leftarrow \Pi$ donde f es un símbolo de función definida de aridad n se llaman *qc-hechos*.

- $\varphi : \delta\sharp d \Leftarrow \Pi$, i.e. φ es un *qc-átomo*, donde δ es una $\mathcal{C}$-restricción atómica (no necesariamente primitiva), d un valor de cualificación distinto de **b** y Π un conjunto de $\mathcal{C}$-restricciones primitivas atómicas. Intuitivamente, significa que el átomo δ se deduce de las reglas de programa con valor de cualificación asociado d, bajo la condición de que Π se satisface.

Además, una qc-declaración φ se dice que es *trivial* cuando Π es insatisfactible, o también cuando φ es una qc-producción de la forma $(e \to \bot)\sharp d \Leftarrow \Pi$.

A continuación, se define QCRWL($\mathcal{D}, \mathcal{C}$) como el sistema formal de inferencia que puede verse en [13](A.4, Fig. 2), y cuyas reglas de inferencia permiten, a partir de un QCFLP($\mathcal{D}, \mathcal{C}$)-programa $\mathcal{P}$, un conjunto Π de $\mathcal{C}$-restricciones primitivas atómicas y un valor de cualificación d distinto de **b**, deducir lo siguiente:

- (**QTI**) toda qc-declaración trivial;
- (**QRR**) toda qc-producción de la forma $(v \to v)\sharp d \Leftarrow \Pi$ donde v es una variable o un valor básico;
- (**QDC**) toda qc-producción de la forma $c(\overline{e}_n) \to c(\overline{t}_n)\sharp d \Leftarrow \Pi$ donde c es un símbolo de constructora de aridad n y d es tal que $d \trianglelefteq d_i$ con $i = 1 \ldots n$, siempre que
  - $(e_i \to t_i)\sharp d_i \Leftarrow \Pi$ para $i = 1 \ldots n$

  se deduzca de $\mathcal{P}$;

- (**QDF**$_{\mathcal{P}}$) toda qc-producción de la forma $(f(\overline{e}_n) \to t)\sharp d \Leftarrow \Pi$ con la instancia $f(\overline{t}_n) \xrightarrow{\alpha} r \Leftarrow \delta_1, \ \ldots, \ \delta_m$ de una regla de programa de $\mathcal{P}$, donde f es un símbolo de función definida de aridad n y d es tal que $d \trianglelefteq d_i$ con $i = 1 \ldots n$ y $d \trianglelefteq \alpha \circ d'_j$ con $j = 0 \ldots m$, siempre que

  - $(e_i \to t_i)\sharp d_i \Leftarrow \Pi$ para $i = 1 \ldots n$,
  - $(r \to t)\sharp d'_0 \Leftarrow \Pi$, y
  - $\delta_j \sharp d'_j \Leftarrow \Pi$ para $j = 1 \ldots m$

  se deduzcan de $\mathcal{P}$;

- (**QPF**) toda qc-producción de la forma $(p(\overline{e}_n) \to v)\sharp d \Leftarrow \Pi$ siendo $p(\overline{t}_n) \to v$ una consecuencia de Π, y donde p es un símbolo de función primitiva de aridad n, v es variable, constante o valor básico, d es tal que $d \trianglelefteq d_i$ con $i = 1 \ldots n$, siempre que

  - $(e_i \to t_i)\sharp d_i \Leftarrow \Pi$ para $i = 1 \ldots n$

  se deduzca de $\mathcal{P}$; y

- (**QAC**) toda qc-producción de la forma $(p(\overline{e}_n) == v)\sharp d \Leftarrow \Pi$ siendo $p(\overline{t}_n) == v$ una consecuencia de Π, y donde p es un símbolo de función primitiva de aridad n, v es variable, constante o valor básico, d es tal que $d \trianglelefteq d_i$ con $i = 1 \ldots n$, siempre que

  - $(e_i \to t_i)\sharp d_i \Leftarrow \Pi$ para $i = 1 \ldots n$

  se deduzca de $\mathcal{P}$.

En particular, la regla **QDF**$_{\mathcal{P}}$ formaliza la aplicación de la instancia de una regla de programa de $\mathcal{P}$ para inferir que $f(\overline{e}_n)$ devuelve como resultado t con cualificación d. Nótese que d estará limitado por las cualificaciones d_i correspondientes a la evaluación de los argumentos e_i, y también por los $\alpha \circ d'_j$ correspondientes a la atenuación con α de las evaluaciones del lado derecho y de las condiciones de la regla.

De manera similar a como se ha hecho con el resto de cálculos de inferencia lógica, se escribirá $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} \varphi$ para indicar que φ se infiere en QCRWL a partir del QCFLP-programa $\mathcal{P}$ en un número finito de pasos de inferencia con las reglas de inferencia. Como ilustración del proceso de inferencia, véase el siguiente ejemplo.

Ejemplo 4.4 (Derivación en QCRWL($\mathcal{D}, \mathcal{C}$))**.** Considérese el programa del ejemplo 4.3. Entonces, puede inferirse la qc-producción

$$(\mathit{estima\_nivel}(\mathit{difícil}, \mathit{ensayo}, 59) \to \mathit{alto})\sharp 0.9 \Leftarrow \emptyset$$

a partir de las reglas de programa mediante el siguiente árbol de inferencia:

$$\dfrac{(\spadesuit) \quad (\clubsuit) \quad (\heartsuit) \quad \dfrac{}{(\mathit{alto} \to \mathit{alto})\sharp 1 \Leftarrow \emptyset}\ (2) \quad \dfrac{(\heartsuit) \quad (\diamondsuit)}{(59 < 200)\sharp 1 \Leftarrow \emptyset}\ (3)}{(\mathit{estima\_nivel}(\mathit{difícil}, \mathit{ensayo}, 59) \to \mathit{alto})\sharp 0.9 \Leftarrow \emptyset}\ (1)$$

con:

$$(\spadesuit)\quad \frac{}{(\mathit{difícil} \to \mathit{difícil})\sharp 1 \Leftarrow \emptyset}\ (2)$$

$$(\clubsuit)\quad \frac{}{(\mathit{ensayo} \to \mathit{ensayo})\sharp 1 \Leftarrow \emptyset}\ (2)$$

$$(\heartsuit)\quad \frac{}{(59 \to 59)\sharp 1 \Leftarrow \emptyset}\ (4)$$

$$(\diamondsuit)\quad \frac{}{(200 \to 200)\sharp 1 \Leftarrow \emptyset}\ (4)$$

y donde cada paso de inferencia es posible porque:

(1) Inferencia **QDF**$_{\mathcal{P}}$ con instancia de la regla de programa 5 con sustitución $\{G \mapsto \mathit{ensayo},\ \mathit{Págs} \mapsto 59\}$. Nótese que $0.9 \leq 1$ (para las tres primeras premisas) y que $0.9 \leq 0.9 \times 1$ (para las dos últimas premisas).

(2) Inferencia **QDC** dado que *alto*, *difícil* y *ensayo* son constantes.

(3) Inferencia **QAC** con $\emptyset \models_{\mathcal{R}} (59 < 200) == \mathit{true}$. Nótese que en realidad el qc-átomo es $(<(59, 200) == \mathit{true})\sharp 1 \Leftarrow \emptyset$ con $<$ símbolo de función primitiva, y que este puede simplificarse en $<(59, 200)\sharp 1 \Leftarrow \emptyset$ o también $(59 < 200)\sharp 1 \Leftarrow \emptyset$ como aparece en la derivación.

(4) Inferencia **QRR** dado que 59 y 200 son valores básicos.

Extendiendo las ideas de [47], es posible definir las *qc-interpretaciones* [12](B.3, Def. 6) como conjuntos de qc-hechos que verifican ciertas condiciones de clausura como [12](B.3, Def. 5) y, además, los *modelos* [12](B.3, Def. 7) de un QCFLP-programa $\mathcal{P}$ pueden definirse como esas qc-interpretaciones que satisfacen el conjunto de reglas del programa $\mathcal{P}$ de un modo adecuado. Finalmente, es posible demostrar que el menor modelo de un QCFLP-programa $\mathcal{P}$ es

$$S_{\mathcal{P}} = \{\varphi \mid \varphi \text{ es un qc-hecho y } \mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} \varphi\}$$

tal y como se prueba en [13](A.4, T.1), es decir, que coincide con el conjunto de qc-hechos que son derivables en QCRWL a partir de las reglas de programa de $\mathcal{P}$. Nótese que esto incluye, en particular, todos los qc-hechos triviales.

4.3.2. Objetivos y soluciones

Las nociones de *objetivo* y *solución* que se presentan aquí continúan en la misma dirección que las nociones análogas para los esquemas vistos con anterioridad. En este caso, un objetivo G para un QCFLP-programa $\mathcal{P}$ tendrá la forma

$$\delta_1\sharp W_1,\ \ldots,\ \delta_m\sharp W_m \,[\!]\, W_1 \trianglerighteq \beta_1,\ \ldots,\ W_m \trianglerighteq \beta_m$$

abreviado como $(\ \delta_i\sharp W_i,\ W_i \trianglerighteq \beta_i\ )_{i=1\ldots m}$, donde $\delta_i\sharp W_i$ son $\mathcal{C}$-restricciones atómicas (no necesariamente primitivas) anotadas con variables de cualificación W_i donde recoger el valor de cualificación final con el que se prueba δ_i, y $W_i \trianglerighteq \beta_i$ son las condiciones umbral que limitan los valores de cualificación aceptables con los que se espera puedan probarse los átomos δ_i del objetivo.

De manera similar a como se presentaron los objetivos en los esquemas anteriores, las β_i de las condiciones umbral podrían perfectamente aceptar un valor "?" que permitiese convertir en trivial la condición umbral, pero esto no se desarrolló en la publicación [13](A.4).

La noción de *solución* será en este caso una terna $\langle \sigma, \mu, \Pi \rangle$ donde σ es una sustitución de variables por términos, μ una sustitución de variables de cualificación por valores de cualificación y Π un conjunto de $\mathcal{C}$-restricciones primitivas atómicas tales que:

1. $W_i\mu = d_i \trianglerighteq \beta_i$ para $i = 1 \ldots m$, y
2. $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} \delta_i\sigma \sharp W_i\mu \Leftarrow \Pi$ para $i = 1 \ldots m$.

Es decir, si puede probarse en QCRWL a partir de $\mathcal{P}$ toda $\mathcal{C}$-restricción atómica $\delta_i \sharp W_i$ del objetivo, implicada por Π, una vez instanciado δ_i por σ y W_i por μ. Y donde, además, μ asigna valores aceptables a las variables de cualificación — en el sentido de que se satisfagan todas las condiciones umbrales impuestas en el objetivo.

Ejemplo 4.5 (Objetivo y solución)**.** Sea $\mathcal{P}$ el QCFLP($\mathcal{U}, \mathcal{R}$)-programa del ejemplo 4.3. Entonces:

$$G \;:\; (\mathit{estima\_nivel}(\mathit{difícil}, \mathit{ensayo}, 59) == \mathit{alto}) \sharp W \;[\!]\; W \geq 0.9$$

es un objetivo para $\mathcal{P}$ y $\langle \varepsilon, \{W \mapsto 0.9\} \rangle$ una solución para G.

4.4. Transformación de QCFLP-programas en CFLP-programas

La resolución de objetivos para las instancias CFLP($\mathcal{C}$) de [47] ha sido formalizada por medio de técnicas de *estrechamiento con restricciones* como puede verse en [46, 16], y está soportado en sistemas como `Curry` [29] y `Toy` [6]. En esta subsección estaremos interesados en proporcionar herramientas adecuadas tanto para la resolución de objetivos de instancias QCFLP($\mathcal{D}, \mathcal{C}$) como para la implementación práctica de dichas instancias y, al igual que ocurría para el esquema SQLP presentado en la sección 3, se puede pensar en dos posibles alternativas: a) extender con cualificación un procedimiento de resolución de CFLP-objetivos existente; o b) transformar los QCFLP-programas en CFLP-programas equivalentes para los que se conozcan procedimientos de resolución de objetivos.

Con vistas a una futura implementación de instancias del esquema QCFLP, la segunda opción resulta más conveniente y permite que el trabajo de implementación se centre en la implementación de la transformación de los QCFLP-programas a CFLP-programas equivalentes, pues los sistemas `Curry` [29] y `Toy` [6] ya pueden ejecutar CFLP-programas adecuadamente, y no tanto en la implementación completa del método de resolución —con todo el trabajo que ello conlleva— cuando la mayor parte de los problemas técnicos a los que nos enfrentaríamos han sido ya resueltos en los dos sistemas que hemos comentado. Por lo tanto, al igual que ya ocurriese en la sección 3, optaremos aquí por presentar una transformación de QCFLP($\mathcal{D}, \mathcal{C}$)-programas en CFLP($\mathcal{C}$)-programas equivalentes o, más concretamente, en programas de un fragmento de primer orden de CFLP($\mathcal{C}$), al que por abuso de notación nos referiremos también como CFLP($\mathcal{C}$).

En [47] se ha mostrado que la semántica declarativa de CFLP($\mathcal{C}$) puede caracterizarse mediante una *lógica de reescritura con restricciones* CRWL($\mathcal{C}$) en la que pueden inferirse *c-declaraciones* a partir de un programa dado. Una *c-declaración* será o bien una *c-producción* de la forma $e \to t \Leftarrow \Pi$, o bien un *c-átomo* de la forma $\delta \Leftarrow \Pi$. Si nos restringimos al fragmento de primer orden del esquema CFLP presentado en [47], la lógica de reescritura CRWL($\mathcal{C}$) puede formalizarse mediante seis reglas de inferencia que se obtienen sencillamente por eliminación de todo lo relativo a la cualificación del conjunto de reglas de reescritura presentadas en la subsección 4.3.1 para QCRWL($\mathcal{D}, \mathcal{C}$) y, aunque en [13](A.4, §4) se da también la idea general de esta simplificación, al principio de [12](B.3, §4) se presenta la lógica de reescritura CRWL($\mathcal{C}$) completa. Análogamente, la notación $\mathcal{P} \vdash_{\mathcal{C}} \varphi$ indica que φ puede inferirse a partir de $\mathcal{P}$ en CRWL($\mathcal{C}$) y, también en analogía con la caracterización del modelo mínimo de un QCFLP($\mathcal{D}, \mathcal{C}$)-programa, es posible probar que $S_{\mathcal{P}} = \{\varphi \mid \varphi$ es un c-hecho y $\mathcal{P} \vdash_{\mathcal{C}} \varphi\}$ es el menor modelo de un CFLP($\mathcal{C}$)-programa $\mathcal{P}$. Por lo que, para el resto de esta subsección, bastarán inferencias formales en las lógicas de reescritura QCWRL($\mathcal{D}, \mathcal{C}$) y CRWL($\mathcal{C}$) para demostrar la corrección y completitud de la transformación propuesta.

De manera análoga a la definición presentada en la subsección 4.3.2 para objetivos y soluciones en el esquema QCFLP, los objetivos G para un CFLP($\mathcal{C}$)-programa $\mathcal{P}$ tienen la forma $\delta_1, \ldots, \delta_m$ donde cada δ_i es una $\mathcal{C}$-restricción atómica; y las soluciones son pares $\langle \sigma, \Pi \rangle$ tales que σ es una sustitución de variables por términos, Π es un conjunto de $\mathcal{C}$-restricciones primitivas atómicas y se tiene que pueden probarse en CRWL, a partir de $\mathcal{P}$, todas las $\mathcal{C}$-restricciones atómicas δ_i del objetivo una vez instanciadas por σ e implicadas por Π.

La transformación que se presenta a continuación presenta dos diferencias fundamentales con la presentada para SQLP-programas en la subsección 3.4:

1. Dado que el esquema CFLP no tiene cualificación, en esta transformación sí deberá indicarse cómo se debe operar con los valores de cualificación y, por lo tanto, aparecerán explícitamente las dos restricciones de cualificación mencionadas al final de la subsección 4.1, y que son dependientes de los dominios $\mathcal{D}$ y $\mathcal{C}$ elegidos como parámetros de la instancia QCFLP($\mathcal{D}, \mathcal{C}$) que se está implementando.

2. Dado que los símbolos de función definida —que son a un QCFLP-programa lo que los símbolos de predicado a un SQLP-programa— aumentarán en uno su número de argumentos —el que corresponderá a la variable de cualificación—, y dado que estos pueden aparecer tanto en los argumentos de la cabeza de una regla de programa, como en la expresión del lado derecho y en las condiciones de la regla, la transformación de expresiones deja de ser trivial, como era el caso de los átomos en las cláusulas SQLP. Además, la transformación de programas y objetivos de QCFLP($\mathcal{D}, \mathcal{C}$) a CFLP($\mathcal{C}$) debe tener en cuenta el hecho de que las dos lógicas de reescritura QCRWL($\mathcal{D}, \mathcal{C}$) y CRWL($\mathcal{C}$) especifican que los símbolos de función definida se interpretan como funciones perezosas.

Por otro lado, al no haber relaciones de similaridad en QCFLP, el número final de reglas de programa transformadas será, en esencia, el mismo que el de reglas de programa original, i.e. cada regla se transformará en una única regla transformada. En resumen, dado un

QCFLP($\mathcal{D}, \mathcal{C}$)-programa $\mathcal{P} = \{R_1, \ldots, R_n\}$, obtendremos un CFLP($\mathcal{C}$)-programa

$$\mathcal{P}^{\mathcal{T}} = \{R_1^{\mathcal{T}}, \ldots, R_n^{\mathcal{T}}\} \uplus \{R_{\mathsf{qVal}}, R_{\mathsf{qBound}}\}$$

donde $R_i^{\mathcal{T}}$ es la transformación de cada regla de programa R_i de $\mathcal{P}$, y R_{qVal} y R_{qBound} serán, respectivamente, las reglas de programa que implementen en CFLP($\mathcal{C}$) dos funciones correspondientes a las $\mathcal{C}$-restricciones **qVal** y **qBound** encargadas de representar a $\mathcal{D}$ en $\mathcal{C}$.

Nota 15. En la presentación realizada en [13](A.4) no se precisó de manera suficientemente explícita la construcción de funciones definibles en CFLP($\mathcal{C}$) que correspondan a las restricciones **qVal** y **qBound** al realizar la transformación de programas. En la sección 6 se verá un tratamiento detallado de este asunto para el esquema SQCLP, el cual se puede adaptar muy fácilmente a QCFLP.

Formalmente, la transformación consta de 4 grupos de reglas de transformación que pueden verse en [13](A.4, Fig. 3). Estos grupos son: 1) reglas de transformación de expresiones; 2) reglas de transformación de qc-declaraciones; 3) reglas de transformación de reglas de programa; y 4) reglas de transformación de objetivos. En esta presentación nos centraremos en la explicación intuitiva de los cuatro grupos de reglas, relegando los detalles técnicos y formales de la transformación a la publicación [13](A.4) y el informe técnico [12](B.3) que la extiende.

Los resultados relativos a la transformación de programas y objetivos QCFLP a programas y objetivos equivalentes CFLP son los teoremas de corrección y completitud para la transformación de programas [13](A.4, T. 2) y objetivos [13](A.4, T. 3) que prueban la equivalencia semántica de ambos programas y objetivos (el original y su transformado) a partir de la derivabilidad en QCRWL($\mathcal{D}, \mathcal{C}$) y CRWL($\mathcal{C}$).

4.4.1. Transformación de expresiones

La transformación de una expresión e nos dará, en concreto, una terna

$$e^{\mathcal{T}} = (e', \Omega, \mathcal{W})$$

donde e' es también una expresión, Ω es un conjunto de restricciones de cualificación y $\mathcal{W}$ es el conjunto de variables de cualificación que aparecen en e' en la posición más exterior. Aunque normalmente $\mathcal{W}$ contendrá una única variable de cualificación, en general no podrá asegurarse.

La razón del conjunto Ω es la de asegurar que las variables de cualificación que aparecen en e' se instancien de manera satisfactoria, i.e. cumpliendo lo que dicta la lógica de la cualificación en cada caso. Y la del conjunto $\mathcal{W}$ es la de permitir el "enganche" de la expresión e' en una expresión o regla de programa mayor que contenga, a su vez, otras variables de cualificación que deban instanciarse de manera que sus valores de cualificación no excedan a aquellos valores de cualificación que instancien las variables de cualificación de e'. Por construcción de ambos conjuntos, bastará con que la expresión o regla de programa contenedora de e' establezca restricciones de cualificación adicionales entre las variables de cualificación que no aparecen en e' (que deberán instanciarse con valores de cualificación acotados superiormente en función de los valores de las variables de cualificación de e) y

las variables de cualificación más exteriores de e' (que aparecerán en el conjunto $\mathcal{W}$), dado que será el conjunto de restricciones de cualificación Ω de la expresión e' el que se encargue de asegurar eso mismo para las variables de cualificación exteriores de e' con respecto a las posibles variables de cualificación interiores de e'. Veamos un ejemplo:

Ejemplo 4.6 (Transformación de expresiones)**.** Sea e la siguiente expresión:

$$e = suma(\ suma(s(c), c),\ suma(s(s(c)), s(c))\ )$$

Al ser *suma* un símbolo de función definida, el transformado de nombre *suma'* tendrá un argumento más, el correspondiente a la variable de cualificación. El resto de símbolos, permanecen sin cambios, así:

$$e' = suma'(\ suma'(s(c), c, W_1),\ suma'(s(s(c)), s(c), W_2),\ W_3\ )$$

porque tanto s como c son símbolos de constructora de datos.

Dado que las subexpresiones $suma'(s(c), c, W_1)$ y $suma'(s(s(c)), s(c), W_2)$ son interiores a e', no tendría sentido que una sustitución ρ de variables de cualificación por valores de cualificación hiciera $W_3\rho \trianglerighteq W_1\rho$ ni tampoco $\mathcal{W}_3\rho \trianglerighteq W_2\rho$, porque estaría diciendo que es posible probar e' con un valor de cualificación mejor que con los que es posible probar sus subexpresiones. Por lo tanto, el conjunto de restricciones Ω deberá evitar este tipo de instanciaciones incluyendo dos restricciones de cualificación $W_3 \trianglelefteq W_1$ y $W_3 \trianglelefteq W_2$. Entonces:

$$\Omega = \{\mathbf{qVal}(W_1), \mathbf{qVal}(W_2), \mathbf{qVal}(W_3), \mathbf{qBound}(W_3, \mathbf{t}, W_1), \mathbf{qBound}(W_3, \mathbf{t}, W_2)\}$$

Nótese que las dos restricciones de cualificación **qBound** utilizan **t** como factor de atenuación porque en este caso no hay factor de atenuación que aplicar.

Por último, la variable que será necesaria "enganchar" cuando queramos utilizar e' en otra expresión mayor o en una regla de programa será W_3, por lo que $\mathcal{W} = \{W_3\}$ y entonces: $e^{\mathcal{T}} = (e',\ \Omega,\ \mathcal{W})$.

4.4.2. Transformación de qc-declaraciones

Las qc-producciones y los qc-átomos incondicionales se transforman mediante reglas basadas, de la manera obvia, en la transformación de expresiones. Así:

- $(e \to t)^{\mathcal{T}} = (e' \to t,\ \Omega,\ \mathcal{W})$ siempre que $e^{\mathcal{T}} = (e',\ \Omega,\ \mathcal{W})$.
- $(p(\overline{e}_n) == v)^{\mathcal{T}} = (p(\overline{e'}_n) == v,\ \bigcup_{i=1}^{n} \Omega_i,\ \bigcup_{i=1}^{n} \mathcal{W}_i)$ donde p es símbolo de función primitiva y v es variable, constante o valor básico, y siempre que $e_i^{\mathcal{T}} = (e_i',\ \Omega_i,\ \mathcal{W}_i)$ para $i = 1 \ldots n$.

Las qc-declaraciones $\varphi \sharp d \Leftarrow \Pi$ se transforman en c-declaraciones $\varphi' \Leftarrow \Pi'$ donde la parte condicional Π' incluye, además de Π, las restricciones de cualificiación Ω provenientes de φ' y otras restricciones de cualificación que aseguren que d esté acotado superiormente por los valores de cualificación con los que se prueben las variables de cualificación en φ'.

4.4.3. Transformación de reglas de programa

Las reglas de programa se transforman basándose en la transformación de la expresión del lado derecho y la transformación de las $\mathcal{C}$-restricciones del conjunto de condiciones de la regla. Así, una QCFLP-regla como

$$f(\overline{t}_n) \xrightarrow{\alpha} r \Leftarrow \delta_1,\ \ldots,\ \delta_m$$

se transforma en una CFLP-regla de la forma

$$\begin{aligned} f'(\overline{t}_n, W) \to r' \Leftarrow\ & \mathbf{qVal}(W), \\ & \Omega_r,\ (\ \mathbf{qBound}(W, \alpha, W')\ )_{W' \in \mathcal{W}_r}, \\ & (\ \Omega_i,\ (\ \mathbf{qBound}(W, \alpha, W')\ )_{W' \in \mathcal{W}_i},\ \delta_i'\ )_{i=1\ldots n} \end{aligned}$$

donde W es una variable de cualificación nueva, siempre que:

- $r^{\mathcal{T}} = (r',\ \Omega_r,\ \mathcal{W}_r)$, y
- $\delta_i^{\mathcal{T}} = (\delta_i',\ \Omega_i,\ \mathcal{W}_i)$ para $i = 1 \ldots m$.

En particular, el conjunto de condiciones finales de la CFLP-regla resultante es tal que:

1. obliga a que la nueva variable de cualificación W, que será la que devuelva el valor de cualificación final con el que se puede probar, en efecto, que $f(\overline{t}_n) \to r$, sea un valor aceptable de cualificación;
2. incluye las restricciones de cualificación que sean necesarias debido a la transformación del lado derecho r;
3. incluye las restricciones de cualificación adicionales necesarias que dicen que W no puede ser mejor que la atenuación con α de las variables de cualificación exteriores de r'; y por último
4. incluye, para cada condición δ del conjunto de condiciones original, las restricciones de cualificación que sean necesarias debido a la transformación de la condición δ de la regla orginal, incluye las restricciones de cualificación adicionales necesarias para indicar que el valor de W ha de estar acotado superiormente por los resultados de atenuar con α los valores de las variables de cualificación exteriores de la condición δ' e incluye la propia δ' que resulta de la transformación de la original δ.

Un caso particular es cuando el conjunto de condiciones en la regla original es vacío, en este caso, una QCFLP-regla como

$$f(\overline{t}_n) \xrightarrow{\alpha} r$$

se transforma en una CFLP-regla de la forma

$$f'(\overline{t}_n, W) \to r' \Leftarrow\ \mathbf{qVal}(W),\ \Omega_r,\ (\ \mathbf{qBound}(W, \alpha, W')\ )_{W' \in \mathcal{W}_r},\ \mathbf{qBound}(W, \alpha, \mathbf{t})$$

donde W es una variable de cualificación nueva, siempre que:

- $r^{\mathcal{T}} = (r',\ \Omega_r,\ \mathcal{W}_r)$.

Lo que tiene el efecto de asegurar que W no pueda ser mejor que el factor de atenuación de la QCFLP-regla en cuestión.

Además, cuando ocurre que $\mathcal{W}_r = \emptyset$, se tendrá también que $\Omega_r = \emptyset$ y por lo tanto las condiciones de la CFLP-regla transformada relativas al lado derecho se eliminan.

Como ilustración, considérese el siguiente ejemplo.

Ejemplo 4.7 (Transformación de programas)**.** Considérese el QCFLP($\mathcal{W}, \mathcal{R}$)-programa del ejemplo 4.2. Su CFLP($\mathcal{R}$)-programa equivalente es el siguiente:

1. $num(W) \to c \Leftarrow \mathbf{qVal}(W),\ \mathbf{qBound}(W, 0, 0)$
2. $num(W) \to s(num(W_1)) \Leftarrow \mathbf{qVal}(W),\ \mathbf{qVal}(W_1),\ \mathbf{qBound}(W, 1, W_1)$
3. $suma(c, X, W) \to X \Leftarrow \mathbf{qVal}(W),\ \mathbf{qBound}(W, 0, 0)$
4. $suma(s(X), Y, W) \to s(suma(X,Y,W_1)) \Leftarrow \mathbf{qVal}(W), \mathbf{qVal}(W_1),\ \mathbf{qBound}(W, 1, W_1)$
5. $\mathbf{qVal}(X) \to true \Leftarrow X \geq 0$
6. $\mathbf{qBound}(X, Y, Z) \to true \Leftarrow X \leq Y + Z$

Nótese que el programa traducido incluye dos reglas encargadas de definir las funciones **qVal** y **qBound**, que implementan las dos $\mathcal{C}$-restricciones de nombre idéntico que se corresponden con la representación del dominio de cualificación $\mathcal{D}$ dentro del dominio de restricciones $\mathcal{C}$, y que no hacían falta en el programa original.

4.4.4. Transformación de objetivos

La transformación de objetivos QCFLP en objetivos CFLP equivalentes se realiza transformando un objetivo $(\ \delta_i \sharp W_i,\ W_i \trianglerighteq \beta_i\ )_{i=1\ldots m}$ por medio de transformaciones individuales para cada $\mathcal{C}$-restricción atómica δ_i de forma que, si:

$$\delta_i^{\mathcal{T}} = (\delta_i',\ \Omega_i,\ \mathcal{W}_i)$$

entonces:

$$(\ \delta_i \sharp W_i, W_i \trianglerighteq \beta_i\ )^{\mathcal{T}} = \Omega_i,\ \mathbf{qVal}(W_i),\ (\mathbf{qBound}(W_i, \mathbf{t}, W))_{W \in \mathcal{W}_i},\ \mathbf{qBound}(\beta_i, \mathbf{t}, W_i),\ \delta_i'$$

para asegurar que cada variable de cualificación W_i se interpreta como un valor de cualificación acotado superiormente por la mayor cualificación calculada para δ_i y satisfaciendo también la condición umbral $W_i \trianglerighteq \beta_i$ impuesta en el objetivo.

Como ilustración, considérese el siguiente ejemplo.

Ejemplo 4.8 (Transformación de objetivos)**.** Sea G el QCFLP-objetivo

$$suma(s(c), X, s(s(c)))\sharp W_1\ [\!]\ W_1 \leq 5$$

para el QCFLP-programa $\mathcal{P}$ del ejemplo 4.2. Entonces el transformado $G^{\mathcal{T}}$ de G es

$$\mathbf{qVal}(W),\ \mathbf{qVal}(W_1),\ \mathbf{qBound}(W_1, 0, W),\ \mathbf{qBound}(5, 0, W_1),\ suma(s(c), X, s(s(c)), W)$$

dado que $(suma(s(c), X, s(s(c))))^{\mathcal{T}} = (suma(s(c), X, s(s(c)), W),\ \{\mathbf{qVal}(W)\},\ \{W\})$.

5. Programación lógica con restricciones, cualificación y proximidad

En esta sección se define un esquema genérico SQCLP cuyas instancias SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$) están parametrizadas por una relación de proximidad $\mathcal{S}$, un dominio de cualificación $\mathcal{D}$ y un dominio de restricciones $\mathcal{C}$; y se muestra cómo muchos de los esquemas anteriores propuestos pueden verse como casos particulares de SQCLP obtenidos mediante instanciación parciales de sus parámetros. Además, se presenta una semántica declarativa para SQCLP inspirada en la semántica de observables para CLP de [24, 23] que proporciona caracterizaciones de punto fijo y basadas en un cálculo lógico de los modelos mínimos, así como una noción de solución a objetivos que resulta independiente de implementación y que puede utilizarse para especificar el comportamiento esperado, en la práctica, de un sistema de resolución de objetivos.

Concretamente, el esquema SQCLP proporcionará:

1. programación con restricciones al estilo de CLP($\mathcal{C}$) de [35] para un dominio de restricciones $\mathcal{C}$ (véase 4.1);
2. programación cualificada con elementos de un dominio de cualificación $\mathcal{D}$ (véase 2.1) con atenuaciones α en las cláusulas de los programas y restricciones umbrales en los cuerpos de las mismas al estilo de las presentadas en la sección 2.2; y
3. relaciones de proximidad $\mathcal{S}$ como generalización de las relaciones de similaridad (véase 3.1) al relajar el axioma de transitividad.

Nota 16. Aquí se mantiene la notación $\mathcal{S}$ para las relaciones de proximidad y la letra S en el nombre del esquema a pesar de tratarse en este momento de una generalización de las relaciones de similaridad.

La publicación [61](A.5) detalla las principales características de la semántica declarativa de SQCLP, y el informe técnico [62](B.4) es una extensión detallada de aquella con explicaciones y resultados auxiliares y demostraciones completas todos sus resultados. Finalmente, mientras que el objeto de esta sección será la definición de la semántica declarativa del esquema SQCLP, la próxima sección se centrará en la implementación de SQCLP, mediante técnicas de transformación de programas al estilo de las vistas en las secciones 3.4 y 4.4, sobre los sistemas CLP($\mathcal{C}$) `SICStus Prolog` y `SWI-Prolog`.

5.1. Base computacional

El esquema SQCLP hace uso de los dominios de cualificación que se vieron en la sección 2.1; las relaciones de proximidad (como generalización de las relaciones de similaridad) vistas en la sección 3.1; y los dominios de restricciones vistos en la sección 4.1, una vez adaptados al marco de la programación lógica sin funciones.

En esencia, los dominios de cualificación no presentan diferencias con respecto a las vistas con anterioridad.

Las relaciones de proximidad se definen como relaciones de similaridad que pueden no cumplir el axioma de transitividad. Así, dado un dominio de cualificación $\mathcal{D}$ con conjunto

soporte D y un conjunto S, se define formalmente una *relación de proximidad $\mathcal{D}$-valuada* sobre el conjunto S como cualquier función $\mathcal{S} : S \times S \to D$ tal que los dos axiomas siguientes se satisfacen para todo x, y, z del conjunto S de la relación de proximidad:

- **Reflexividad.** $\mathcal{S}(x, x) = \mathbf{t}$.
- **Simetría.** $\mathcal{S}(x, y) = \mathcal{S}(y, x)$.

Nótese que la *identidad* $\mathcal{S}_{\mathsf{id}}$ tal y como fue definida en la sección 3.1 es también una relación de proximidad, y que las extensiones de una relación de proximidad a elementos construidos como términos y átomos se realizan también como en el caso de las relaciones de similaridad.

Y, por último, los dominios de restricciones se definen análogamente a como se ha visto en la sección 4.1, aunque adaptándolos a su uso en el contexto de un esquema de programación lógica puramente relacional. Así:

- *Términos construidos* t. Son: o bien una variable $X \in \mathcal{V}ar$; o bien un valor básico $u \in B$; o bien de la forma $c(\overline{t}_n)$ donde c es un símbolo de constructora de aridad n. Cuando $n = 0$, $c(\overline{t}_n)$ se escribe simplemente c.

Nota 17. Ahora no pueden ser el valor indefinido ($\bot$).

- *Átomos* A. Pueden ser: *átomos definidos* $r(\overline{t}_n)$, donde r es un símbolo de predicado definido; *átomos primitivos* $\kappa : p(\overline{t}_n)$, donde p es un símbolo de predicado primitivo; o *ecuaciones* $t_1 == t_2$, donde $==$ es el *símbolo de igualdad*, que no pertenece a la signatura.

La definición del dominio $\mathcal{R}$ de restricciones reales varía en el sentido de que su conjunto de operaciones primitivas $+$, $-$, ... y de comparaciones primitivas $>$, $\geq$, ... pasan a ser conjuntos de predicados de operación primitivos op_+, op_-, ... y predicados de comparación primitivos $cp_>$, $cp_\geq$, ... de aridad 3 y 2, respectivamente. A diferencia del esquema QCFLP, en SQCLP las primitivas de cualquier dominio de restricciones se representan por medio de predicados, por lo que no pueden anidarse. Por ejemplo, una restricción que podría escribirse en QCFLP como:

$$Y > 2 * (X + 1)$$

debe escribirse en SQCLP del siguiente modo:

$$\exists Z_1 \exists Z_2 (op_+(X, 1, Z_1) \wedge op_*(2, Z_1, Z_2) \wedge cp_>(Y, Z_2))$$

en la que el tercer argumento de las operaciones op_+ y op_* es donde devuelven el resultado. Para una definición más formal, véase [62](B.4, Def. 2.6).

Para terminar con la base computacional del esquema SQCLP, se introducen tres nociones auxiliares que resultan de interés al permitir comparar, a distintos niveles, la igualdad entre elementos de la signatura. Dado un dominio de restricciones $\mathcal{C}$ y una relación de proximidad $\mathcal{S}$, se dice:

- que t y s son Π-*equivalentes*, y se escribe $t \approx_\Pi s$, cuando se cumpla $\Pi \models_{\mathcal{C}} t == s$, véase [62](B.4, Lem. 2.2) para propiedades de $\approx_\Pi$;

- que t y s son *próximos a nivel* λ, y se escribe $t \approx_\lambda s$, cuando se cumpla $\mathcal{S}(t,s) \trianglerighteq \lambda$, véase [62](B.4, Lem. 2.5) para propiedades de $\approx_\lambda$; y
- que t y s son Π-próximos a nivel λ, y se escribe $t \approx_{\lambda,\Pi} s$, cuando se cumpla que existen $\hat{t}$ y $\hat{s}$ tales que $t \approx_\Pi \hat{t}$, $s \approx_\Pi \hat{s}$ y $\hat{t} \approx_\lambda \hat{s}$, véase [62](B.4, Lem. 2.7) para propiedades de $\approx_{\lambda,\Pi}$.

5.2. Sintaxis

En el esquema SQCLP interesarán instancias SQCLP($\mathcal{S},\mathcal{D},\mathcal{C}$) que cumplan ciertas condiciones de *admisibilidad* para la terna $\langle \mathcal{S},\mathcal{D},\mathcal{C}\rangle$ que, en esencia, no son más que la unión de las condiciones de admisibilidad que se vieron para los esquemas SQLP y QCFLP, es decir:

1. que la relación $\mathcal{S}$ sea una relación de proximidad $\mathcal{D}$-valuada; y
2. que el dominio de cualificación $\mathcal{D}$ sea representable en el dominio de restricciones $\mathcal{C}$.

Por lo tanto, se dirá que $\langle \mathcal{S},\mathcal{D},\mathcal{C}\rangle$ es una terna admisible cuando cumpla las condiciones (1) y (2).

Nota 18. La noción de representabilidad de un dominio de cualificación en un dominio de restricciones ha sido refinada, con respecto a como se presentaba en publicaciones anteriores, por medio de una función inyectiva $\imath$ que asigna, a cada valor de cualificación (tal vez excepto **b**) un elemento del conjunto soporte del dominio de restricciones. En el informe técnico [62](B.4, §2.2.5) puede verse este refinamiento con más detalle.

Al igual que en secciones anteriores, un SQCLP($\mathcal{S},\mathcal{D},\mathcal{C}$)-programa $\mathcal{P}$ será un conjunto de cláusulas cualificadas de la forma

$$H \overset{\alpha}{\leftarrow} B_1 \sharp w_1,\ \ldots,\ B_m \sharp w_m$$

en la que H, B_1, $\ldots$, B_m son átomos, α es un factor de atenuación de $\mathcal{D}$ distinto de **b** y los w_1, $\ldots$, w_m son o bien elementos de $\mathcal{D}$ distintos de **b**, o bien "?", y representan la restricción umbral con la que cada átomo del cuerpo debe poder probarse. La intuición que hay detrás de una SQCLP-cláusula cualificada es la siguiente: si para todo $1 \leq j \leq m$ se tiene que puede probarse $B_j \sharp e_j$ para un valor de cualificación e_j satisfactorio, i.e. $e_j \trianglerighteq^? w_j$, entonces podrá inferirse $A \sharp d$, para cualquier átomo A que sea Π-próximo a nivel δ de la cabeza H de la cláusula, siempre que el valor de cualificación d esté acotado superiormente por el nivel δ de Π-proximidad y por el ínfimo de las atenuaciones con α de los valores de cualificación e_j con los que se probaron los B_j.

Un ejemplo motivador utilizado en las publicaciones es el siguiente:

Ejemplo 5.1 (SQCLP: Obras)**.** Sea $\mathcal{S}$ como aparece más adelante, $\mathcal{D} = \mathcal{U} \otimes \mathcal{W}$ y $\mathcal{C} = \mathcal{R}$. Entonces:

1 $\quad buena\_obra(X) \overset{(0.75,3)}{\leftarrow} famoso(Y)\sharp(0.5,100),\ escribió(Y,X)\sharp?$

2 $\quad famoso(shakespeare) \overset{(0.9,1)}{\leftarrow}$

3 $\quad escribió(shakespeare, king\_lear) \overset{(1,1)}{\leftarrow}$

$\quad \mathcal{S}(king\_lear, king\_liar) = (0.8, 2)$

es un SQCLP($\mathcal{S}$,$\mathcal{U}\otimes\mathcal{W}$,$\mathcal{R}$)-programa $\mathcal{P}$ en el que las cualificaciones (d, e) son tales que d representa el grado de certidumbre con el que confiamos en la veracidad de la información y e el coste de obtener (o demostrar) dicha información. Así, la restricción umbral $(0.5, 100)$ que aparece en el cuerpo de la cláusula 1 nos sirve para indicar que estamos interesados sólo en utilizar dicha cláusula cuando Y tome el valor de una persona famosa con certidumbre de al menos 0.5 y que no cueste obtenerla más de 100 unidades.

5.3. Semántica declarativa

Al igual que se hizo en la sección 2.3, y con el objetivo de ser más completos y formales, se desarrollan a continuación las dos caracterizaciones alternativas de la semántica declarativa más conocidas: una semántica de punto fijo, y otra basada en un cálculo lógico, ambas inspiradas por [24, 23]. Estas dos publicaciones proponían tres semánticas para programas $\mathcal{S}_1$, $\mathcal{S}_2$ y $\mathcal{S}_3$ diferentes que caracterizaban *soluciones válidas cerradas para objetivos*, *soluciones válidas abiertas para objetivos* y *respuestas calculadas para objetivos* en CLP, respectivamente. En la publicación a la que se corresponde esta sección, se desarrolla una semántica al estilo de la $\mathcal{S}_2$ que permite caracterizar soluciones válidas abiertas para SQCLP-objetivos y que se expone a continuación. Como base de la semántica, se trabajará con *qc-átomos* de la forma

$$A\sharp d \Leftarrow \Pi$$

en los que se pretende asertar que el átomo A es consecuencia del conjunto de $\mathcal{C}$-restricciones Π con valor de cualificación d. Además, se utilizará también una noción especial de implicación con la intención de capturar algunas implicaciones entre qc-átomos cuya validez no depende de la relación de proximidad $\mathcal{S}$ ni de la semántica de los predicados definidos. Estas nociones son:

- Un qc-átomo $A\sharp d \Leftarrow \Pi$ se dice que es *definido*, *primitivo* o *ecuacional* según sea la forma del átomo A.
- Un qc-átomo $A\sharp d \Leftarrow \Pi$ se dice que es *observable* cuando d es distinto de **b** y Π es satisfactible.
- Dados dos qc-átomos $\varphi : A\sharp d \Leftarrow \Pi$ y $\varphi' : A'\sharp d' \Leftarrow \Pi$, se dice que φ $\langle\mathcal{D},\mathcal{C}\rangle$-implica φ', y se escribe $\varphi \succcurlyeq_{\mathcal{D},\mathcal{C}} \varphi'$, sii existe una sustitución θ tal que: (1) $A' = A\theta$; (2) $d' \preccurlyeq d$; y (3) $\Pi' \models_{\mathcal{C}} \Pi\theta$.

En lo que sigue, se centra la atención en qc-átomos observables porque podrán ser interpretados como observaciones de soluciones abiertas válidas para objetivos atómicos en SQCLP($\mathcal{S},\mathcal{D},\mathcal{C}$) como se verá en la subsección 5.3.4.

Como ya ocurriera en la sección 2, las nociones de *interpretación*, *consecuencia inmediata* y *modelo* son coincidentes para ambas caracterizaciones de la semántica declarativa, y son el objeto de la siguiente subsección.

5.3.1. Interpretaciones y modelos

Las *qc-interpretaciones* (o interpretaciones) $\mathcal{I}$ en SQCLP son conjuntos de qc-átomos definidos observables cerrados bajo $\langle \mathcal{D}, \mathcal{C} \rangle$-implicación.

A diferencia de la noción de implicación empleada en la semántica del esquema SQLP, en esta la relación de proximidad no juega ningún papel. El motivo es que el cierre de un conjunto de átomos con una relación de implicación es transitivo por naturaleza, y dado que la relación de proximidad no tiene porqué ser transitiva obligaría a probar más átomos de los que pueden probarse directamente con la relación de proximidad. Es decir, considérese la instancia SQCLP($\mathcal{S}, \mathcal{U}, \mathcal{R}$) donde la relación de proximidad $\mathcal{S}$ es tal que $\mathcal{S}(a, b) = 0.8$, $\mathcal{S}(b, c) = 0.8$. Y sea $\mathcal{S}(a, c) = 0$[2] y el qc-átomo $a \sharp 1 \Leftarrow \emptyset$ un qc-átomo perteneciente a una interpretación $\mathcal{I}$ del esquema SQCLP. Debido a la noción de $\langle \mathcal{D}, \mathcal{C} \rangle$-implicación, y dado que $\mathcal{I}$ que contiene a $a \sharp 1 \Leftarrow \emptyset$, deberá contener también a cualquier qc-átomo más particular que él mismo o con peor cualificación, así que tendremos también que cualquier qc-átomo de la forma $a \sharp \lambda \Leftarrow \emptyset$ con $0 < \lambda \leq 1$ pertenece también a $\mathcal{I}$ por ser más particular y con peor cualificación que $a \sharp 1 \Leftarrow \emptyset$.

Ahora bien, si la relación de $\langle \mathcal{D}, \mathcal{C} \rangle$-implicación tuviera en cuenta la proximidad, se tendría que si $a \sharp 1 \Leftarrow \emptyset$, entonces también $b \sharp 0.8 \Leftarrow \emptyset$ porque se consideraría más particular que $a \sharp 1 \Leftarrow \emptyset$, y si se quisiera hacer el cierre de esa implicación, deberían introducirse también las implicaciones de este nuevo qc-átomo, haciendo que, de hecho, $c \sharp 0.8 \Leftarrow \emptyset$ pertenezca a $\mathcal{I}$, por ser a su vez más particular que $b \sharp 0.8 \Leftarrow \emptyset$, y este más particular que $a \sharp 1 \Leftarrow \emptyset$, a pesar de que la relación de proximidad dice expresamente que $\mathcal{S}(a, c) = 0$. En consecuencia, no resulta aceptable decir que un átomo próximo a otro es "más particular" que el primero, y por lo tanto, no debe deducirse directamente de que un átomo pertenezca a una interpretación que sus átomos próximos también pertenezcan a ella.

La noción de implicación utilizada en la sección 3 y sus publicaciones asociadas [11](A.3) y [14](B.2) sí dependía de $\mathcal{S}$, y ello no causaba problemas técnicos debido a que $\mathcal{S}$ se suponía transitiva. No obstante, también hubiese sido factible definir una semántica de SQLP basada en una implicación entre átomos independiente de $\mathcal{S}$.

Como se habrá observado, se ha dicho que existen qc-átomos definidos, primitivos y ecuacionales y, sin embargo, las interpretaciones se han definido como conjuntos de qc-átomos observables definidos. Esto se ha hecho así porque la validez de los átomos primitivos depende únicamente del conjunto de $\mathcal{C}$-restricciones, y la validez de las ecuaciones depende únicamente de la relación de proximidad $\mathcal{S}$ y del conjunto de restricciones $\mathcal{C}$. Formalmente, se dice que un qc-átomo observable φ es válido en una interpretación $\mathcal{I}$, y se escribe como $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$, sii se da alguno de los tres casos siguientes:

- φ es definido y pertenece a $\mathcal{I}$;
- φ es de la forma $\kappa \sharp d \Leftarrow \Pi$ primitivo, y es consecuencia de Π, i.e. $\Pi \models_{\mathcal{C}} \kappa$;
- φ es de la forma $(t == s) \sharp d \Leftarrow \Pi$ ecuacional, y t y s son Π-próximos a nivel d, i.e. $t \approx_{d,\Pi} s$.

La noción de *consecuencia inmediata* simplifica la posterior definición de la noción de modelo y de las caracterizaciones de la semántica declarativa. Intuitivamente, son consecuencias inmediatas de una interpretación dada, con respecto a la instancia de una cláusula

[2]Nótese que $\mathcal{S}$ no es transitiva porque no se tiene $\mathcal{S}(a, c) \trianglerighteq \mathcal{S}(a, b) \sqcap \mathcal{S}(b, c) = 0.8$.

lógica cualificada también dada, aquellos qc-átomos $A\sharp d \Leftarrow \Pi$ que son próximos a nivel d_0 a la cabeza de la cláusula instanciada, entendiendo que un qc-átomo es próximo a nivel λ a la cabeza de una cláusula cuando el símbolo de predicado de ambos elementos es próximo a dicho nivel, y puede probarse que los argumentos del qc-átomo son Π-próximos a nivel λ a los argumentos de la cabeza de la cláusula, y además:

- el cuerpo de la cláusula instanciada es válido en la interpretación para valores de cualificación adecuados, i.e. mejores que sus restricciones umbrales respectivas; y
- el valor de cualificación d del qc-átomos es peor o igual que el ínfimo entre λ y de la atenuación de los valores de cualificación empleados para probar los átomos del cuerpo.

Nota 19. Otra diferencia con respecto al esquema SQLP es que los valores de cualificación que son debidos a la relación de proximidad no se atenúan con el factor de atenuación de la cláusula aplicada, dado que la información que proporcionan no es debida a la implicación de la cláusula sino al parecido de un elemento que ya ha sido probado fehacientemente para un determinado grado de cualificación.

Es decir, son consecuencias inmediatas de una interpretación dada via una cláusula de programa también dada, aquellos qc-átomos que son próximos a la cabeza de la cláusula instanciada cuyo cuerpo es válido en la interpretación para un valor de cualificación mejor que con el que se prueba la cabeza. Formalmente, dadas una interpretación $\mathcal{I}$ y una cláusula:

$$C \ : \ p(\overline{t}_n) \overset{\alpha}{\Leftarrow} B_1\sharp w_1, \ \ldots, \ B_m\sharp w_m$$

se dice que un qc-átomo observable $p'(\overline{t'}_n)\sharp d \Leftarrow \Pi$ es *consecuencia inmediata* de $\mathcal{I}$ con respecto a C, sii existe una sustitución θ y una serie de valores de cualificación $d_0, d_1, \ldots, d_n, e_1, \ldots, e_m$ —distintos de **b**— que cumplen (1), (2), (3) y (4) como siguen:

(1) $\mathcal{S}(p', p) = d_0$.

(2) $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} (t'_i == t_i\theta)\sharp d_i \Leftarrow \Pi$ para $i = 1 \ldots n$.

(3) $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} B_j\theta\sharp e_j \Leftarrow \Pi$ con $e_j \trianglerighteq^? w_j$ para $j = 1 \ldots m$.

(4) $d \trianglelefteq d_i \ \ (0 \leq i \leq n)$ y $d \trianglelefteq \alpha \circ e_j \ (1 \leq j \leq m)$.

Nótese que el nivel λ empleado en la intuición de la noción de consecuencia inmediata vendría determinado por las condiciones (1) y (2) y sería equivalente a $d_0 \sqcap d_1 \sqcap \cdots \sqcap d_n$, y que en la condición (4) se pide, en su primera parte, precisamente que $d \trianglelefteq \lambda$.

Finalmente:

- una interpretación $\mathcal{I}$ es *modelo de una cláusula* C, sii todo qc-átomo definido observable φ que es consecuencia inmediata de $\mathcal{I}$ via C verifica que $\varphi \in \mathcal{I}$; y
- una interpretación $\mathcal{I}$ es *modelo de un programa* $\mathcal{P}$ sii es modelo de toda cláusula C del programa $\mathcal{P}$.

5.3.2. Semántica de punto fijo

En primer lugar, es posible demostrar que el conjunto de interpretaciones forma un retículo completo con respecto al orden de inclusión como se prueba en [62](B.4, Prop. 3.2). A continuación se define el *transformador de interpretaciones* $\mathsf{T}_{\mathcal{P}}$ con la intención de obtener todas las posibles consecuencias inmediatas de una interpretación $\mathcal{I}$, con respecto a las cláusulas C de un programa $\mathcal{P}$. Así:

$$\mathsf{T}_{\mathcal{P}}(\mathcal{I}) =_{\text{def}} \{\varphi \mid \varphi \text{ es consecuencia inmediata de } \mathcal{I} \text{ vía alguna } C \in \mathcal{P}\}$$

donde la noción de consecuencia inmediata es como se definió en la subsección anterior. Además, en este caso también, son propiedades del transformador de interpretaciones es ser una función bien definida, mónotona, continua y cuyos puntos prefijos son los modelos $\mathcal{P}$, i.e. para toda interpretación $\mathcal{I}$, se tiene que $\mathcal{I}$ es modelo de $\mathcal{P}$ sii $\mathsf{T}_{\mathcal{P}}(\mathcal{I}) \subseteq \mathcal{I}$.

Finalmente, la caracterización del modelo mínimo $\mathcal{M}_{\mathcal{P}}$ de $\mathcal{P}$ coincide, como cabría esperar, con el menor punto fijo (*lfp*) del operador $\mathsf{T}_{\mathcal{P}}$, i.e.:

$$\mathcal{M}_{\mathcal{P}} = \mathit{lfp}(\mathsf{T}_{\mathcal{P}}) = \bigcup_{k \in \mathbb{N}} \mathsf{T}_{\mathcal{P}}{\uparrow}^k(\emptyset) \ .$$

5.3.3. Semántica basada en un cálculo lógico

Esta semántica es también una extensión de las presentadas anteriormente para los casos de los esquemas BQLP y SQLP. Sin embargo, dado que ahora nos podemos encontrar con qc-átomos primitivos y ecuacionales, tendremos dos reglas adicionales de inferencia en el cálculo para tratar estos casos de manera adecuada. El cálculo de la *lógica de Horn con restricciones, cualificación y proximidad*, i.e. SQCHL$(\mathcal{S}, \mathcal{D}, \mathcal{C})$, puede verse completo en la figura 5.

SQDA $$\frac{\left(t'_i == t_i\theta \sharp d_i \Leftarrow \Pi\right)_{i=1\ldots n} \quad \left(B_j\theta \sharp e_j \Leftarrow \Pi\right)_{j=1\ldots m}}{p'(\overline{t'}_n)\sharp d \Leftarrow \Pi}$$

si $(p(\overline{t}_n) \overset{\alpha}{\Leftarrow} B_1 \sharp w_1, \ldots, B_m \sharp w_m) \in \mathcal{P}$, θ sust., $\mathcal{S}(p', p) = d_0 \neq \mathbf{b}$, $e_j \trianglerighteq^? w_j \ \ (1 \leq j \leq m)$ y $d \trianglelefteq \prod_{i=0}^{n} d_i \sqcap \alpha \circ \prod_{j=1}^{m} e_j$.

SQEA $$\frac{}{(t == s)\sharp d \Leftarrow \Pi} \quad \text{si } t \approx_{d,\Pi} s.$$

SQPA $$\frac{}{\kappa \sharp d \Leftarrow \Pi} \quad \text{si } \Pi \models_{\mathcal{C}} \kappa.$$

Figura 5: Cálculo lógico SQCHL$(\mathcal{S}, \mathcal{D}, \mathcal{C})$

En el cálculo SQCHL, la regla **SQDA** resulta de la extensión de la regla *modus ponens* de los cálculos de inferencia lógica que hems visto con anterioridad. En ella, se hace necesaria utilizar una cláusula del programa y encontrar una sustitución θ y una serie de valores de cualificación d_i (para las igualdades de los argumentos de la cabeza) y e_j (para el cuerpo de

la cláusula). Nótese que en este caso, el primer grupo de premisas se encargan de probar que los argumentos de la cabeza de la cláusula instanciada por θ son, en efecto, Π-próximos a los argumentos del qc-átomo que se está probando, cosa que, junto a la petición de que los símbolos principales de predicado de ambos elementos sean próximos, prueban que el propio qc-átomo es próximo a una instanciación válida de la cabeza de la cláusula con sustitución θ a un nivel que resulta, como ya se ha comentado, de calcular el ínfimo $d_0 \sqcap d_1 \sqcap \cdots \sqcap d_n$. El otro grupo de premisas, es el usual para probar el cuerpo de la cláusula una vez instanciado por la sustitución θ.

Las otras dos reglas del cálculo se utilizan para permitir probar qc-átomos ecuacionales (caso de la regla **SQEA**) y qc-átomos primitivos (regla **SQPA**). Estas dos reglas resultan bastante sencillas de entender si se tiene en cuenta que son, respectivamente, equivalentes a la noción de Π-proximidad a nivel d y deducibilidad a partir de Π.

De forma análoga a como se ha hecho para los cálculos de inferencia anteriores, la notación $\mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ indica que el qc-átomo φ puede inferirse a partir de las cláusulas de $\mathcal{P}$ en la lógica SQCHL en un número finito de pasos de inferencia. Si fuera necesario hacer explícito el número n de pasos de inferencia, utilizaríamos la notación $\mathcal{P} \vdash^{n}_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$. Nótese también en este caso la diferencia entre $\vdash_{\mathcal{S},\mathcal{D},\mathcal{C}}$ y las inferencias $\vdash_{\mathcal{S},\mathcal{D}}$ y $\vdash_{\mathcal{D}}$ correspondientes a los cálculos de inferencia lógica para los esquemas SQLP y QLP (o BQLP), respectivamente.

Los árboles de inferencia SQCHL presentan una forma algo más complicada que en los casos anteriores por el motivo de presentar reglas adicionales de inferencia lógica y por hacer explícito las igualdades entre los argumentos de los qc-átomos que se prueban y las cláusulas empleadas. Sin embargo, no debe resultar excesivamente complejo desarrollar una demostración en SQCHL partiendo de la base de que las ideas fundamentales coinciden con la de los cálculos de inferencia lógica anteriormente presentados. Como ilustración, considérese el siguiente ejemplo.

Ejemplo 5.2 (Inferencia en SQCHL)**.** Sean $\mathcal{S}$ y $\mathcal{P}$ como en el ejemplo 5.1. Entonces, el siguiente árbol de inferencia prueba que el qc-átomo

$$buena\_obra(kli)\sharp(0.64, 5) \Leftarrow \emptyset$$

es, en efecto, derivable de $\mathcal{P}$ en la lógica SQCHL, cuando las constantes *shakespeare*, *king_lear* y *king_liar* han sido reemplazadas, respectivamente, por *sha*, *kle* y *kli*.

$$\dfrac{\dfrac{}{(kli == kle)\sharp(0.8, 2) \Leftarrow \emptyset}\,(4) \quad \dfrac{\dfrac{}{(sha == sha)\sharp(1, 0) \Leftarrow \emptyset}\,(5)}{famoso(sha)\sharp(0.9, 1) \Leftarrow \emptyset}\,(2) \quad (\spadesuit)}{buena\_obra(kli)\sharp(0.64, 5) \Leftarrow \emptyset}\,(1)$$

con:

$$(\spadesuit)\ \dfrac{\dfrac{}{(sha == sha)\sharp(1, 0) \Leftarrow \emptyset}\,(5) \quad \dfrac{}{(kle == kle)\sharp(1, 0) \Leftarrow \emptyset}\,(6)}{escribió(sha, kle)\sharp(1, 1) \Leftarrow \emptyset}\,(3)$$

y donde casa paso de inferencia es como sigue:

(1) Inferencia **SQDA** con cláusula 1, sustitución $\{X \mapsto kle, Y \mapsto sha\}$ y donde

$$
\begin{aligned}
(0.6,5) &\trianglelefteq \bigsqcap\{(0.8,2)\} \sqcap (0.75,3) \circ \bigsqcap\{(0.9,1),(1,1)\} \\
&= (0.8,2) \sqcap (0.675,4) \\
&= (0.675,4)\ .
\end{aligned}
$$

(2) Inferencia **SQDA** con cláusula 2, sustitución ε y donde

$$
\begin{aligned}
(0.9,1) &\trianglelefteq \bigsqcap\{(1,0)\} \sqcap (0.9,1) \circ \mathbf{t} \\
&= (1,0) \sqcap (0.9,1) \\
&= (0.9,1)\ .
\end{aligned}
$$

(3) Inferencia **SQDA** con cláusula 3, sustitución ε y donde

$$
\begin{aligned}
(1,1) &\trianglelefteq \bigsqcap\{(1,0),(1,0)\} \sqcap (1,1) \circ \mathbf{t} \\
&= (1,0) \sqcap (1,1) \\
&= (1,1)\ .
\end{aligned}
$$

(4) Inferencia **SQEA** con $kli \approx_{(0.8,2),\emptyset} kle$.

(5) Inferencia **SQEA** con $sha \approx_{(1,0),\emptyset} sha$.

(6) Inferencia **SQEA** con $kle \approx_{(1,0),\emptyset} kle$.

Finalmente, dado un SQCLP$(\mathcal{S},\mathcal{D},\mathcal{C})$-programa $\mathcal{P}$ es posible demostrar, véase [62](B.4, T. 3.2), que su modelo mínimo $\mathcal{M}_{\mathcal{P}}$ es

$$\mathcal{M}_{\mathcal{P}} = \{\varphi \mid \varphi \text{ es definido observable y } \mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi\}$$

es decir, es el conjunto de qc-átomos definidos observables que pueden probarse en SQCHL a partir de las cláusulas del programa $\mathcal{P}$.

5.3.4. Objetivos y soluciones

Un SQCLP-objetivo tendrá la forma

$$G \ :\ A_1 \sharp W_1,\ \ldots,\ A_m \sharp W_m \,[\!]\, W_1 \rhd^? \beta_1,\ \ldots,\ W_m \rhd^? \beta_m$$

o de manera más abreviada $\left(A_i \sharp W_i,\ W_i \rhd^? \beta_i\right)_{i=1\ldots m}$. Y, dado un SQCLP-programa $\mathcal{P}$, y un objetivo G para $\mathcal{P}$, se dice que $\langle \sigma, \mu, \Pi \rangle$ es una solución para G cuando σ es una sustitución de variables por términos, μ es una sustitución de variables de cualificación a términos que representan elementos del dominio de cualificación (recuérdese que en este esquema se hacía uso de una función inyectiva $\imath$ de valores de cualificación en términos) y Π es un conjunto de $\mathcal{C}$-restricciones y se cumplen, para $1 \leq i \leq m$, las condiciones (1) y (2) como siguen:

(1) $W_i\mu = d_i \rhd^? \beta_i$.

(2) $\mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} A_i\sigma \sharp W_i\mu \Leftarrow \Pi$.

Ejemplo 5.3 (Objetivo y solución)**.** Sean $\mathcal{S}$ y $\mathcal{P}$ como en el ejemplo 5.1. Entonces,

$$G \ : \ buena\_obra(X) \sharp W \ [] \ W \trianglerighteq^{?} (0.6, 5)$$

es un objetivo para $\mathcal{P}$ y $\langle \{X \mapsto king\_liar\}, \{W \mapsto (0.675, 4)\} \rangle$ una solución para G.

5.4. Casos particulares

A continuación se detalla una serie de esquemas de programación que pueden obtenerse mediante instanciación parcial de los parámetros del esquema SQCLP y también de la no utilización de alguno de los elementos disponibles en el esquema. Un listado de estas características con mayor detalle y profundidad aparece en [61](A.5, §4).

Haciendo uso de la relación de proximidad/similaridad $\mathcal{S}_{\text{id}}$ de identidad, del dominio de cualificación $\mathcal{B}$ de los valores booleanos y del dominio de restricciones $\mathcal{R}$ de las restricciones reales pueden obtenerse los siguientes esquemas de programación:

1. El esquema de la programación lógica con restricciones y cualificación, QCLP; con instancias QCLP($\mathcal{D}, \mathcal{C}$) = SQCLP($\mathcal{S}_{\text{id}}, \mathcal{D}, \mathcal{C}$).

2. El esquema de la programación lógica con cualificación y proximidad (o similaridad), SQLP; con instancias SQLP($\mathcal{S}, \mathcal{D}$) = SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{R}$). El esquema del mismo nombre que se vio en la sección 3 puede verse, en realidad, como una forma restringida de esta, en la que las cláusulas no contienen en sus cuerpos restricciones de $\mathcal{R}$, ni restricciones umbrales distintas de "?".

3. El esquema de la programación lógica con cualificación, QLP; con instancias QLP($\mathcal{D}$) = SQCLP($\mathcal{S}_{\text{id}}, \mathcal{D}, \mathcal{R}$) siempre que no se haga uso de restricciones de $\mathcal{R}$ en los cuerpos de las cláusulas.

4. El esquema de la programación lógica con restricciones, CLP; con instancias CLP($\mathcal{C}$) = SQCLP($\mathcal{S}_{\text{id}}, \mathcal{B}, \mathcal{C}$).

5. La programación lógica clásica, LP; que puede verse, cuando $\mathcal{H}$ es el *dominio de restricciones de Herbrand*, como la instancia SQCLP($\mathcal{S}_{\text{id}}, \mathcal{B}, \mathcal{H}$) del esquema SQCLP.

Además, el esquema de la programación lógica basada en similaridad, SLP; con instancias SLP($\mathcal{S}$) = SQCLP($\mathcal{S}, \mathcal{U}, \mathcal{R}$). Las formulaciones SLP de otros autores como en [64, 39] puede verse como una formulación restringida de la presente de modo que en estos programas no aparezcan factores de atenuación diferentes de 1, restricciones umbrales diferentes de "?" y restricciones de $\mathcal{R}$.

6. Implementación práctica

Las secciones precedentes y las publicaciones que las acompañan no han abordado la presentación de implementaciones efectivas de los diversos esquemas de programación declarativa cualificada que se han investigado. Pese a ello, todas las investigaciones y publicaciones de

esta tesis han ido siempre acompañadas de un esfuerzo importante por proporcionar, como mínimo, facilidades para probar la ejecución de ejemplos representativos; y en el caso del esquema SQCLP un prototipo de implementación cuidadosamente desarrollado que soporta varias instancias significativas para SQCLP y otros esquemas que resultan de instanciaciones parciales de SQCLP admitidas en el prototipo. La implementabilidad de los esquemas investigados ha sido siempre un factor determinante que ha influido en la presentación de las publicaciones y en la formulación de la semántica declarativa, en particular, en la definición de las interpretaciones como conjunto de observaciones (en la práctica, cada una será el resultado de una posible computación), la definición de las restricciones de cualificación (en la práctica, definidas desde un primer momento como restricciones de $\mathcal{R}$), etc.

Es por ello objeto de esta sección mostrar, de un modo general pero claro, el trabajo y esfuerzo realizado en el aspecto más práctico del desarrollo de todos estos esquemas de programación y que, generalmente, ha quedado relegado a posiciones menos relevantes en las publicaciones formales.

6.1. El sistema `(S)QCLP`

Para SQCLP se ha propuesto, en [15](B.5), un método de implementación basado en una técnica de transformación de programas que ha dado lugar a una implementación práctica que se concreta en el sistema `(S)QCLP`.

El desarrollo realizado ha aprovechado las experiencias adquiridas en los trabajos previos de implementación de QLP, SQLP y QCFLP, así como las técnicas de transformación de programas semánticamente correctas obtenidas en las publicaciones ya descritas en la sección 5. Teniendo en cuenta la expresividad del esquema SQCLP y algunas elecciones significativas de dominios con los que instanciar sus parámetros, se decidió desarrollar un prototipo que diese soporte a lo siguiente:

- programación lógica cualificada para dominios de cualificación construidos mediante producto estricto de los dominios de cualificación `b`, `u`, `w` como versión implementada de los dominios $\mathcal{B}$, $\mathcal{U}$ y $\mathcal{W}$, respectivamente;
- programación lógica basada en proximidad (o similaridad cuando la relación de proximidad satisface el axioma de transitividad) implementando incluso el predicado $==$ como versión implementada de la noción de Π-proximidad;
- programación lógica con restricciones al estilo CLP aunque con sintaxis reducida —mediante predicados primitivos como se ha explicado en el final de la subsección 5.1 dedicada a la base computacional del esquema SQCLP—;
- restricciones umbrales en los cuerpos de las cláusulas; y
- restricciones umbrales opcionales en los objetivos (es decir, con la posibilidad de utilizar el símbolo "?" también en los objetivos).

De todo ello, resultaba que el sistema `(S)QCLP` sería capaz de transformar y ejecutar programas para los esquemas SQCLP, QCLP, SQLP, QLP y SLP, que como se ha visto pueden todos verse como casos particulares de SQCLP, y resolver objetivos para cada uno de ellos.

Además, un paso posterior incorporó algunos elementos para mejorar la eficiencia final del sistema, dando lugar a:

1. Capacidad para ejecutar programas QCLP mediante una transformación optimizada, es decir, sin necesidad de asumir que existía la relación de proximidad identidad y reduciendo, por tanto, el número de cláusulas finales. Es por esta característica por la que el sistema recibe el nombre de `(S)QCLP` porque en el caso de no establecer una relación de proximidad en un determinado programa, hace que se considere al programa como perteneciente al esquema QCLP en lugar de al esquema SQCLP, afectando esto únicamente a la eficiencia en la ejecución.

2. Capacidad para mostrar las soluciones estableciendo el vínculo entre las variables de cualificación y su valor máximo posible de acuerdo con el conjunto de restricciones de la solución, o no.

Nota 20. En su versión actual, el *flag* que determina este comportamiento no está disponible para el usuario final dado que debe modificarse en el código fuente del sistema y, por tanto, el comportamiento por defecto es el de establecer el vínculo y no mostrar el conjunto de restricciones finales. El motivo para actuar de esta forma es que `SWI-Prolog` no muestra soluciones al objetivo `clpr`

$$X > 0,\ X \leq 0.8,\ Y > 0,\ Y \leq 1,\ X \leq 0.9 \times Y$$

como demuestra lo siguiente:

```
?- {X>0, X=<0.8, Y>0, Y=<1, X=<0.9*Y}.
^CAction (h for help) ? abort
% Execution Aborted
?- {X>0, X=<0.8, Y>0, Y=<1, X=<0.9*Y},fail.
false.
```

3. Traducción optimizada para cláusulas con factor de atenuación **t** y cláusulas sin restricciones umbrales en sus cuerpos, para las que es posible reducir el número de restricciones finales a establecer.

4. Posibilidad de indicar al sistema que el programa sea tratado como programa SLP en el sentido de [64]. En este caso el sistema resuelve ecuaciones basándose en las transformaciones de unificación módulo similaridad presentadas en [64], evitando ciertas alternativas de búsqueda que son necesarias para no perder soluciones en el caso de programas y objetivos SQCLP generales. Una discusión más detallada de esta cuestión se puede encontrar al final de [15](B.5, §2).

Las mejoras en la eficiencia según el tipo de programa pueden verse en el cuadro 1. En él, y a modo de comparación con `Prolog`, se proponen una serie de programas extraídos del *SICStus Prolog Benchmark*[3] que han podido ser adecuadamente adaptados para ejecutarse tanto en `Prolog` como en `(S)QCLP` (dado que no es posible ejecutar cualquier programa `Prolog` en `(S)QCLP`) y se muestra el factor de sobrecarga introducido por la transformación

[3]Véase `http://www.sics.se/isl/sicstuswww/site/perfomance.html`.

Programa	Q(b)[a]	Q(u)[b]	PQ(b)[c]	PQ(u)[d]	SQ(b)[e]	SQ(u)[f]
naivrev	1.80	10.71	4289.79	4415.11	56.22	65.75
deriv	1.94	10.60	331.45	469.67	29.63	39.32
qsort	1.05	1.11	135.59	136.98	2.51	2.83
query	1.02	1.12	7.17	7.13	3.80	3.88

[a]Versión QCLP($\mathcal{B}, \mathcal{R}$) (i.e. el programa no tiene la directiva `#prox`).
[b]Versión QCLP($\mathcal{U}, \mathcal{R}$) (i.e.el programa no tiene la directiva `#prox`).
[c]Versión SQCLP($\mathcal{S}_{\mathsf{id}}, \mathcal{B}, \mathcal{R}$).
[d]Versión SQCLP($\mathcal{S}_{\mathsf{id}}, \mathcal{U}, \mathcal{R}$).
[e]Versión SQCLP($\mathcal{S}_{\mathsf{id}}, \mathcal{B}, \mathcal{R}$) con directiva `#optimized_unif`.
[f]Versión SQCLP($\mathcal{S}_{\mathsf{id}}, \mathcal{U}, \mathcal{R}$) con directiva `#optimized_unif`.

Cuadro 1: Sobrecarga con respecto a `Prolog`

en la ejecución en `(S)QCLP`. Como puede verse, en algunos casos la sobrecarga es considerable, mientras que en otros es mucho más aceptable. Para más detalles sobre la eficiencia del sistema, véase [15](B.5, §5.3).

Por último, se optó por hacer una implementación sobre los sistemas CLP `SICStus Prolog` y *SWI-Prolog*, de manera que pudiera optarse, a discreción del usuario, por uno u otro sistema sin consecuencia alguna en el funcionamiento y utilización del sistema.

El sistema `(S)QCLP` está públicamente disponible en

`http://gpd.sip.ucm.es/cromdia/qclp`

y, tanto en la misma página como en [15](B.5, §5.2), se da la información necesaria para su instalación, ejecución y utilización.

Sobre los detalles de la técnica de implementación utilizada, se definen en [15](B.5) dos transformaciones de programas, denominadas $\mathsf{elim}_{\mathcal{S}}$ y $\mathsf{elim}_{\mathcal{D}}$ que eliminan, respectivamente, la relación de proximidad y el dominio de cualificación, y que en conjunto permiten transformar un SQCLP-programa en un CLP-programa equivalente. Estas dos transformaciones se formalizan, respectivamente, en [15](B.5, §4.1) y en [15](B.5, §4.2), donde se prueban, además, resultados de corrección y completitud con respecto a la semántica declarativa de los esquemas SQCLP, QCLP y CLP considerados por las transformaciones.

El sistema `(S)QCLP` implementa, de hecho, ambas transformaciones por separado y tras efectuarlas a partir de un programa de una instancia del esquema SQCLP aceptada, obtiene un programa CLP($\mathcal{R}$) equivalente que puede cargarse en cualquiera de los sistemas `Prolog` soportados y sobre el que pueden resolverse objetivos propuestos por el usuario.

A modo de ejemplo, se muestran a continuación dos ejecuciones para los programas de los ejemplos[4] 2.1 y 5.1 en el sistema `(S)QCLP` que se esperan resulten ilustrativas sobre el funcionamiento general del sistema. Para más detalles sobre el mismo, véase [15](B.5, §5) o `http://gpd.sip.ucm.es/cromdia/qclp`.

[4]En las ejecuciones siguientes se ha mantenido la versión original en inglés de los ejemplos que es como se distribuyen con el sistema `(S)QCLP`.

6.1.1. Ejecución 1: Números de Peano

Considérese el programa `Peano.qclp` como sigue:

Archivo: `qclp/samples/Peano.qclp`

```
1  # qdom w
2  % num( ?Num )
3  num(z) <--
4  num(s(X)) <-1- num(X)
5  % add( ?A, ?B, ?Sum )
6  add(c, X, X) <--
7  add(s(X), Y, s(Z)) <-1- add(X, Y, Z)
```

En donde la directiva `#qdom` de la línea 1 selecciona el dominio de cualificación $\mathcal{W}$, que se representa como `w` en la práctica. Otros dominios posibles serían `b` (para $\mathcal{B}$), `u` (para $\mathcal{U}$) y cualquier producto (d_1,d_2) construido a partir de los dominios de cualificación `b`, `u` y `w`. Desde la carpeta `qclp/` podemos, entonces, hacer lo siguiente[5]:

```
$ sicstus
SICStus 4.0.4 (x86_64-darwin-8.11.1): Mon Jun 16 23:58:36 CEST 2008
Licensed to SP4fdi.ucm
| ?- [qclp].

WELCOME TO (S)QCLP 0.6.2
(S)QCLP is free software and comes with absolutely no warranty.
Support & Updates: http://gpd.sip.ucm.es/cromdia/qclp.

Type ':help.' for help.
yes
| ?- :cd(samples).
qclp/samples/
yes
| ?- :run('Peano').
<Peano> Compiling...
<Peano> QDom: 'w'.
<Peano> Translating to QCLP...
<Peano> Translating to CLP...
<Peano> Generating code...
<Peano> Done.
<Peano> Loaded.
yes
| ?- num(X)#W.
W = 0.0, X = z ? ;
W = 1.0, X = s(z) ?
yes
| ?- num(X)#W::W>=0.
```

[5]Los comentarios de `SICStus Prolog` se han omitido y los vínculos de cada solución se muestran en una única línea.

```
W = 0.0, X = z ? ;
no
| ?- add(s(s(X)),s(X),Z)#W::W>=1.
no
| ?- add(s(s(X)),s(X),Z)#W::W>=2.
W = 2.0, X = c, Z = s(s(s(c))) ? ;
no
| ?-
```

6.1.2. Ejecución 2: Obras

Considérese la relación de proximidad `Work.prox` como sigue:

File: `qclp/samples/Work.prox`

```
1  % PREDICATES PROXIMITY...
2  % pprox( Symbol1, Symbol2, Arity, Value )
3  % CONSTRUCTORS PROXIMITY...
4  % cprox( Symbol1, Symbol2, Arity, Value )
5  cprox(king_lear, king_liar, 0, (0.8,2.0)).
```

Nótese que en la línea 5 se muestra la codificación de la ecuación

$$\mathcal{S}(\texttt{king\_lear}, \texttt{king\_liar}) = (0.8, 2.0)$$

que define propiamente la relación de proximidad. Y considérese también el programa `Work.qclp` como sigue:

Archivo: `qclp/samples/Work.qclp`

```
1  # qdom (u,w)
2  # prox 'Work'
3  % famous( ?Author )
4  famous(shakespeare) <-(0.9,1)-
5  % wrote( ?Author, ?Book )
6  wrote(shakespeare, king_lear) <-(1,1)-
8  % good_work( ?Work )
9  good_work(X) <-(0.75,3)- famous(Y)#(0.5,100), wrote(Y,X)
```

En donde las directivas `#qdom` y `#prox` permiten especificar, respectivamente, el dominio de cualificación del programa, en este caso el producto cartesiano estricto $\mathcal{U}\otimes\mathcal{W}$, y la relación de proximidad a utilizar. Entonces, desde la carpeta `qclp/` podemos hacer:

```
$ sicstus
SICStus 4.0.4 (x86_64-darwin-8.11.1): Mon Jun 16 23:58:36 CEST 2008
Licensed to SP4fdi.ucm
| ?- [qclp].
```

```
WELCOME TO (S)QCLP 0.6.2
(S)QCLP is free software and comes with absolutely no warranty.
Support & Updates: http://gpd.sip.ucm.es/cromdia/qclp.

Type ':help.' for help.
yes
| ?- :cd(samples).
qclp/samples/
yes
| ?- :run('Work').
<Work> Compiling...
<Work> QDom: 'u,w'.
<Work> Prox: 'Work'.
<Work> Translating to QCLP...
<Work> Translating to CLP...
<Work> Generating code...
<Work> Done.
<Work> Loaded.
yes
| ?- king_lear==X#W.
W = (1.0,0.0), X = king_lear ? ;
W = (0.8,2.0), X = king_liar ? ;
no
| ?- good_work(king_lear)#W::W>=(0.65,5).
W = (0.675,4.0) ? ;
no
| ?- good_work(king_liar)#W::W>=(0.65,5).
W = (0.675,4.0) ? ;
no
| ?-
```

6.2. Otros prototipos

Antes del desarrollo final de (S)QCLP, se procedió a construir otro prototipo preliminar destinado a implementar las instancias QLP($\mathcal{U}$), QLP($\mathcal{W}$) y QLP($\mathcal{U}\times\mathcal{W}$) sobre el sistema CFLP Toy [6] como una opción activable por el usuario que hacía un preprocesado de los programas QLP para que pudieran ser ejecutados desde el sistema Toy.

El proceso de transformación de programas implementado seguía las ideas de la resolución QSLD($\mathcal{D}$) de la sección 2.4, que a pesar de haber sido pensadas para implementar una variante de la resolución SLD estándar capaz de resolver objetivos para programas lógicos con cualificación, requeriría de mayores esfuerzos de implementación que la implementación de una transformación de programas, como ya se ha razonado con anterioridad.

En este momento se optó por la utilización del sistema Toy como parte de la implementación y no por un sistema Prolog, pensando en sentar las bases para integrar en Toy la implementación de futuras extensiones de QLP que soportasen características de programación lógico-funcional perezosas, tal como el esquema QCFLP desarrollado posteriormente.

El prototipo permitía transformar programas QLP básicos, es decir que no incluyeran

ninguna de las extensiones propuestas en el marco BQLP como era las marcas `tt` y `ff`, y las restricciones umbrales en los cuerpos de las cláusulas cualificadas. Asimismo, el cálculo de los valores de cualificación finales se hacía mediante dos tipos de restricciones de cualificación, implementadas como restricciones de $\mathcal{R}$, que venían a ser las precursoras de las dos restricciones **qVal** y **qBound** que se definirían más tarde para los esquemas con restricciones, i.e. QCFLP y SQCLP. La principal diferencia entre las restricciones empleadas por el prototipo y las definidas **qVal** y **qBound**, era que la restricción **qBound** se definió mediante una igualdad, i.e. se hacía algo como $W = \alpha \circ \bigsqcap\{W_1, \ldots, W_m\}$, en lugar de la posteriormente utilizada $W \trianglelefteq \alpha \circ \bigsqcap\{W_1, \ldots, W_m\}$, lo que tenía la ventaja de no dejar sin instanciar ninguna variable de cualificación del objetivo a costa de convertir en impaciente, y por tanto exigir la evaluación de las variables W_i del programa.

La versión actual de este prototipo no está públicamente accesible, aunque sigue formando parte de los planes de desarrollo del sistema `Toy` y aparecerá como parte integrante de una futura versión pública de dicho sistema.

Para el esquema SQLP no llegó a desarrollarse un prototipo funcional completo antes del propio `(S)QCLP`, y los ejemplos se ejecutaron en el prototipo desarrollado para QLP tras una transformación manual de los mismos siguiendo la técnica aportada en [11](A.3).

7. Conclusiones y trabajo futuro

A lo largo del trabajo de esta tesis se han aportado una serie de esquemas de programación declarativa para programas lógicos y lógico-funcionales desarrollando, para cada uno de ellos, una semántica declarativa clara y rigurosa, y una técnica de resolución de objetivos basada ya en una semántica operacional, ya en una técnica de transformación de programas que se prueba correcta y completa con respecto a la semántica declarativa de ambos esquemas de programación involucrados en la transformación. Así, podemos efectivamente decir que los objetivos de la tesis han quedado cubiertos de manera satisfactoria, tanto por los esquemas de programación desarrollados como por el sistema `(S)QCLP`, implementado siguiendo las ideas de la técnica de transformación de programas propuesta para el esquema SQCLP.

Más concretamente, se han desarrollado esquemas de programación declarativa que pueden clasificarse de dos maneras diferentes: a) atendiendo al paradigma de programación que toman como base y extienden; y b) atendiendo al elemento o elementos que incorporan. Por lo tanto, tenemos:

- *esquemas que extienden la programación lógica*, como el esquema QLP (sección 2) con instancias QLP($\mathcal{D}$), el esquema BQLP (sección 2) con instancias BQLP($\mathcal{D}$), y el esquema SQLP (sección 3) con instancias SQLP($\mathcal{S}, \mathcal{D}$);
- *esquemas que extienden la programación lógica con restricciones*, como el esquema SQCLP (sección 5) con instancias SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$); y
- *esquemas que extienden la programación lógico-funcional*, como el esquema QCFLP (sección 4) con instancias QCFLP($\mathcal{D}, \mathcal{C}$).

Y también:

- *esquemas que incorporan cualificación*, como los esquemas QLP (sección 2), BQLP (sección 2), SQLP (sección 3), QCFLP (sección 4) y SQCLP (sección 5);

- *esquemas que incorporan similaridad* o *proximidad*, como el esquema SQLP (sección 3) para el primer caso, y el esquema SQCLP (sección 5) para el segundo;
- *esquemas que incorporan restricciones*, como los esquemas QCFLP (sección 4) y SQCLP (sección 5);
- *esquemas que incorporan funciones*, como el esquema QCFLP (sección 4).

Por otro lado, y dado que es posible ver los esquemas QLP, BQLP y SQLP como instanciaciones parciales del esquema SQCLP tal y como se cuenta en las conclusiones de [61](A.5, §4), puede decirse que el trabajo desarrollado ha seguido dos direcciones principales: a) extender la programación lógica primero con cualificación, después con similaridad (o proximidad) y por último con restricciones al estilo de la semántica $\mathcal{S}_2$ para CLP de [23]; y b) extender la programación lógico-funcional de primer orden, entendida como un subconjunto de la lógica de reescritura propuesta en [47], con cualificación. Por lo tanto, se han desarrollado efectivamente dos esquemas de programación: uno para programación lógica (o programación lógica con restricciones), i.e. SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$); y otro para programación lógico-funcional de primer orden, i.e. QCFLP($\mathcal{D}, \mathcal{C}$). Y también un sistema `(S)QCLP` (Sección 6) que implementa varias instancias interesantes del esquema SQCLP sobre los sistemas `SICStus Prolog` y `SWI-Prolog`, y que permite escribir, ejecutar y resolver objetivos para programas escritos en los esquemas QLP, SQLP y SQCLP, así como en cualquier otro esquema resulte de una instanciación parcial de cualquiera de las instancias de SQCLP efectivamente implementadas en `(S)QCLP`, sobre cualquiera de los dos sistemas `Prolog` anteriormente mencionados.

En comparación con otros enfoques relacionados, y a modo de resumen de las conclusiones de las publicaciones asociadas a esta tesis, la programación cualificada presenta interesantes contribuciones. Con respecto a los programas cuantitativos (QLP) de van Emden [71], la programación lógica basada en similaridad (SLP) [64] y el lenguaje `Bousi~Prolog` [39], los esquemas QLP, SQLP y SQCLP engloban, respectivamente, a cada uno de ellos y, por consiguiente, el esquema SQCLP a los tres. Con respecto a los programas anotados generalizados (GAP) de Kifer y Subrahmanian [41], a pesar de que la sintaxis de los programas QLP($\mathcal{D}$), tal vez también los BQLP($\mathcal{D}$), y el procedimiento de resolución QSLD($\mathcal{D}$), pueden encajar perfectamente dentro del esquema GAP, la visión de la semántica aquí presentada permite probar resultados más fuertes que aquellos para GAP, y los esquemas sucesivos basados en QLP introducen elementos imposibles de representar, de manera natural, en el esquema GAP como las relaciones de similaridad o proximidad y la programación con restricciones. Además, el método de resolución QSLD($\mathcal{D}$) puede implementarse de manera más eficiente que la resolución SLD restringida de GAP al evitar el uso de los costosos *reductantes*.

Los predicados bivaluados introducidos en el esquema BQLP aportan una sencilla forma de permitir devolver un valor diferente de *cierto* a la hora de interpretar un predicado en un programa lógico. De la misma manera, permiten introducir una muy sencilla negación en los programas al permitir interpretar predicados que demuestran veracidad y predicados que demuestran falsedad. La extensión de los sucesivos SQLP y SQCLP con predicados bivaluados resulta sencilla, a pesar de haberse omitido para permitir centrarse en las novedades que cada esquema presenta y sus respectivos problemas técnicos. En el esquema QCFLP, la valoración de un predicado vendría a ser la expresión que constituye el lado derecho de

una regla de programa y, por este motivo, no tiene sentido pensar en posibles valoraciones para las funciones definidas en el paradigma de la programación lógico-funcional.

Las relaciones de similaridad empleadas en SQLP, y posteriormente las relaciones de proximidad empleadas en SQCLP, son bastante generales al permitir tomar valores en un determinado dominio de cualificación y no necesariamente en el intervalo real $[0, 1]$ como en [64]. En relación al enfoque multi-adjunto de MALP [48], los esquemas SQLP y SQCLP tienen una finalidad y motivación diferente, como prueban las diferencias entre las álgebras multi-adjuntas y los dominios de cualificación como estructuras.

En lo que a programación lógico-funcional se refiere, el esquema QCFLP, a pesar de basarse únicamente en un fragmento de primer orden de la lógica de reescritura CRWL, resulta más expresivo que el resto de enfoques relacionados de los que se tiene constancia. Además, la técnica de transformación de programas QCFLP($\mathcal{D}, \mathcal{C}$) a programas CFLP($\mathcal{C}$) equivalentes permitiría efectivamente implementar un sistema para resolver QCFLP-objetivos para QCFLP-programas sobre un sistema CFLP actual como `Toy` [6] o `Curry` [29].

Por último, junto al esquema SQCLP se proporciona una noción de Π-proximidad $\approx_{\lambda,\Pi}$, presentada por primera vez en [61](A.5, Def. 2.3), que permite especificar el papel semántico de una relación de proximidad en un enfoque basado en restricciones con menos dificultades técnicas que en enfoques relacionados previos. Asimismo, la semántica declarativa inspirada en la CLP semántica de observables de [24, 23] proporciona una caracterización del modelo mínimo de un programa y una noción declarativa de objetivo y solución adecuadas para su implementación en un sistema como `(S)QCLP`, que sobre un sistema CLP($\mathcal{R}$), es capaz de resolver objetivos para programas de algunas instancias interesante del esquema SQCLP.

Las líneas de investigación que quedan abiertas para el trabajo futuro se centran por un lado en la extensión del esquema QCFLP a orden superior, incorporando tal vez relaciones de proximidad como en SQCLP y permitiendo por lo tanto servir como extensión con incertidumbre de la base computacional de un sistema CFLP como `Toy` [6], y el desarrollo de una resolución SLD basada en proximidad para SQCLP al estilo de la proporcionada en [64] para programas SLP. Por otro lado, queda pendiente la tarea de encontrar aplicaciones reales interesantes de estos esquemas de programación declarativa, en especial, en lo que a representación de consultas sobre dominios específicos se refiere (como por ejemplo dentro del campo de la medicina) o en aplicaciones de búsqueda flexible de información en la web.

Otras líneas de menor relevancia, o a más largo plazo, en las que cabría trabajar son:

- buscar otros posibles dominios de cualificación que aporten expresividad a la programación declarativa con cualificación;
- estudiar otras posibles estructuras para representar la incertidumbre y ver cómo podrían encajar con los dominios de cualificación, como pueden ser los birretículos o los semianillos; y
- una vez desarrollada la extensión de QCFLP a orden superior, implementar una extensión de `Toy` con cualificación.

Referencias

[1] Sergio Antoy and Michael Hanus. Functional logic programming. *Communications of the ACM*, 53(4):74–85, April 2010.

[2] 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.

[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] Francesca Arcelli Fontana. Likelog for flexible query answering. *Soft Computing*, 7:107–114, 2002.

[6] 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 2.3.1). In R. Caballero and J. Sánchez, editors, *User Manual*, available at `http://toy.sourceforge.net`, July 2007.

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

[8] H. P. Barendregt. Functional programming and lambda calculus. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 7, pages 321–363. Elsevier and The MIT Press, 1990.

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

[10] Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. Semiring-based constraint logic programming: Syntax and semantics. *ACM Transactions on Programming Languages and Systems*, 3(1):1–29, January 2001.

[11] Rafael Caballero, Mario Rodríguez-Artalejo, and Carlos A. Romero-Díaz. Similarity-based reasoning in qualified logic programming. In *PPDP '08: Proceedings of the 10th international ACM SIGPLAN conference on Principles and Practice of Declarative Programming*, pages 185–194, Valencia, Spain, July 15–17 2008. ACM.

[12] Rafael Caballero, Mario Rodríguez-Artalejo, and Carlos A. Romero-Díaz. A generic scheme for qualified constraint functional logic programming. Technical Report SIC-1-09 (CoRR abs/1101.2146), Universidad Complutense, Departamento de Sistemas Informáticos y Computación, Madrid, Spain, 2009.

[13] Rafael Caballero, Mario Rodríguez-Artalejo, and Carlos A. Romero-Díaz. Qualified computations in functional logic programming. In P.M. Hill and D.S. Warren, editors, *Logic Programming (ICLP'09)*, volume 5649 of *LNCS*, pages 449–463, Pasadena, CA, USA, July 14-17 2009. Springer-Verlag Berlin Heidelberg.

[14] Rafael Caballero, Mario Rodríguez-Artalejo, and Carlos A. Romero-Díaz. Similarity-based reasoning in qualified logic programming. CoRR abs/1008.3867, 2010.

[15] Rafael Caballero, Mario Rodríguez-Artalejo, and Carlos A. Romero-Díaz. A transformation-based implementation for CLP with qualification and proximity. Technical Report SIC-4-10 (CoRR abs/1009.1976), Universidad Complutense, Departamento de Sistemas Informáticos y Computación, Madrid, Spain, 2010.

[16] 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.

[17] D. Dubois and H. Prade. *Fuzzy Sets and Systems: Theory and Applications*. Academic Press, New York, NY, USA, 1980.

[18] D. Dubois and H. Prade. *Possibility Theory*. New York: Plenum, 1988.

[19] D. Dubois and H. Prade. Possibilistic logic: a retrospective and prospective view. *Fuzzy Sets and Systems*, 144:3–23, 2004.

[20] Melvin Fitting. Bilattices and the semantics of logic programming. *Journal of Logic Programming*, 11:91–116, 1991.

[21] FLOPER: A Fuzzy LOgic Programming Environment for Research. Available at `http://www.dsi.uclm.es/investigacion/dect/FLOPERpage.htm`, 2010.

[22] F. Formato, G. Gerla, and M. I. Sessa. Similarity-based unification. *Fundamenta Informaticae*, 40:1–22, 2000.

[23] Maurizio Gabbrielli, Giovanna M. Dore, and Giorgio Levi. Observable semantics for constraint logic programs. *Journal of Logic and Computation*, 5(2):133–171, 1995.

[24] Maurizio Gabbrielli and Giorgio Levi. Modeling answer constraints in constraint logic programs. In *Proceedings of the 8th International Conference on Logic Programming (ICLP'91)*, pages 238–252. The MIT Press, 1991.

[25] Yan Georget and Philippe Codognet. Compiling semiring-based constraints with CLP(FD,S). In *Proceedings of the 4th International Conference on Principles and Practice of Constraint Programming*, volume 1520 of *LNCS*, pages 205–219. Springer-Verlag, 1998.

[26] G. Gerla and M.I. Sessa. Similarity in logic programming. In G. Chen, M. Ying, and K.Y. Cai, editors, *Fuzzy Logic and Soft Computing*, pages 19–31. Kluwer Accademic Publishers, 1999.

[27] 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.

[28] Petr Hájek. *Metamathematics of Fuzzy Logic*. Dordrecht: Kluwer, 1998.

[29] M. Hanus, Ed. Curry: An integrated functional logic language (vers. 0.8.2, 2006); `http://www.curry-language.org`.

[30] Haskell 98. Available at `http://www.haskell.org/`, 2010.

[31] M. Höhfeld and G. Smolka. Definite relations over constraint languages. Technical Report LILOG Report 53, IBM Deutschland, 1988.

[32] M. Ishizuka and N. Kanai. Prolog-ELF incorporating fuzzy logic. In Aravind K. Joshi, editor, *Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI'85)*, pages 701–703, Los Angeles, CA, USA, August 1985. Morgan Kaufmann.

[33] J. Jaffar and J. L. Lassez. Constraint logic programming. In *Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of Programming Languages (POPL'87)*, pages 111–119, Munich, West Germany, 1987. ACM New York, NY, USA.

[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] Simon Peyton Jones. The haskell 98 language. *Journal of Functional Programming*, 13(1):7–255, January 2003.

[37] R. Julián, G. Moreno, and J. Penabad. An improved reductant calculus using fuzzy partial evaluation techniques. *Fuzzy Sets and Systems*, 160(2):162–181, 2009.

[38] P. Julián-Iranzo, C. Rubio, and J. Gallardo. Bousi∼Prolog: a prolog extension language for flexible query answering. In J. M. Almendros-Jiménez, editor, *Proceedings of the Eighth Spanish Conference on Programming and Computer Languages (PROLE 2008)*, volume 248 of *ENTCS*, pages 131–147, Gijón, Spain, October 7–10 2009. Elsevier.

[39] Pascual Julián-Iranzo and Clemente Rubio-Manzano. A declarative semantics for Bousi∼Prolog. In *PPDP'09: Proceedings of the 11th ACM SIGPLAN conference on Principles and practice of declarative programming*, pages 149–160, Coimbra, Portugal, September 7–9 2009. ACM.

[40] Pascual Julián-Iranzo and Clemente Rubio-Manzano. A similarity-based WAM for Bousi∼Prolog. In *Bio-Inspired Systems: Computational and Ambient Intelligence (IWANN 2009)*, volume 5517 of *LNCS*, pages 245–252, Salamanca, Spain, 10-12 June 2009. Springer Berlin / Heidelberg.

[41] 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.

[42] R. C. T. Lee. Fuzzy logic and the resolution principle. *Journal of the Association for Computing Machinery (ACM)*, 19(1):109–119, January 1972.

[43] Dayi Li and Dongbo Liu. *A Fuzzy Prolog Database System*. John Wiley & Sons, 1990.

[44] John W. Lloyd. *Foundations of Logic Programming, Second Edition*. Springer, 1987.

[45] 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.

[46] F. J. López-Fraguas, M. Rodríguez-Artalejo, and R. del Vado-Virseda. A lazy narrowing calculus for declarative constraint programming. In *Proceedings of the 6th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP'04)*, pages 43–54. ACM Press, 2004.

[47] 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.

[48] J. Medina, M. Ojeda-Aciego, and P. Vojtáš. Multi-adjoint logic programming with continuous semantics. In T. Eiter, W. Faber, and M. Truszczyinski, editors, *Logic Programming and Non-Monotonic Reasoning (LPNMR'01)*, volume 2173 of *LNAI*, pages 351–364. Springer-Verlag, 2001.

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

[50] Ginés Moreno and Vicente Pascual. Formal properties of needed narrowing with similarity relations. *Electronic Notes in Theoretical Computer Science*, 188:21–35, 2007.

[51] Raymond T. Ng and V. S. Subrahmanian. Probabilistic logic programming. *Information and Computation*, 101(2):150–201, 1992.

[52] J. Pavelka. On fuzzy logic I, II, III. *Zeitschrift fur Math. Logik und Grundlagen der Math*, 25:45–52, 119–134, 447–464, 1979.

[53] Jaime Penabad Vázquez. *Desplegado de Programas Lógicos Difusos*. PhD thesis, Departamento de Sistemas Informáticos, Universidad de Castilla-La Mancha, 2010.

[54] N. Rescher. *Plausible Reasoning*. Van Gorcum, Amsterdam, 1976.

[55] 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.

[56] S. Riezler. *Probabilistic Constraint Logic Programming*. PhD thesis, Neuphilologischen Fakultät del Universität Tübingen, 1998.

[57] 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.

[58] Mario Rodríguez-Artalejo and Carlos A. Romero-Díaz. A generic scheme for qualified logic programming. Technical Report SIC-1-08 (CoRR abs/1008.3863), Universidad Complutense, Departamento de Sistemas Informáticos y Computación, Madrid, Spain, 2008.

[59] Mario Rodríguez-Artalejo and Carlos A. Romero-Díaz. Quantitative logic programming revisited. In J. Garrigue and M. Hermenegildo, editors, *Functional and Logic Programming (FLOPS'08)*, volume 4989 of *LNCS*, pages 272–288, Ise, Japan, April 14–16 2008. Springer-Verlag.

[60] Mario Rodríguez-Artalejo and Carlos A. Romero-Díaz. Qualified logic programming with bivalued predicates. In J. M. Almendros-Jiménez, editor, *Proceedings of the Eighth Spanish Conference on Programming and Computer Languages (PROLE 2008)*, volume 248 of *ENTCS*, pages 67–82, Gijón, Spain, October 7–10 2009. Elsevier.

[61] Mario Rodríguez-Artalejo and Carlos A. Romero-Díaz. A declarative semantics for CLP with qualification and proximity. *Theory and Practice of Logic Programming, 26th Int'l. Conference on Logic Programming (ICLP'10) Special Issue*, 10(4–6):627–642, 2010.

[62] Mario Rodríguez-Artalejo and Carlos A. Romero-Díaz. Fixpoint & Proof-theoretic Semantics for CLP with Qualification and Proximity. Technical Report SIC-1-10 (CoRR abs/1009.1977), Universidad Complutense, Departamento de Sistemas Informáticos y Computación, Madrid, Spain, 2010.

[63] M. I. Sessa. Translations and similarity-based logic programming. *Soft Computing*, 5(2), 2001.

[64] M. I. Sessa. Approximate reasoning by similarity-based SLD resolution. *Theoretical Computer Science*, 275(1-2):389–426, 2002.

[65] 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.

[66] S. Shenoi and A. Melton. Proximity relations in the fuzzy relational database model. *Fuzzy Sets and Systems*, 100(supl.):51–62, 1999.

[67] SICS AB. SICStus Prolog. Available at `http://www.sics.se/sicstus`, 2010.

[68] 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.

[69] 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.

[70] SWI-Prolog. Available at `http://www.swi-prolog.org`, 2010.

[71] M. H. van Emden. Quantitative deduction and its fixpoint theory. *Journal of Logic Programming*, 3(1):37–53, 1986.

[72] 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.

[73] C. Vaucheret, S. Guadarrama, and S. Muñoz. Fuzzy prolog: A simple general implementation using CLP($\mathcal{R}$). In M. Baaz and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'02)*, volume 2514 of *LNCS*, pages 450–463, Tbilisi, Georgia, October 14–18 2002. Springer Berlin / Heidelberg.

[74] A. Visser. Four valued semantics and the liar. *Journal of Philosophical Logic*, 13:181–212, 1984.

[75] P. Vojtáš. Fuzzy logic programming. *Fuzzy Sets and Systems*, 124:361–370, 2001.

[76] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965.

[77] L. A. Zadeh. Similarity relations and fuzzy orderings. *Information Sciences*, 3(2):177–200, 1971.

[78] L. A. Zadeh. Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, 1:3–28, 1978.

A. Publicaciones

(A.1) **Quantitative logic programming revisited**
M. Rodríguez-Artalejo and C. A. Romero-Díaz

In J. Garrigue and M. Hermenegildo, editors, Functional and Logic Programming (FLOPS 2008), volume 4989 of LNCS, pages 272–288, Ise, Japan, April 14–16 2008. Springer-Verlag.

→ **Página** 86

(A.2) **Qualified logic programming with bivalued predicates**
M. Rodríguez-Artalejo and C. A. Romero-Díaz

In J. M. Almendros-Jiménez, editor, Proceedings of the Eighth Spanish Conference on Programming and Computer Languages (PROLE 2008), volume 248 of ENTCS, pages 67–82, Gijón, Spain, October 7–10 2009. Elsevier.

→ **Página** 103

(A.3) **Similarity-based reasoning in qualified logic programming**
R. Caballero, M. Rodríguez-Artalejo, and C. A. Romero-Díaz

In PPDP'08: Proceedings of the 10th international ACM SIGPLAN conference on Principles and Practice of Declarative Programming, pages 185–194, Valencia, Spain, July 15–17.

→ **Página** 119

(A.4) **Qualified computations in functional logic programming**
R. Caballero, M. Rodríguez-Artalejo, and C. A. Romero-Díaz

In P.M. Hill and D.S. Warren, editors, Logic Programming (ICLP 2009), volume 5649 of LNCS, pages 449–463, Pasadena, CA, USA, July 14–17 2009. Springer-Verlag Berlin Heidelberg.

→ **Página** 129

(A.5) **A declarative semantics for CLP with qualification and proximity**
M. Rodríguez-Artalejo and C. A. Romero-Díaz

Theory and Practice of Logic Programming, 26th Int'l. Conference on Logic Programming (ICLP 2010) Special Issue, 10(4–6):627–642, 2010.

→ **Página** 144

B. Recursos adicionales

(B.1) **A generic scheme for qualified logic programming**
M. Rodríguez-Artalejo and C. A. Romero-Díaz

Technical Report SIC-1-08 (CoRR abs/1008.3863), Universidad Complutense, Departamento de Sistemas Informáticos y Computación, Madrid, Spain, 2008

→ **Página** 161

(B.2) **Similarity-based reasoning in qualified logic programming (revised edition)**
R. Caballero, M. Rodríguez-Artalejo and C. A. Romero-Díaz

CoRR abs/1008.3867, 2010

→ **Página** 184

(B.3) **A generic scheme for qualified constraint functional logic programming**
R. Caballero, M. Rodríguez-Artalejo and C. A. Romero-Díaz

Technical Report SIC-1-09 (CoRR abs/1101.2146), Universidad Complutense, Departamento de Sistemas Informáticos y Computación, Madrid, Spain, 2009

→ **Página** 194

(B.4) **Fixpoint & proof-theoretic semantics for CLP with qualification and proximity**
M. Rodríguez-Artalejo and C. A. Romero-Díaz

Technical Report SIC-1-10 (CoRR abs/1009.1977), Universidad Complutense, Departamento de Sistemas Informáticos y Computación, Madrid, Spain, 2010

→ **Página** 230

(B.5) **A transformation-based implementation for CLP with qualification and proximity (preliminary version)**
R. Caballero, M. Rodríguez-Artalejo and C. A. Romero-Díaz

Technical Report SIC-4-10 (CoRR abs/1009.1976), Universidad Complutense, Departamento de Sistemas Informáticos y Computación, Madrid, Spain, 2010

→ **Página** 277

A Generic Scheme for Qualified Logic Programming*

Technical Report SIC-1-08

Mario Rodríguez-Artalejo and Carlos A. Romero-Díaz

Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid, Spain
mario@sip.ucm.es and cromdia@fdi.ucm.es

Abstract. Uncertainty in Logic Programming has been investigated since about 25 years, publishing papers dealing with various approaches to semantics and different applications. This report is intended as a first step towards the investigation of *qualified computations* in Constraint Functional Logic Programming, including *uncertain computations* as a particular case. We revise an early proposal, namely van Emden's *Quantitative Logic Programming* [24], and we improve it in two ways. Firstly, we generalize van Emden's QLP to a generic scheme $QLP(\mathcal{D})$ parameterized by any given *Qualification Domain* $\mathcal{D}$, which must be a lattice satisfying certain natural axioms. We present several interesting instances for $\mathcal{D}$, one of which corresponds to van Emden's QLP. Secondly, we generalize van Emden's results by providing stronger ones, concerning both semantics and goal solving. We present *Qualified SLD Resolution* over $\mathcal{D}$, a sound and strongly complete goal solving procedure for $QLP(\mathcal{D})$, which is applicable to open goals and can be efficiently implemented using CLP technology over any constraint domain $\mathcal{C}_\mathcal{D}$ able to deal with qualification constraints over $\mathcal{D}$. We have developed a prototype implementation of some instances of the $QLP(\mathcal{D})$ scheme (including van Emden's QLP) on top of the $CFLP$ system $\mathcal{TOY}$.
Keywords: Quantitative Logic Programming, Qualification Domains, Qualification Constraints.

1 Introduction

The investigation of uncertainty in logic programming has proceeded along various lines during the last 25 years. A recent recollection by V. S. Subrahmanian [23] highlights some phases in the evolution of the topic from the viewpoint of a committed researcher.

Research on the field has dealt with various approaches to semantics, as well as different applications. One of the earliest approaches was *Quantitative Logic Programming*, QLP for short. This can be traced back to a paper by Shapiro

* Research partially supported by projects MERIT-FORMS (TIN2005-09027-C03-03) and PROMESAS-CAM(S-0505/TIC/0407)

[19], who proposed to use real numbers in the interval $(0, 1]$ as *certainty factors*, as well as *certainty functions* for propagating certainty factors from the bodies to the heads of program clauses. Subsequently, van Emden [24] considered QLP with an *attenuation factor* $f \in (0, 1]$ attached to the implication of each program clause and restricted his attention to the certainty function which propagates to a clause head the certainty factor $f \times b$, where f is the clause's attenuation factor and b is the minimum of the certainty factors known for the body atoms. Van Emden's approach was less general than Shapiro's because of the fixed choice of a particular certainty function, but it allowed to prove more general results on model theoretic and fixpoint semantics, similar to those previously obtained in [25,1] for classical Logic Programming. Moreover, [24] gave a procedure for computing the certainty of atoms in the least Herbrand model of a given program, by applying an alpha-beta heuristic to the atoms' and/or search trees. This procedure worked only for ground atoms having a finite search tree.

Following these beginnings, logic programming with uncertainty developed in various directions. Subrahmanian [21] proposed an alternative to [24], using a different lattice of numeric values (aiming at a separate representation of certainty degrees for truth an falsity) as well as clauses whose atoms were annotated with values from this lattice. Neither certainty functions nor attenuation factors were used in this approach, which was extended in [22] to provide goal solving procedures enjoying stronger soundness and completeness results. As a brief summary of some significant later contributions let us mention: generalized *annotated logic programs* [11], a quite general framework which will be discussed in more detail in Section 6; semantics based on *bilattices* of generalized truth values with both a 'knowledge' order and a 'truth' order [8]; logic programming with probabilistic semantics and applications to deductive databases [14,15]; quantitative and probabilistic constraint logic programming and applications to natural language processing [16]; *hybrid probabilistic programs* [5]; probabilistic agent programs [7] and their extension to deal with both time and uncertainty [6]; logic programs with similarity based unification and applications to flexible data retrieval [2,18,9,12]; and functional logic programming with similarity based unification [13].

We are interested in a long-term research project aiming at a generalization of existing work on logic programming with uncertainty. The generalization we plan to develop will operate in two directions: a) extending logic programming languages to more expressive multi-paradigm declarative languages supporting *functions* and *constraints*; and b) generalizing uncertain truth values to so-called *qualification values*, attached to computed answers and intended to measure the degree in which such computed answers satisfy various user's expectations. In this setting, (constraint) logic programming with uncertainty becomes the particular case in which no functional programming features are used and qualification values are just uncertain truth values. As a first step, we present in this report a generalization of the early QLP proposal by van Emden [24], which is still appealing because of its neat semantics. Syntactically, our proposal is very close to van Emden's QLP: we use qualified definite Horn clauses $A \leftarrow d - \overline{B}$ with

an attenuation value d attached to the implication and no annotations attached to the atoms. However, we improve [24] in the two ways summarized in the abstract: firstly, we replace numeric certainty values (in particular, those playing the role of attenuation factors in program clauses) by qualification values belonging to a parametrically given *Qualification Domain* $\mathcal{D}$ with a lattice structure, which provides abstract operations generalizing the use of *min* (minimum) and $\times$ (product) in [24]. In this way we get a *generic scheme* $QLP(\mathcal{D})$. Secondly, we present stronger semantic results and a sound and strongly complete goal solving procedure called *Qualified SLD Resolution* over $\mathcal{D}$(in symbols, $SLD(\mathcal{D})$), which extends SLD resolution using *annotated atoms* and *qualification constraints* over $\mathcal{D}$. The $QLP(\mathcal{D})$ scheme enjoys nice semantic properties and has interesting instances that can be efficiently implemented using CLP technology: $QLP(\mathcal{D})$ programs and goals can be easily translated into $CLP(\mathcal{C}_\mathcal{D})$ for any choice of a constraint domain $\mathcal{C}_\mathcal{D}$ able to compute with qualification constraints over $\mathcal{D}$.

We have developed a prototype implementation of some instances of the $QLP(\mathcal{D})$ scheme (including van Emden's QLP) on top of the $CFLP$ system $\mathcal{TOY}$.

After this introduction, the rest of the report is structured as follows: Section 2 presents the axioms for qualification domains $\mathcal{D}$, showing some basic instances and proving that the class of such domains is closed under cartesian product. Section 3 presents the syntax and declarative semantics of the $QLP(\mathcal{D})$ scheme. Section 4 presents qualified SLD resolution over $\mathcal{D}$ with its soundness and strong completeness properties. Section 5 presents the general implementation technique for $QLP(\mathcal{D})$ that we have used to implement some instances of the scheme (including van Emden's QLP) on top of the $CFLP$ system $\mathcal{TOY}$. Finally, Section 6 presents our conclusions and plans for future work. Appendix A includes detailed proofs for the main results. Other proofs that have been ommitted or sketched can be found in [17] (in Spanish).

2 Qualification Domains

By definition, a *Qualification Domain* is any structure $\mathcal{D} = \langle D, \sqsubseteq, \bot, \top, \circ \rangle$ such that:

1. $\langle \mathcal{D}, \sqsubseteq, \bot, \top \rangle$ is a lattice with extreme points $\bot$ and $\top$ w.r.t. the partial ordering $\sqsubseteq$. For given elements $d, e \in D$, we write $d \sqcap e$ for the *greatest lower bound* (*glb*) of d and e and $d \sqcup e$ for the *least upper bound* (*lub*) of d and e. We also write $d \sqsubset e$ as abbreviation for $d \sqsubseteq e \wedge d \neq e$.
2. $\circ : D \times D \to D$, called *attenuation operation*, verifies the following axioms:
   (a) $\circ$ is associative, commutative and monotonic w.r.t. $\sqsubseteq$.
   (b) $\forall d \in D : d \circ \top = d$.
   (c) $\forall d \in D : d \circ \bot = \bot$.
   (d) $\forall d, e \in D \setminus \{\bot, \top\} : d \circ e \sqsubset e$.
   (e) $\forall d, e_1, e_2 \in D : d \circ (e_1 \sqcap e_2) = d \circ e_1 \sqcap d \circ e_2$.

In the rest of the report, $\mathcal{D}$ will generally denote an arbitrary qualification domain. For any finite $S = \{e_1, e_2, \ldots, e_n\} \subseteq D$, the *glb* of S (noted as $\bigsqcap S$) exists and can be computed as $e_1 \sqcap e_2 \sqcap \cdots \sqcap e_n$ (which reduces to $\top$ in the case $n = 0$). As an easy consequence of the axioms, one gets the identity $d \circ \bigsqcap S = \bigsqcap\{d \circ e \mid e \in S\}$. We generalize van Emden's QLP to a generic scheme $QLP(\mathcal{D})$ which uses qualification values $d \in D \setminus \{\bot\}$ in place of certainty values $d \in (0, 1]$, the *glb* operator $\bigsqcap$ in place of the minimum operator min, and the attenuation operator $\circ$ in place of the multiplication operator $\times$. Three interesting instances of qualification domains are shown below.

The Domain of Classical Boolean Values: $\mathcal{B} = (\{0, 1\}, \leq, 0, 1, \wedge)$, where 0 and 1 stand for the two classical truth values *false* and *true*, $\leq$ is the usual numerical ordering over $\{0, 1\}$, and $\wedge$ stands for the classical conjunction operation over $\{0, 1\}$. The instance $QLP(\mathcal{B})$ of our $QLP(\mathcal{D})$ scheme behaves as classical Logic Programming.

The Domain of van Emden's Uncertainty Values: $\mathcal{U} = (\mathrm{U}, \leq, 0, 1, \times)$, where $\mathrm{U} = [0, 1] = \{d \in \mathbb{R} \mid 0 \leq d \leq 1\}$, $\leq$ is the usual numerical ordering, and $\times$ is the multiplication operation. In this domain, the top element $\top$ is 1 and the greatest lower bound $\bigsqcap S$ of a finite $S \subseteq \mathrm{U}$ is the minimum value min(S), which is 1 if $S = \emptyset$. For this reason, the instance $QLP(\mathcal{U})$ of our $QLP(\mathcal{D})$ scheme behaves as van Emden's QLP.

The Domain of Weight Values: $\mathcal{W} = (\mathrm{P}, \geq, \infty, 0, +)$, where $\mathrm{P} = [0, \infty] = \{d \in \mathbb{R} \cup \{\infty\} \mid d \geq 0\}$, $\geq$ is the reverse of the usual numerical ordering (with $\infty \geq d$ for any $d \in \mathrm{P}$), and $+$ is the addition operation (with $\infty + d = d + \infty = \infty$ for any $d \in \mathrm{P}$). In this domain, the top element $\top$ is 0 and the greatest lower bound $\bigsqcap S$ of a finite $S \subseteq \mathrm{P}$ is the maximum value max(S), which is 0 if $S = \emptyset$. When working in the instance $QLP(\mathcal{W})$ of our $QLP(\mathcal{D})$ scheme one propagates to a clause head the qualification value $f + b$, where f is the clause's 'attenuation factor' and b is the maximum of the qualification values known for the body atoms. Therefore, qualification values in the instance $QLP(\mathcal{W})$ of our $QLP(\mathcal{D})$ scheme behave as a weighted measure of the depth of proof trees.

It is easily checked that the axioms of qualification domains are satisfied by $\mathcal{B}, \mathcal{U}$ and $\mathcal{W}$. In fact, the axioms have been chosen as a natural generalization of some basic properties satisfied by the ordering $\leq$ and the operation $\times$ in $\mathcal{U}$. In general, the values belonging to a qualification domain are intended to qualify logical assertions by measuring the degree in which they satisfy some kind of user's expectations. In this way, one can think of $\mathcal{U}$ values as measuring the degree of truth, $\mathcal{W}$ values as measuring proofs sizes, etc.

Given two qualification domains $\mathcal{D}_i = \langle D_i, \sqsubseteq_i, \bot_i, \top_i, \circ_i \rangle$ $(i \in \{1, 2\})$, their *cartesian product* $\mathcal{D}_1 \times \mathcal{D}_2$ is defined as $\mathcal{D} =_{\mathrm{def}} \langle D, \sqsubseteq, \bot, \top, \circ \rangle$, where $D =_{\mathrm{def}} D_1 \times D_2$, the partial ordering $\sqsubseteq$ is defined as $(d_1, d_2) \sqsubseteq (e_1, e_2) \Longleftrightarrow_{\mathrm{def}} d_1 \sqsubseteq_1 e_1$ and $d_2 \sqsubseteq_2 e_2$, $\bot =_{\mathrm{def}} (\bot_1, \bot_2)$, $\top =_{\mathrm{def}} (\top_1, \top_2)$, and the attenuation operator $\circ$ is defined as $(d_1, d_2) \circ (e_1, e_2) =_{\mathrm{def}} (d_1 \circ_1 e_1, d_2 \circ_2 e_2)$. The class of the qualification domains is closed under cartesian products, as stated in the following result.

Proposition 1. *The cartesian product $\mathcal{D} = \mathcal{D}_1 \times \mathcal{D}_2$ of two given qualification domains is always another qualification domain.*

Proof. According to the axiomatic definition of qualification domains, one must prove two items:

1. $\mathcal{D}$ is a lattice with extreme points $\bot$ and $\top$ w.r.t. the partial ordering $\sqsubseteq$. This is easily checked using the definition of $\sqsubseteq$ in the product domain. In particular, one gets the equalities $(d_1, d_2) \sqcap (e_1, e_2) = (d_1 \sqcap_1 e_1, d_2 \sqcap_2 e_2)$ and $(d_1, d_2) \sqcup (e_1, e_2) = (d_1 \sqcup_1 e_1, d_2 \sqcup_2 e_2)$.
2. $\circ$ satisfies the five axioms required for attenuation operators, i.e.:
   (a) $\circ$ is associative, commutative and monotonic w.r.t. $\sqsubseteq$.
   (b) $\forall (d_1, d_2) \in D_1 \times D_2 : (d_1, d_2) \circ \top = (d_1, d_2)$.
   (c) $\forall (d_1, d_2) \in D_1 \times D_2 : (d_1, d_2) \circ \bot = \bot$.
   (d) $\forall (d_1, d_2), (e_1, e_2) \in D_1 \times D_2 \setminus \{\bot, \top\} : (d_1, d_2) \circ (e_1, e_2) \sqsubset (e_1, e_2)$.
   (e) $\forall (d_1, d_2), (e_1, e_2), (e'_1, e'_2) \in D_1 \times D_2 : (d_1, d_2) \circ ((e_1, e_2) \sqcap (e'_1, e'_2)) = ((d_1, d_2) \circ (e_1, e_2)) \sqcap ((d_1, d_2) \circ (e'_1, e'_2))$.
   All these conditions are easily proved, using the hypothesis that both $\mathcal{D}_1$ and $\mathcal{D}_2$ are qualification domains as well as the construction of $\mathcal{D}$ as cartesian product of $\mathcal{D}_1$ and $\mathcal{D}_2$. □

Intuitively, each value (d_1, d_2) belonging to a product domain $\mathcal{D}_1 \times \mathcal{D}_2$ imposes the qualification d_1 *and also* the qualification d_2. In particular, values (c, d) belonging to the product domain $\mathcal{U} \times \mathcal{W}$ impose two qualifications, namely: a certainty value greater or equal than c and a proof tree with depth less or equal than d. These intuitions indeed correspond to the declarative and operational semantics formally defined in Sections 3 and 4.

3 Syntax and Semantics of QLP($\mathcal{D}$)

3.1 Programs, Interpretations and Models

We assume a *signature* Σ providing free function symbols (a.k.a. constructors) and predicate symbols. *Terms* are built from constructors and variables from a countably infinite set $\mathcal{V}ar$, disjoint from Σ. *Atoms* are of the form $p(t_1, \ldots, t_n)$ (abbreviated as $p(\overline{t_n})$) where p is a n-ary predicate symbol and t_i are terms. We write At_Σ for the set of all the atoms, called the *open Herbrand base*. A $QLP(\mathcal{D})$ program $\mathcal{P}$ is a set of *qualified definite Horn clauses* of the form $A \leftarrow d - \overline{B}$ where A is an atom, $\overline{B}$ a finite conjunction of atoms and $d \in D \setminus \{\bot\}$ is the *attenuation value* attached to the clause's implication. In $QLP(\mathcal{B})$ programs, the only choice for d is 1, standing for *true*, and therefore $QLP(\mathcal{B})$ behaves as classical LP. The following example presents two simple programs over the domains $\mathcal{U}$ and $\mathcal{W}$. It is not meant as a realistic application, but just as an illustration.

Example 1.

1. The $QLP(\mathcal{U})$ program $\mathcal{P}_\mathcal{U}$ displayed below can be understood as a *knowledge base* given by the facts for the predicates `animal`, `plant`, `human` and `eats`, along with *knowledge inference rules* corresponding to the clauses with non-empty body. The clauses for the predicate `human` specify the human beings as

the ancestors of `adam` and `eve`, with the certainty of being an actual human decreasing as one moves back along the ancestors' chain. Therefore, the certainty of being a cruel human also decreases when moving from descendants to ancestors.

```
cruel(X) <-0.90- human(X), eats(X,Y), animal(Y)
cruel(X) <-0.40- human(X), eats(X,Y), plant(Y)

animal(bird) <-1.0-        human(adam) <-1.0-
animal(cat) <-1.0-         human(eve) <-1.0-
plant(oak) <-1.0-          human(father(X)) <-0.90- human(X)
plant(apple) <-1.0-        human(mother(X)) <-0.90- human(X)

eats(adam, X) <-0.80-
eats(eve,X) <-0.30- animal(X)
eats(eve,X) <-0.60- plant(X)
eats(father(X),Y) <-0.80- eats(X,Y)
eats(mother(X),Y) <-0.70- eats(X,Y)
```

2. The $QLP(\mathcal{W})$ program $\mathcal{P}_{\mathcal{W}}$ is very similar to $\mathcal{P}_{\mathcal{U}}$, except that the attenuation value `1` is attached to all the clauses. Therefore, each clause is intended to convey the information that the depth of a proof tree for the head is 1 plus the maximum depth of proof trees for the atoms in the body. As we will see, qualification constraints over $\mathcal{W}$ can be used to impose upper bounds to the depths of proof trees when solving goals w.r.t. $\mathcal{P}_{\mathcal{W}}$.

Note that the two programs in this example are different qualified versions of the classical LP program $\mathcal{P}$ obtained by dropping all the annotations. Due to the left recursion in the clauses for the predicates `human` and `eats`, some goals for $\mathcal{P}$ have an infinite search space where SLD resolution with a leftmost selection strategy would fail to compute some expected answers. For instance, the answer $\{\mathtt{X} \mapsto \mathtt{mother(eve)}, \mathtt{Y} \mapsto \mathtt{apple}\}$ would not be computed for the goal `eats(X,Y)`. However, when solving goals for the qualified programs $\mathcal{P}_{\mathcal{U}}$ and $\mathcal{P}_{\mathcal{W}}$ using the resolution method presented in Section 4, qualification constraints can be used for imposing bounds to the search space, so that even the leftmost selection strategy leads to successful computations. □

As shown in the example, clauses contain classic atoms in both their head and their body. But for our semantics, we will be interested in not only proving that we can infer an atom for a given program, but proving that we can infer it with qualification greater or equal than some given value. For this reason, we introduce *$\mathcal{D}$-annotated atoms* $A \sharp d$, consisting of an atom A with an attached 'annotation' $d \in D \setminus \{\bot\}$. For use in goals to be solved, we consider also *open annotated atoms* of the form $A \sharp W$, where W is a *qualification variable* intended to take values over $D \setminus \{\bot\}$. We postulate a countably infinite set $\mathcal{V}ar$ of qualification variables, disjoint from $\mathcal{V}ar$ and Σ.

The *annotated Herbrand base* over $\mathcal{D}$ is defined as the set $At_\Sigma(\mathcal{D})$ of all $\mathcal{D}$-annotated atoms. The *$\mathcal{D}$-entailment relation* over $At_\Sigma(\mathcal{D})$ is defined as follows: $A \sharp d \succcurlyeq_\mathcal{D} A' \sharp d'$ iff there is some substitution θ such that $A' = A\theta$ and $d' \sqsubseteq d$. Finally, we define an *open Herbrand interpretation* over $\mathcal{D}$ as any subset $\mathcal{I} \subseteq At_\Sigma(\mathcal{D})$ which is closed under $\mathcal{D}$-entailment. That is, an open Herbrand interpretation $\mathcal{I}$ including a given annotated atom $A \sharp d$ is required to include all the 'instances' $A' \sharp d'$ such that $A \sharp d \succcurlyeq_\mathcal{D} A' \sharp d'$, because we intend to formalize a semantics such that all such instances are valid whenever $A \sharp d$ is valid.

In the sequel we refer to open Herbrand interpretations just as Herbrand interpretations, and we write $\mathrm{Int}_\Sigma(\mathcal{D})$ for the family of all Herbrand interpretations over $\mathcal{D}$. The following proposition is easy to prove from the definition of a Herbrand interpretation and the definitions of the union and intersection of a family of sets.

Proposition 2. *The family $\mathrm{Int}_\Sigma(\mathcal{D})$ of all Herbrand interpretations over $\mathcal{D}$ is a complete lattice under the inclusion ordering $\subseteq$, whose extreme points are $\mathrm{Int}_\Sigma(\mathcal{D})$ as maximum and $\emptyset$ as minimum. Moreover, given any family of interpretations $I \subseteq \mathrm{Int}_\Sigma(\mathcal{D})$, its lub and glb are $\bigsqcup I = \bigcup\{\mathcal{I} \in \mathrm{Int}_\Sigma(\mathcal{D}) \mid \mathcal{I} \in I\}$ and $\bigsqcap I = \bigcap\{\mathcal{I} \in \mathrm{Int}_\Sigma(\mathcal{D}) \mid \mathcal{I} \in I\}$, respectively.* □

Let C be any clause $A \leftarrow d - B_1, \ldots, B_k$ in the program $\mathcal{P}$, and $\mathcal{I} \in \mathrm{Int}_\Sigma(\mathcal{D})$ any interpretation over $\mathcal{D}$. We say that $\mathcal{I}$ is a *model* of C if and only if for any substitution θ and any qualification values $d_1, \ldots, d_k \in D \setminus \{\bot\}$ such that $B_i\theta \sharp d_i \in \mathcal{I}$ for all $1 \leq i \leq k$, one has $A\theta \sharp (d \circ \bigsqcap\{d_1, \ldots, d_k\}) \in \mathcal{I}$. And we say that $\mathcal{I}$ is a model of the $QLP(\mathcal{D})$ program $\mathcal{P}$ (in symbols, $\mathcal{I} \models \mathcal{P}$) if and only if $\mathcal{I}$ is a model of each clause in $\mathcal{P}$.

3.2 Declarative Semantics

As in any logic language, we need some technique to infer formulas (in our case, $\mathcal{D}$-annotated atoms) from a given $QLP(\mathcal{D})$ program $\mathcal{P}$. Following traditional ideas, we consider two alternative ways of formalizing an inference step which goes from the body of a clause to its head: an operator $\mathrm{T}_\mathcal{P}$ and a qualified variant of Horn Logic, noted as $QHL(\mathcal{D})$ and called *Qualified Horn Logic*. The operator $\mathrm{T}_\mathcal{P} : \mathrm{Int}_\Sigma(\mathcal{D}) \to \mathrm{Int}_\Sigma(\mathcal{D})$ is defined as:

$$\begin{aligned}\mathrm{T}_\mathcal{P}(\mathcal{I}) =_{\mathrm{def}} \{A' \sharp d' \mid\ & (A \leftarrow d - B_1, \ldots, B_k) \in \mathcal{P},\\ & \theta \text{ subst.}, B_i\theta \sharp d_i \in \mathcal{I} \text{ for all } 1 \leq i \leq k,\ A' = A\theta,\\ & d' \in D \setminus \{\bot\},\ d' \sqsubseteq d \circ \bigsqcap\{d_1, \ldots, d_k\}\}\end{aligned}$$

Intuitively, we can see that for a given interpretation $\mathcal{I}$, $\mathrm{T}_\mathcal{P}(\mathcal{I})$ is the set of those $\mathcal{D}$-annotated atoms obtained by considering $\mathcal{D}$-annotated bodies of clause instances that are included in $\mathcal{I}$ and propagating an annotation to the head via the clause's qualification value.

The logic $QHL(\mathcal{D})$ is defined as a deductive system consisting just of one inference rule $\mathrm{QMP}(\mathcal{D})$, called *Qualitative Modus Ponens* over $\mathcal{D}$. If there are some $(A \leftarrow d - B_1, \ldots, B_k) \in \mathcal{P}$, some substitution θ such that $A' = A\theta$ and

$B'_i = B_i\theta$ for all $1 \leq i \leq k$ and $d' \sqsubseteq d \circ \bigsqcap\{d_1, \ldots, d_k\}$, the following inference step is allowed:

$$\frac{B'_1 \sharp d_1 \quad \cdots \quad B'_k \sharp d_k}{A' \sharp d'} \quad \mathrm{QMP}(\mathcal{D})$$

We will use the notations $\mathcal{P} \vdash_{\mathrm{QHL}(\mathcal{D})} A \sharp d$ (resp. $\mathcal{P} \vdash^n_{\mathrm{QHL}(\mathcal{D})} A \sharp d$) to indicate that $A \sharp d$ can be inferred from the clauses in program $\mathcal{P}$ in finitely many steps (resp. n steps). Note that $QHL(\mathcal{D})$ proofs can be naturally represented as upwards growing *proof trees* with $\mathcal{D}$-annotated atoms at their nodes, each node corresponding to one inference step having the children nodes as premises.

The following proposition collects the main results concerning the declarative semantics of the $QLP(\mathcal{D})$ scheme. We just sketch some key proof ideas. The full proofs are given in [17]. As in [24], they can be developed in analogy to the classical papers [25,1], except that our Herbrand interpretations are open, as first suggested by Clark in [4]. Our use of the $QHL(\mathcal{D})$ calculus is obviously related to the classical $\mathrm{T}_\mathcal{P}$ operator, although it has no direct counterpart in the historical papers we are aware of.

Proposition 3. *The following assertions hold for any $QLP(\mathcal{D})$ program $\mathcal{P}$:*

1. $\mathcal{I} \models \mathcal{P} \iff \mathrm{T}_\mathcal{P}(\mathcal{I}) \subseteq \mathcal{I}$.
2. *$\mathrm{T}_\mathcal{P}$ is monotonic and continuous.*
3. *The least fixpoint $\mu(\mathrm{T}_\mathcal{P})$ is the least Herbrand model of $\mathcal{P}$, noted as $\mathcal{M}_\mathcal{P}$.*
4. $\mathcal{M}_\mathcal{P} = \bigcup_{n \in \mathbb{N}} \mathrm{T}_\mathcal{P} \uparrow^n (\emptyset) = \{A \sharp d \mid \mathcal{P} \vdash_{\mathrm{QHL}(\mathcal{D})} A \sharp d\}$.

Proof (Sketch). Item (1) is easy to prove from the definition of $\mathrm{T}_\mathcal{P}$. In item (2), monotonicity ($\mathcal{I} \subseteq \mathcal{J} \Longrightarrow \mathrm{T}_\mathcal{P}(\mathcal{I}) \subseteq \mathrm{T}_\mathcal{P}(\mathcal{J})$) follows easily from the definition of $\mathrm{T}_\mathcal{P}$ and continuity ($\mathrm{T}_\mathcal{P}(\bigcup_{n\in\mathbb{N}} \mathcal{I}_n) = \bigcup_{n\in\mathbb{N}} \mathrm{T}_\mathcal{P}(\mathcal{I}_n)$ for any chain $\{\mathcal{I}_n \mid n \in \mathbb{N}\} \subseteq \mathrm{Int}_\Sigma(\mathcal{D})$ with $\mathcal{I}_n \subseteq \mathcal{I}_{n+1}$ for all $n \in \mathbb{N}$) follows from monotonicity and properties of chains and sets of interpretations. Item (3) follows from (1), (2), Proposition 2 and some known properties about lattices. Finally, item (4) follows from proving the two implications $\mathcal{P} \vdash^n_{\mathrm{QHL}(\mathcal{D})} A \sharp d \Longrightarrow \exists m\, (A \sharp d \in \mathrm{T}_\mathcal{P} \uparrow^m (\emptyset))$ and $A \sharp d \in \mathrm{T}_\mathcal{P} \uparrow^n (\emptyset) \Longrightarrow \exists m\, (\mathcal{P} \vdash^m_{\mathrm{QHL}(\mathcal{D})} A \sharp d)$ by induction on n. □

The next example presents proofs deriving annotated atoms that belong to the least models of the programs $\mathcal{P}_\mathcal{U}$ and $\mathcal{P}_\mathcal{W}$ from Example 1.

Example 2.

1. The proof tree displayed below shows that the $\mathcal{U}$-annotated atom at its root can be deduced from $\mathcal{P}_\mathcal{U}$ in $QHL(\mathcal{U})$. Therefore, the atom belongs to $\mathcal{M}_{\mathcal{P}_\mathcal{U}}$.

$$\dfrac{\dfrac{}{\texttt{human(eve)\#1.0}}}{\texttt{human(mother(eve))\#0.90}} \quad \dfrac{\dfrac{}{\texttt{animal(bird)\#1.0}}}{\texttt{eats(eve,bird)\#0.30}} \quad \dfrac{}{\texttt{animal(bird)\#1.0}}$$
$$\texttt{eats(mother(eve),bird)\#0.21}$$
$$\texttt{cruel(mother(eve))\#0.15}$$

   It is easy to see which clause was used in each inference step. Note that the atom at the root could have been proved even with the greater certainty value `0.189`. However, since `0.15` $\leq$ `0.189`, the displayed inference it is also correct (albeit less informative).

2. A proof tree quite similar to the previous one, but with different annotations, can be easily built to show that `cruel(mother(eve))#4` can be deduced from $\mathcal{P}_{\mathcal{W}}$ in $QHL(\mathcal{W})$. Therefore, this annotated atom belongs to $\mathcal{M}_{\mathcal{P}_{\mathcal{W}}}$. It conveys the information that `cruel(mother(eve))` has a proof tree of depth `4` w.r.t. to the classical LP program $\mathcal{P}$ obtained by dropping $\mathcal{P}_{\mathcal{W}}$'s annotations. □

4 Goal Solving by SLD($\mathcal{D}$) Resolution

4.1 Goals and Solutions

In classical logic programming a goal is presented as a conjunction of atoms. In our setting, proving atoms with arbitrary qualifications may be unsatisfactory, since qualification values too close to $\bot$ may not ensure sufficient information. For this reason, we present goals as conjunctions of open $\mathcal{D}$-annotated atoms and we indicate the minimum qualification value required each of them. Hence initial goals look like: $A_1 \sharp W_1, \ldots, A_n \sharp W_n \parallel W_1 \sqsupseteq \beta_1, \ldots, W_n \sqsupseteq \beta_n$, where $W_i \in \mathcal{W}ar$ and $\beta_i \in D \setminus \{\bot\}$. Observe that we have annotated all atoms in the goal with qualification variables W_i instead of plain values because we are interested in any solution that satisfies the *qualification constraints* $W_i \sqsupseteq \beta_i$, used to impose lower bounds to the atoms' qualifications.

As explained in the next Subsection, goal resolution proceeds from an initial goal through intermediate goals until reaching a final solved goal. The intermediate goals have a more general form, consisting of a composition of three items: a conjunction of $\mathcal{D}$-annotated atoms $\overline{A}$ waiting to be solved, a substitution σ computed in previous steps, and a set of qualification constraints Δ. We consider two kinds of qualification constraints:

1. $\alpha \circ W \sqsupseteq \beta$, where $W \in \mathcal{W}ar$ is qualification variable and $\alpha, \beta \in D \setminus \{\bot\}$ are such that $\alpha \sqsupseteq \beta$. This is called a *threshold constraint* for W.
2. $W = d \circ \bigsqcap\{W_1, \ldots, W_k\}$, where $W, W_1, \ldots, W_k \in \mathcal{W}ar$ are qualification variables and $d \in D \setminus \{\bot\}$. This is called a *defining constraint* for W.

In order to understand why these two kinds of constraints are needed, think of an annotated atom $A \sharp W$ within an initial goal which includes also an initial threshold constraint $\top \circ W \sqsupseteq \beta$ (i.e. $W \sqsupseteq \beta$) for W. Applying a resolution step with a program clause whose head unifies with A and whose attenuation value is $d \in D \setminus \{\bot\}$ will lead to a new goal including a defining constraint $W = d \circ \bigsqcap\{W_1, \ldots, W_k\}$ for W and a threshold constraint $d \circ \top \circ W_i \sqsupseteq \beta$ for each $1 \leq i \leq k$, where the new qualification variables W_i correspond to the atoms in the clause's body. This explains the need to introduce defining constraints as well as more general threshold constraints $\alpha \circ W \sqsupseteq \beta$. Intuitively, the values α and β within such constraints play the role of an *upper* and a *lower* bound, respectively. As we will see, our goal solving procedure takes advantage of these bounds for pruning useless parts of the computation search space.

Let us now present some notations needed for a formal definition of goals. Given a conjunction of $\mathcal{D}$-annotated atoms $\overline{A}$ and a set of qualification constraints Δ, we define the following sets of variables:

- $\mathrm{var}(\overline{A}) =_{\mathrm{def}} \bigcup\{\mathrm{var}(A) \mid A \sharp W \in \overline{A}\}$.
- $\mathrm{war}(\overline{A}) =_{\mathrm{def}} \bigcup\{W \mid A \sharp W \in \overline{A}\}$.
- $\mathrm{war}(\Delta)$ as the set of qualification variables that appears in any qualification constraint in Δ.
- $\mathrm{dom}(\Delta)$ as the set of qualification variables that appear in the left hand side of any qualification constraint in Δ.

We also say that Δ is *satisfiable* iff there is some $\omega \in \mathrm{Subst}_\Sigma(\mathcal{D})$ –the set of all the substitutions of values in $D\backslash\{\bot\}$ for variables in $\mathcal{V}ar$– such that $\omega \in Sol(\Delta)$, what means that ω satisfies every qualification constraint in Δ, i.e. ω is a *solution* of Δ. Moreover, we say that Δ is *admissible* iff it satisfies the following three conditions:

1. Δ is satisfiable,
2. for every $W \in \mathrm{war}(\Delta)$ there exists one and only one constraint for W in Δ (this implies $\mathrm{dom}(\Delta) = \mathrm{war}(\Delta)$), and
3. the relation $>_\Delta$ defined by $W >_\Delta W_i$ iff there is some defining constraint $W = \alpha \circ \bigsqcap\{W_1, \ldots, W_i, \ldots, W_k\}$ in Δ, satisfies that $>_\Delta^*$ is irreflexive.

Finally, we say that Δ is *solved* iff Δ is admissible and only contains defining constraints. Now we are in a position to define *goals* and their *solutions*:

Definition 1 (Goals and its Variables). *Given a conjunction of $\mathcal{D}$-annotated atoms $\overline{A}$, a substitution $\sigma \in \mathrm{Subst}_\Sigma$ –the set of all substitutions of terms for variables in $\mathcal{V}ar$– and a set of qualification constraints Δ, we say that $G \equiv \overline{A} \,[\!]\, \sigma \,[\!]\, \Delta$ is a* goal *iff*

i. $\sigma \in \mathrm{Subst}_\Sigma$ is idempotent and such that $dom(\sigma) \cap var(\overline{A}) = \emptyset$.
ii. Δ is admisible.
iii. For every qualification variable in $\mathrm{war}(\overline{A})$ there is one and only one threshold constraint for W in Δ. And there are no more threshold constraints in Δ.

Furthermore, if $\sigma = \epsilon$ (the identity substitution) then G is called initial*, and if $\overline{A}$ is empty and Δ is solved, then G is called* solved*. For any goal G, we define the set of variables of G as $\mathrm{var}(G) =_{\mathrm{def}} \mathrm{var}(\overline{A}) \cup \mathrm{dom}(\sigma)$ and the set of qualification variables of G as $\mathrm{war}(G) =_{\mathrm{def}} \mathrm{war}(\overline{A}) \cup \mathrm{dom}(\Delta)$.* □

Definition 2 (Goal Solutions). *A pair of substitutions (θ, ρ) such that $\theta \in \mathrm{Subst}_\Sigma$ and $\rho \in \mathrm{Subst}_\Sigma(\mathcal{D})$ is called a* solution *of a goal $G \equiv \overline{A} \,[\!]\, \sigma \,[\!]\, \Delta$ iff:*

1. $\theta = \sigma\theta$.
2. $\rho \in Sol(\Delta)$.
3. $\mathcal{P} \vdash_{\mathrm{QHL}(\mathcal{D})} A\theta \sharp W\rho$ for all $A \sharp W \in \overline{A}$.

In addition, a solution (σ, μ) *for a goal* G *is said to be more general than another solution* (θ, ρ) *for the same goal* G *(one also says in this case that* (θ, ρ) *is subsumed by* (σ, μ)*) iff* $\sigma \preccurlyeq \theta$ $[\mathrm{var}(G)]$ *and* $\mu \sqsupseteq \rho$ $[\mathrm{war}(G)]$*, where* $\sigma \preccurlyeq \theta$ $[\mathrm{var}(G)]$ *means that there is some substitution* η *such that the composition* $\sigma\eta$ *behaves the same as* θ *over any variable in the set* $\mathrm{var}(G)$ *and* $\mu \sqsupseteq \rho$ $[\mathrm{war}(G)]$ *means that* $\mu(W) \sqsupseteq \rho(W)$ *holds for any* $W \in \mathrm{war}(G)$. □

Any solved goal $G' \equiv \sigma \parallel \Delta$ has the *associated solution* (σ, μ), where $\mu = \omega_\Delta$ is the qualification substitution given by Δ, such that $\omega_\Delta(W)$ is the qualification value determined by the defining constraints in Δ for all $W \in \mathrm{dom}(\Delta)$, and $\omega_\Delta(W) = \bot$ for any $W \in \mathcal{W}ar \setminus \mathrm{dom}(\Delta)$. Note that for any $W \in \mathrm{dom}(\Delta)$ there exists one unique defining constraint $W = d \circ \bigsqcap\{W_1, \ldots, W_k\}$ for W in Δ, and then $\omega_\Delta(W)$ can be recursively computed as $d \circ \bigsqcap\{\omega_\Delta(W_1), \ldots, \omega_\Delta(W_k)\}$. The solutions associated to solved goals are called *computed answers*.

Example 3.

1. A possible goal for program $\mathcal{P}_\mathcal{U}$ in Example 1 is `eats(father(X),Y)#W1, human(father(X))#W2 | W1>=0.4, W2>=0.6`; and a valid solution for it is {`X` ↦ `adam, Y` ↦ `apple`} | {`W1` ↦ `0.50, W2` ↦ `0.75`}.
2. A goal for program $\mathcal{P}_\mathcal{W}$ in Example 1 may be `eats(X,Y)#W | W<=5.0`; and a valid solution is {`X` ↦ `father(adam), Y` ↦ `apple`} | {`W` ↦ `4.0`}. □

Note that the goal for $\mathcal{P}_\mathcal{U}$ in the previous example imposes lower bounds to the certainties to be computed, while the goal for $\mathcal{P}_\mathcal{W}$ imposes an upper bound to the proof depth. In general, goal solving in $QLP(\mathcal{W})$ corresponds to depth-bound goal-golving in classical Logic Programming.

4.2 SLD($\mathcal{D}$) Resolution

We propose a sound and strongly complete goal solving procedure called *Qualified SLD Resolution* parameterized over a given qualification domain $\mathcal{D}$, written as $SLD(\mathcal{D})$, which makes use of *annotated atoms* and *qualification constraints* over $\mathcal{D}$. The implementation of this goal solving procedure using CLP technology will be discussed in the next section. Resolution computations are written $G_0 \Vdash_{C_1,\sigma_1} G_1 \Vdash_{C_2,\sigma_2} \cdots \Vdash_{C_n,\sigma_n} G_n$, abbreviated as $G_0 \Vdash^*_\sigma G_n$ with $\sigma = \sigma_1\sigma_2\cdots\sigma_n$. They are finite sequences of resolution steps $G_{i-1} \Vdash_{C_i,\sigma_i} G_i$, starting with an initial goal G_0 and ending up with a solved goal G_n. One single resolution step is formally defined as follows:

Definition 3 (Resolution step). *A* resolution step *has the form* $\overline{L}, A \sharp W, \overline{R} \parallel \sigma \parallel \alpha \circ W \sqsupseteq \beta, \Delta \Vdash_{C_1,\sigma_1} (\overline{L}, B_1 \sharp W_1, \ldots B_k \sharp W_k, \overline{R})\sigma_1 \parallel \sigma\sigma_1 \parallel \Delta_1$ *where* $A \sharp W$ *is the* selected atom*,* $\Delta_1 = d \circ \alpha \circ W_1 \sqsupseteq \beta, \ldots, d \circ \alpha \circ W_k \sqsupseteq \beta, W = d \circ \bigsqcap\{W_1, \ldots, W_k\}, \Delta$*,* $C_1 \equiv (H \leftarrow d - B_1, \ldots, B_k) \in_{\mathrm{var}} \mathcal{P}$ *is chosen as a variant of a clause in* $\mathcal{P}$ *with fresh variables and such that* $d \circ \alpha \sqsupseteq \beta$*,* σ_1 *is the m.g.u. between* A *and* H*, and* $W_1, \ldots, W_k \in \mathcal{W}ar$ *are fresh qualification variables.* □

The notation $\alpha \circ W \sqsupseteq \beta, \Delta$ represents a set of qualification constraints including the threshold constraint $\alpha \circ W \sqsupseteq \beta$ plus those in Δ, with no particular ordering assumed. Notice that the condition $d \circ \alpha \sqsupseteq \beta$ is required for the resolution step to be enabled. In this way, threshold constraints $\alpha \circ W \sqsupseteq \beta$ are actively used for pruning parts of the computation search space where no solutions can be found. In the instance of $QLP(\mathcal{B})$ it is easily checked that all the qualification values and constraints become trivial, so that $SLD(\mathcal{B})$ boils down to classical SLD resolution. In the rest of this section we present the main properties of $SLD(\mathcal{D})$ resolution in the general case.

Proposition 4. *If G is a goal and $G_0 \Vdash_{C_1,\sigma_1} G_1$, then G_1 is also a goal.*

Proof (Sketch). Assume a goal G_0 and a $SLD(\mathcal{D})$ resolution step $G_0 \Vdash_{C_1,\sigma_1} G_1$, as in Definition 3. Then G_0 satisfies the conditions required for goals in Definition 1, and we must show that G_1 also satisfies such conditions. This is not difficult to check, using the fact that C_1 has been chosen without variables in common with G_0. In particular, note that the threshold constraint for W in G_0 is absent in G_1, which includes a defining constraint for W and threshold constraints for the new qualification variables W_i. □

The next two theorems are the main theoretical results in this report. The Soundness Theorem 1 guarantees that every computed answer is correct in the sense that it is a solution of the given goal. The Strong Completeness Theorem 2 ensures that, for any solution of a given goal and any fixed selection strategy, $SLD(\mathcal{D})$ resolution is able to compute an equal, if not better, solution. The proofs, given in Appendix A, use inductive techniques similar to those presented in [20] for classical SLD resolution. Example 4 below illustrates the Completeness Theorem.

Theorem 1 (Soundness). *Assume $G_0 \Vdash^* G$ and $G = \sigma \parallel \Delta$ solved. Let (σ, μ) be the solution associated to G. Then (σ, μ) –called the computed answer– is a solution of G_0.* □

Theorem 2 (Strong Completeness). *Assume a given solution (θ, ρ) for G_0 and any fixed strategy for choosing the selected atom at each resolution step. Then there is some computed answer (σ, μ) for G_0 which subsumes (θ, ρ).* □

Example 4.

1. The following $SLD(\mathcal{U})$ computation solves the goal for program $\mathcal{P}_{\mathcal{U}}$ presented in Example 3:

```
eats(father(X),Y)#W1,
     human(father(X))#W2 |
     W1 >= 0.4, W2 >= 0.6                          ⊩_{eats.4,{X↦adam}}
eats(adam,Y)#W3,
     human(father(adam))#W2 | {X ↦ adam} |
     W1 = 0.8 * min{W3},
     W2 >= 0.6, 0.8 * W3 >= 0.4                    ⊩_{eats.1,ϵ}
human(father(adam))#W2 | {X ↦ adam} |
     W1 = 0.8 * min{W3},
     W2 >= 0.6, W3 = 0.8                           ⊩_{human.3,ϵ}
human(adam)#W4 | {X ↦ adam} |
     W1 = 0.8 * min{W3},
     W2 = 0.9 * min{W4},
     W3 = 0.8, 0.90 * W4 >= 0.6                    ⊩_{human.1,ϵ}
| {X ↦ adam} |
     W1 = 0.8 * min{W3},
     W2 = 0.9 * min{W4},
     W3 = 0.8, W4 = 1.0
```

Note that the computed answer {X ↦ adam} | {W1 ↦ 0.64, W2 ↦ 0.90} subsumes the solution for the same goal given in Example 3.

2. Similarly, $SLD(\mathcal{W})$ resolution can solve the goal `eats(X,Y)#W | W <= 5.0` for $\mathcal{P}_{\mathcal{W}}$, obtaining a computed answer {X ↦ father(adam)} | {W ↦ 3.0} which subsumes the solution for the same goal given in Example 3. □

5 Towards an Implementation

In this section we assume a qualification domain $\mathcal{D}$ and a constraint domain $\mathcal{C}_{\mathcal{D}}$ such that the qualification constraints used in $SLD(\mathcal{D})$ resolution can be expressed as $\mathcal{C}_{\mathcal{D}}$ constraints, and we describe a translation of $QLP(\mathcal{D})$ programs $\mathcal{P}$ and goals G into $CLP(\mathcal{C}_{\mathcal{D}})$ programs $\mathcal{P}^t$ and goals G^t, such that solving G with $SLD(\mathcal{D})$ resolution using $\mathcal{P}$ corresponds to solving G^t with constrained SLD resolution using $\mathcal{P}^t$ and a solver for $\mathcal{C}_{\mathcal{D}}$.

The translation can be used to develop an implementation of $SLD(\mathcal{D})$ resolution for the $QLP(\mathcal{D})$ language on top of any CLP or $CFLP$ system that supports $\mathcal{C}_{\mathcal{D}}$ constraints. In particular, if $\mathcal{D}$ is any of the two qualification domains $\mathcal{U}$ or $\mathcal{W}$, the constraint domain $\mathcal{C}_{\mathcal{D}}$ can be chosen as $\mathcal{R}$, which supports arithmetic constraints over the real numbers [10]. We have developed prototype implementations for $QLP(\mathcal{U})$, $QLP(\mathcal{W})$ and $QLP(\mathcal{U} \times \mathcal{W})$ on top of the $CFLP$ system $\mathcal{TOY}$ [3], that supports $\mathcal{R}$ constraints. Note that although the use of a $CLP(\mathcal{R})$ system could lead to a more efficient implementation, we have chosen a $CFLP(\mathcal{R})$ system instead of a $CLP(\mathcal{R})$ one due to our interest in a future extension of the $QLP(\mathcal{D})$ scheme to support qualified $CFLP$ programming.

Our translation of a $QLP(\mathcal{D})$ program works by adding three extra arguments to all predicates and translating each clause independently. Given the

$QLP(\mathcal{D})$ clause

$$C \equiv p(\bar{t}) \leftarrow d - q_1(\overline{s_1}), \ldots, q_k(\overline{s_k})$$

its head is translated as $p(\bar{t}, Alpha, W, Beta)$, where the new variables $Alpha$, W and $Beta$ correspond, respectively, to α, W and β in the threshold constraint $\alpha \circ W \sqsupseteq \beta$ related to a $\mathcal{D}$-annotated atom $A \sharp W$ which could be selected for a $SLD(\mathcal{D})$ resolution step using the clause C. The clause's body is translated with the aim of emulating such a resolution step, and the translated clause becomes:

$$\begin{aligned} C^t \equiv p(\bar{t}, Alpha, W, Beta) \leftarrow\ & d \circ Alpha \sqsupseteq Beta, \\ & W_1 \sqsupset \bot, W_1 \sqsubseteq \top, q_1(\overline{s_1}, d \circ Alpha, W_1, Beta), \\ & \vdots \\ & W_k \sqsupset \bot, W_k \sqsubseteq \top, q_k(\overline{s_k}, d \circ Alpha, W_k, Beta), \\ & W = d \circ \bigsqcap\{W_1, \ldots, W_k\} \end{aligned}$$

The conditions in the body of C^t do indeed correspond to the performance of a $SLD(\mathcal{D})$ resolution step with clause C. In fact, $d \circ Alpha \sqsupseteq Beta$ checks that C is eligible for such a step; the conditions in the next k lines using new variables W_i correspond to placing the annotated atoms from C's body into the new goal; and the last condition introduces the proper defining constraint for W.

The idea for translating goals is similar. Given an initial goal $QLP(\mathcal{D})$ goal G like

$$q_1(\overline{t_1}) \sharp W_1, \ldots, q_m(\overline{t_m}) \sharp W_m \ [\!] \ W_1 \sqsupseteq \beta_1, \ldots, W_m \sqsupseteq \beta_m$$

where $\beta_1, \ldots, \beta_m \in D \setminus \{\bot\}$, the translated goal G^t is

$$q_1(\overline{t_1}, \top, W_1, \beta_1), \ldots, q_m(\overline{t_m}, \top, W_m, \beta_m)$$

where the three additional arguments at each atom are used to encode the initial threshold constraints $W_i \sqsupseteq \beta_i$, that are equivalent to $\top \circ W_i \sqsupseteq \beta_i$.

Example 5. As an example of the translation process we present the translation of the program $\mathcal{P}_\mathcal{U}$ from Example 1 into a $\mathcal{TOY}$ program which uses $\mathcal{R}$ constraints.

```
min1 [] = 1
min1 [X|Xs] = min2 X (min1 Xs)
min2 W1 W2 = if W1 <= W2 then W1 else W2
data being = adam | eve | bird | cat | oak | apple
           | father being | mother being
cruel(X,F,W,M) :- F*0.9>=M, W1>0, W1<=1.0, human(X,F*0.9,W1,M),
                  W2>0, W2<=1.0, eats(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, human(X,F*0.4,W1,M),
                  W2>0, W2<=1.0, eats(X,Y,F*0.4,W2,M),
                  W3>0, W3<=1.0, plant(Y,F*0.4,W3,M),
                  W == 0.4 * min1 [W1,W2,W3]
```

```
animal(bird,F,W,M) :- F*1.0>=M, W == 1.0 * min1 []
animal(cat,F,W,M)  :- F*1.0>=M, W == 1.0 * min1 []
plant(oak,F,W,M)   :- F*1.0>=M, W == 1.0 * min1 []
plant(apple,F,W,M) :- F*1.0>=M, W == 1.0 * min1 []
human(adam,F,W,M)      :- F*1.0>=M, W == 1.0 * min1 []
human(eve,F,W,M)       :- F*1.0>=M, W == 1.0 * min1 []
human(father(X),F,W,M) :- F*0.9>=M, W1>0, W1<=1.0,
      human(X,F*0.9, W1, M), W == 0.9 * min1 [W1]
human(mother(X),F,W,M) :- F*0.8>=M, W1>0, W1<=1.0,
      human(X,F*0.8, W1, M), W == 0.8 * min1 [W1]
eats(adam,X,F,W,M) :- F*0.8>=M, W == 0.8 * min1 []
eats(eve,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]
eats(eve,X,F,W,M) :- F*0.6>=M, W1>0, W1<=1.0,
      plant(X,F*0.6,W1,M), W == 0.6 * min1 [W1]
eats(father(X),Y,F,W,M) :- F*0.8>=M, W1>0, W1<=1.0,
      eats(X,Y,F*0.8,W1,M), W == 0.8 * min1 [W1]
eats(mother(X),Y,F,W,M) :- F*0.7>=M, W1>0, W1<=1.0,
      eats(X,Y,F*0.7,W1,M), W == 0.7 * min1 [W1]
```

To understand this example it is important to notice the following:

1. Since *glb*s in $\mathcal{U}$ are computed as minimums, translated programs must include functions for this task. Here, `min1` resp. `min2` compute the minimum of a list of numbers resp. two numbers.
2. As $\mathcal{TOY}$ need types for every constructor, we must include suitable datatype declarations in translated programs.
3. The resulting code could be simplified and optimized, but our aim here is to illustrate the literal application of the general translation rules. For this reason, no optimizations have been performed. □

6 Conclusions and Future Work

We have generalized the early QLP proposal by van Emden [24] to a generic scheme $QLP(\mathcal{D})$ parameterized by a qualification domain $\mathcal{D}$, which must be a lattice with extreme points and equipped with an attenuation operator. The values belonging to a qualification domain are intended to qualify logical assertions, ensuring that they satisfy certain user's expectations. Qualification domains include $\mathcal{B}$ (classical truth values of two-valued logic), $\mathcal{U}$ (van Emden's certainty values) and $\mathcal{W}$ (numeric values representing proof weights), as well as arbitrary cartesian products of given qualification domains. As shown by instances such as $QLP(\mathcal{W})$ and $QLP(\mathcal{U} \times \mathcal{W})$, the $QLP(\mathcal{D})$ scheme can express uncertainty in Logic Programming and more, since the user's expectations qualified by $\mathcal{W}$ do not correspond to uncertain truth values.

The semantic results obtained for $QLP(\mathcal{D})$ are stronger than those in [24]. Each program $\mathcal{P}$ has a least open Herbrand model $\mathcal{M}_{\mathcal{P}}$ with two equivalent

characterizations: as the least fixpoint of the operator $T_{\mathcal{P}}$, and as the set of qualified atoms deducible from $\mathcal{P}$ in the logic calculus $QHL(\mathcal{D})$. Moreover, the goal solving calculus $SLD(\mathcal{D})$, based on an extension of SLD resolution with qualification constraints, is sound and strongly complete for arbitrary open goals. $SLD(\mathcal{B})$ boils down to classical SLD resolution.

As implementation technique, we have proposed a translation of $QLP(\mathcal{D})$ programs and goals into $CLP(\mathcal{C}_{\mathcal{D}})$, choosing a constraint domain $\mathcal{C}_{\mathcal{D}}$ able to compute with qualification constraints over $\mathcal{D}$. If $\mathcal{D}$ is $\mathcal{U}$, $\mathcal{B}$, or $\mathcal{U} \times \mathcal{B}$, the constraint domain $\mathcal{C}_{\mathcal{D}}$ can be chosen as $\mathcal{R}$, and $QLP(\mathcal{D})$ can be implemented on top of any CLP or $CFLP$ system which supports constraint solving over $\mathcal{R}$. We have implemented prototypes of $QLP(\mathcal{U})$, $QLP(\mathcal{W})$ and $QLP(\mathcal{U} \times \mathcal{W})$ on top of the $CFLP$ system $\mathcal{TOY}$.

In comparison to the theory of generalized annotated logic programs (GAP for short) presented in [11], our results in this report also include some interesting contributions. With respect to the syntax and goal solving procedure, the $QLP(\mathcal{D})$ scheme can be made to fit into the GAP framework by viewing our attenuation operators as annotation functions. However, our resolution procedure $SLD(\mathcal{D})$ can be implemented more efficiently than the constrained SLD resolution used in GAP, due to an optimized treatment of qualification constraints and, more importantly, because the costly computation of so-called *reductants* between variants of program clauses is needed in GAP resolution but not in $SLD(\mathcal{D})$. The purpose of reductants in GAP is to explicitly compute the *lub*s of several lattice values (qualification values in the case of $QLP(\mathcal{D})$) which would result from finitely many different computations if no reductants were used. In GAP's declarative semantics, interpretations are required to be closed w.r.t. finite *lub*s of lattice values assigned to the same atom, and for this reason reductants are needed for the completeness of goal resolution. In $QLP(\mathcal{D})$ interpretations as defined in Section 3 no closure condition w.r.t. *lub*s is required, and therefore the completeness result stated in Theorem 2 can be proved without reductants. Of course, the $QLP(\mathcal{D})$ approach to semantics means that a user has to observe *several* computed answers for one and the same goal and think of the *lub* of the various $\mathcal{D}$ elements provided by the different computations by himself instead of getting the *lub* computed by one single $SLD(\mathcal{D})$ derivation. In our opinion, this is a reasonable scenario because even in GAP the $\mathcal{T}$ value provided by any single computed answer always corresponds to some *lub* of finitely many $\mathcal{T}$ values, and it may be not the highest possible $\mathcal{T}$ value w.r.t. to the program's declarative semantics. Moreover, our Theorem 2 is much stronger than the one given in [11], which only ensures the possibility of computing *some* solution for any goal whose solvability holds in the least program model. We strongly conjecture that a stronger completeness theorem could be proved also for GAP by using a proof technique more similar to our's.

As possible lines of future work we consider: to improve the current prototype implementations of the instances $QLP(\mathcal{U})$, $QLP(\mathcal{W})$ and $QLP(\mathcal{U} \times \mathcal{W})$; to extend the $QLP(\mathcal{D})$ scheme and its implementation to a more expressive scheme which can support qualitative programming with features such as disjunctive

goals, negation, lazy functions and parametrically given constraint domains; to explore alternative semantic approaches, considering annotations, bilattices, probabilistic semantics and similarity based unification; and to investigate applications to the computation of qualified answers for web search queries.

Acknowledgements

The authors are thankful to their colleagues Paco López and Rafa Caballero for their valuable hints concerning bibliography and implementation techniques. They also appreciate the constructive comments of the anonymous reviewers, that were helpful for improving the presentation.

References

1. K. 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.
2. 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.
3. 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 2.3.1, 2007. R. Caballero and J. Sánchez (Eds.), Available at `http://toy.sourceforge.net`.
4. K. L. Clark. Predicate logic as a computational formalism (res. report doc 79/59). Technical report, Imperial College, Dept. of Computing, London, 1979.
5. A. Dekhtyar and V. S. Subrahmanian. Hybrid probabilistic programs. *Journal of Logic Programming*, 43(3):187–250, 2000.
6. J. Dix, S. Kraus, and V. S. Subrahmanian. Heterogeneous temporal probabilistic agents. *ACM Transactions on Computational Logic*, 7(1):151–198, 2006.
7. J. Dix, M. Nanni, and V. S. Subrahmanian. Probabilistic agent programs. *ACM Transactions on Computational Logic*, 1(2):208–246, 2000.
8. M. Fitting. Bilattices and the semantics of logic programming. *Journal of Logic Programming*, 11:91–116, 1991.
9. 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.
10. 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.
11. 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.
12. 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.
13. G. Moreno and V. Pascual. Programming with fuzzy logic and mathematical functions. In A. P. I. Bloch and A. Tettamanzi, editors, *Proceedings of the 6th International Workshop on Fuzzy Logic and Applications (WILF'05)*, volume 3849 of *LNAI*, pages 89–98. Springer Verlag, 2006.

14. R. T. Ng and V. S. Subrahmanian. Probabilistic logic programming. *Information and Computation*, 101(2):150–201, 1992.
15. R. 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.
16. S. Riezler. *Probabilistic Constraint Logic Programming*. PhD thesis, Neuphilologischen Fakultät del Universität Tübingen, 1998.
17. C. A. Romero-Díaz. Programación lógica cuantitativa y su implementación en $\mathcal{TOY}$. Proyecto de Fin de Máster, DSIC, Universidad Complutense de Madrid.
18. M. I. Sessa. Approximate reasoning by similarity-based SLD resolution. *Theoretical Computer Science*, 275(1-2):389–426, 2002.
19. E. Y. Shapiro. Logic programs with uncertainties: A tool for implementing rule-based systems. In A. Bundy, editor, *Proceedings of the 8th International Joint Conference on Artificial Intelligence (IJCAI'83)*, pages 529–532, Karlsruhe, Germany, 1983.
20. R. F. Stärk. A direct proof for the completeness of SLD-resolution. In E. Börger, H. K. Büning, and M. M. Richter, editors, *Proceedings of the 3rd Workshop on Computer Science Logic (CSL'89)*, volume 440 of *LNCS*, pages 382–383. Springer Verlag, 1990.
21. 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.
22. 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.
23. V. S. Subrahmanian. Uncertainty in logic programming: Some recollections. *Association for Logic Programming Newsletter*, 20(2), 2007.
24. M. H. van Emden. Quantitative deduction and its fixpoint theory. *Journal of Logic Programming*, 3(1):37–53, 1986.
25. 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.

A Appendix: Additional Proofs

This appendix contains the proofs of the Soundness Theorem 1 and the Completeness Theorem 2. In order to prove them, we will previously prove an auxiliary lemma for each of the two cases. In the rest of the Appendix, we assume a given program $\mathcal{P}$ over a qualification domain $\mathcal{D}$.

A.1 Proof of the Soundness Theorem

Lemma 1 (Soundness). *Assume two goals G_0 and G_1 and a pair of substitutions (θ, ρ) such that $G_0 \Vdash_{C_1,\sigma_1} G_1$ and $(\theta, \rho) \in QSol_{\mathcal{P}}(G_1)$ (the set of all solutions of G_1). Then we have that $(\theta, \rho) \in QSol_{\mathcal{P}}(G_0)$.*

Proof. Assume $G_0 \equiv \overline{L}, A \sharp W, \overline{R} \parallel \sigma_0 \parallel \alpha \circ W \sqsupseteq \beta, \Delta$; $C_1 \equiv (H \leftarrow d - B_1, \ldots, B_k) \in_{\text{var}} \mathcal{P}$ a variant of a program clause without variables in common with G_0; and σ_1 the m.g.u. between A and H. Then

$$G_1 \equiv (\overline{L}, B_1 \sharp W_1, \ldots, B_k \sharp W_k \overline{R})\sigma_1 \parallel \sigma_0\sigma_1 \parallel \Delta_1$$

where $\Delta_1 \equiv W = d \circ \bigsqcap\{W_1, \ldots, W_k\}, d \circ \alpha \circ W_1 \sqsupseteq \beta, \ldots, d \circ \alpha \circ W_k \sqsupseteq \beta, \Delta$. As $(\theta, \rho) \in QSol_{\mathcal{P}}(G_1)$ we know

(1) $\sigma_0\sigma_1\theta = \theta$,
(2) $\rho \in Sol(\Delta_1)$, and
(3) $\mathcal{P} \vdash_{\text{QHL}(\mathcal{D})} (\overline{L}, B_1 \sharp W_1, \ldots, B_k \sharp W_k, \overline{R})\sigma_1\hat{}(\theta, \rho)$[1] .

And for (θ, ρ) to be a solution of G_0 we need the following:

(4) $\sigma_0\theta = \theta$,
(5) $\rho \in Sol(\alpha \circ W \sqsupseteq \beta, \Delta)$, and
(6) $\mathcal{P} \vdash_{\text{QHL}(\mathcal{D})} (\overline{L}, A \sharp W, \overline{R})\hat{}(\theta, \rho)$.

Therefore we have to prove (4), (5) and (6).

Proof of (4). First, we can see that for every variable $y \in \mathcal{V}ar$, if $y \in \text{vran}(\sigma_0)$, then $y \notin \text{dom}(\sigma_0)$ because σ_0 is idempotent. Hence, $y \in \text{vran}(\sigma_0) \Longrightarrow y\sigma_1\theta = y\sigma_0\sigma_1\theta =_{(1)} y\theta$, and it is true that (7) $\sigma_1\theta = \theta$ [vran(σ_0)]. Now, for any variable x we can prove $x\sigma_0\theta = x\theta$ by distinguishing two cases: a) if $x \notin \text{dom}(\sigma_0)$ then $x\sigma_0\theta = x\theta$; and b) if $x \in \text{dom}(\sigma_0)$ then $\text{var}(x\sigma_0) \subseteq \text{vran}(\sigma_0) \Longrightarrow x\sigma_0\theta =_{(7)} x\sigma_0\sigma_1\theta =_{(1)} x\theta$.

Proof of (5). We have to prove that $\alpha \circ W\rho \sqsupseteq \beta$ and $\rho \in Sol(\Delta)$. $\alpha \circ W\rho =_{(2)} \alpha \circ d \circ \bigsqcap\{W_1\rho, \ldots, W_k\rho\} = \bigsqcap\{\alpha \circ d \circ W_1\rho, \ldots, \alpha \circ d \circ W_k\rho\}$. It is enough proving $\alpha \circ d \circ W_i\rho \sqsupseteq \beta$ for $1 \leq i \leq k$. But $\{\alpha \circ d \circ W_i \sqsupseteq \beta \mid 1 \leq i \leq k\} \subseteq \Delta_1$ and $\rho \in Sol(\Delta_1)$. $\rho \in Sol(\Delta)$ is trivial because $\Delta \subseteq \Delta_1$.

[1] $\mathcal{P} \vdash_{\text{QHL}(\mathcal{D})} (A \sharp W)\hat{}(\theta, \rho) \Longleftrightarrow_{\text{def}} \mathcal{P} \vdash_{\text{QHL}(\mathcal{D})} A\theta \sharp W\rho$.

Proof of (6). We can split (6) in the following three cases:

(6a) $\mathcal{P} \vdash_{\mathrm{QHL}(\mathcal{D})} \overline{L}\hat{}(\theta,\rho)$. We prove that $\overline{L}\hat{}(\theta,\rho) = \overline{L}\sigma_1\hat{}(\theta,\rho)$ which, because of (3), can be inferred in $QHL(\mathcal{D})$ from $\mathcal{P}$. We know that $\mathrm{dom}(\sigma_0) \cap \mathrm{var}(G_0) = \emptyset$, therefore, $\overline{L}\sigma_1\hat{}(\theta,\rho) = \overline{L}\sigma_0\sigma_1\hat{}(\theta,\rho) = \overline{L}\hat{}(\sigma_0\sigma_1\theta,\rho) =_{(1)} \overline{L}\hat{}(\theta,\rho)$.

(6b) $\mathcal{P} \vdash_{\mathrm{QHL}(\mathcal{D})} A \sharp W\hat{}(\theta,\rho)$. Using $W\rho = d \circ \bigsqcap\{W_1\rho,\ldots,W_k\rho\}$ which holds because of (2), (3) and one inference step with clause C_1 and substitution $\sigma_1\theta$, we obtain $\mathcal{P} \vdash_{\mathrm{QHL}(\mathcal{D})} H\sigma_1\theta \sharp W\rho$. Now, because σ_1 is the m.g.u. between A and H, we have $H\sigma_1\theta = A\sigma_1\theta$. Therefore, we have $\mathcal{P} \vdash_{\mathrm{QHL}(\mathcal{D})} A\sigma_1\theta \sharp W\rho$. Finally, we note that $A\sigma_1\theta = A\sigma_0\sigma_1\theta =_{(1)} A\theta$ because $\mathrm{dom}(\sigma_0) \cap \mathrm{var}(A) = \emptyset$.

(6c) $\mathcal{P} \vdash_{\mathrm{QHL}(\mathcal{D})} \overline{R}\hat{}(\theta,\rho)$. As in (6a). □

Proof (of Soundness Theorem). Assume $G_0 \Vdash^n_{\sigma'} G$ where $G \equiv \sigma \,[\!]\, \Delta$ is solved. Let (σ,μ) be the solution associated to G. We prove $(\sigma,\mu) \in QSol_{\mathcal{P}}(G_0)$ by induction on n.

Base. In this case, $n = 0$ and $G_0 = G$ is solved and $\mathcal{P} \vdash_{\mathrm{QHL}(\mathcal{D})} \overline{A}\hat{}(\sigma,\mu)$ is trivial because the sequence of atoms $\overline{A}$ of G is empty. Moreover, $\mu \in Sol(\Delta)$ because $\mu = \omega_\Delta$.

Induction. In this case we have $n > 0$ and $G_0 \Vdash G_1 \Vdash^{n-1} G$. Then we obtain $(\sigma,\mu) \in QSol_{\mathcal{P}}(G_1)$ by induction hypothesis, and therefore $(\sigma,\mu) \in QSol_{\mathcal{P}}(G_0)$ because of Lemma 1. □

A.2 Proof of the Completeness Theorem

Before going into the proof, just a note on notation: as said, $(\theta,\rho) \in QSol_{\mathcal{P}}(G)$ means that the pair of substitutions (θ,ρ) is a solution of the goal $G \equiv \overline{A} \,[\!]\, \sigma \,[\!]\, \Delta$. Now, writing $(\theta,\rho) \in QSol^n_{\mathcal{P}}(G)$ we are expressing that the exact number of inference steps in $\mathcal{P} \vdash_{\mathrm{QHL}(\mathcal{D})} \overline{A}\hat{}(\theta,\rho)$ is n, written as $\mathcal{P} \vdash^n_{\mathrm{QHL}(\mathcal{D})} \overline{A}\hat{}(\theta,\rho)$.

Lemma 2 (Completeness). *Let $G_0 \equiv \overline{A_0} \,[\!]\, \sigma_0 \,[\!]\, \Delta_0$ be a goal not solved, and $(\theta_0,\rho_0) \in QSol^n_{\mathcal{P}}(G_0)$. Let also V_0 be any finite set of variables such that $\mathrm{var}(G_0) \cup \mathrm{dom}(\theta_0) \subseteq V_0$. For any arbitrary selection of an atom $A \sharp W$ of $\overline{A_0}$, there exists some resolution step $G_0 \Vdash_{\sigma_1} G_1$ selecting the chosen atom and, in addition, some (θ_1,ρ_1) satisfying the following properties:*

a. $\theta_1 = \theta_0\,[V_0]$
b. $\sigma_1\theta_1 = \theta_1$
c. $\sigma_0\sigma_1\theta_1 = \theta_1$
d. $\rho_1 \sqsupseteq \rho_0\,[\mathrm{war}(G_0)]$
e. $\rho_1 \in Sol(\Delta_1)$
f. $\mathcal{P} \vdash^{n-1}_{\mathrm{QHL}(\mathcal{D})} \overline{A_1}\hat{}(\theta_1,\rho_1)$

In particular, (c), (e) and (f) mean that $(\theta_1,\rho_1) \in QSol^{n-1}_{\mathcal{P}}(G_1)$.

Proof. Assume $A \sharp W$ to be the selected atom in G_0. Then, $G_0 \equiv \overline{L_0}, A \sharp W, \overline{R_0} \,[\!]\, \sigma_0 \,[\!]\, \alpha \circ W \sqsupseteq \beta, \Delta$. Because of the lemma's hypothesis we can also assume the following:

(0) $\sigma_0\theta_0 = \theta_0$,
(1) $\rho_0 \in Sol(\Delta_0)$,
(2) $\mathcal{P} \vdash^{m_1}_{\mathrm{QHL}(\mathcal{D})} \widehat{\overline{L_0}}(\theta_0, \rho_0)$,
(3) $\mathcal{P} \vdash^{m_2}_{\mathrm{QHL}(\mathcal{D})} A \sharp \widehat{W}(\theta_0, \rho_0)$, and
(4) $\mathcal{P} \vdash^{m_3}_{\mathrm{QHL}(\mathcal{D})} \widehat{\overline{R_0}}(\theta_0, \rho_0)$

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

Because of (3) there must exist some clause $C_1 \equiv (H \leftarrow d - B_1, \ldots, B_k) \in_{\mathrm{var}} \mathcal{P}$ and some substitution η_0 such that

(5) $A\theta_0 = H\eta_0$ and $\mathcal{P} \vdash^{m_2-1}_{\mathrm{QHL}(\mathcal{D})} B_1\eta_0 \sharp d_1, \ldots, B_k\eta_0 \sharp d_k$ with $d_1, \ldots, d_k \in D \backslash \{\bot\}$ such that $W\rho_0 \sqsubseteq d \circ \bigsqcap\{d_1, \ldots, d_k\}$.

It is possible to choose C_1 and η_0 so that $\mathrm{var}(C_1) \cap V_0 = \emptyset$ and $\mathrm{dom}(\eta_0) \subseteq \mathrm{var}(C_1)$. Therefore, it is guaranteed that $\mathrm{dom}(\eta_0) \cap \mathrm{dom}(\theta_0) = \emptyset$ and then:

(6) $\theta_1 =_{\mathrm{def}} \theta_0 \uplus \eta_0$ is a well-founded substitution that satisfies: $\mathrm{dom}(\theta_1) = \mathrm{dom}(\theta_0) \uplus \mathrm{dom}(\eta_0)$; $\theta_1 = \theta_0\,[V_0]$; $\theta_1 = \eta_0\,[\backslash V_0] \Longrightarrow (a)$ of lemma.

From (5) and (6) we know that θ_1 is an unifier of A and H. Choosing σ_1 as the m.g.u. (in the Robinson's sense) between A and H we will have:

(7) $A\sigma_1 = H\sigma_1$ and $\sigma_1\theta_1 = \theta_1 \Longrightarrow (b)$ of lemma.

Then, taking ρ_1 such that

$$(8)\ W'\rho_1 =_{\mathrm{def}} \begin{cases} d_i & \text{if } W' = W_i \text{ for some } 1 \le i \le k \\ d \circ \bigsqcap\{d_1, \ldots, d_k\} & \text{if } W' = W \\ W'\rho_0 & \text{otherwise} \end{cases}$$

we will have, by (8) and (5), $\rho_1 \sqsupseteq \rho_0\,[\mathrm{war}(G_0)] \Longrightarrow (d)$ of lemma.

Now, doing a resolution step with σ_1 and C_1 we get $G_0 \Vdash_{\sigma_1, C_1} G_1 \equiv \overline{A_1} \,[\!]\, \sigma_0\sigma_1 \,[\!]\, \Delta_1$ where $\overline{A_1} = (\overline{L_0}, B_1 \sharp W_1, \ldots, B_k \sharp W_k, \overline{R_0})\sigma_1$ and $\Delta_1 \equiv d \circ \alpha \circ W_1 \sqsupseteq \beta, \ldots, d \circ \alpha \circ W_k \sqsupseteq \beta, W = d \circ \bigsqcap\{W_1, \ldots, W_k\}, \Delta$. Note that we can deduce $d \circ \alpha \sqsupseteq \beta$ from (5), (1) and the axioms required for the attenuation operation ($\circ$) in any qualification domain, because we have $W\rho_0 = d \circ \bigsqcap\{d_1, \ldots, d_k\}$ and $\alpha \circ W\rho_0 \sqsupseteq \beta \Longrightarrow \alpha \circ d \circ \bigsqcap\{d_1, \ldots, d_k\} \sqsupseteq \beta \Longrightarrow \alpha \circ d \sqsupseteq \beta$. Remember from Definition 3 that the condition $\alpha \circ d \sqsupseteq \beta$ is required for the resolution step to be enabled.

To finish the proof we only need to prove (c), (e) and (f).

Proof of (c). $\sigma_0\underline{\sigma_1\theta_1} =_{(b)} \sigma_0\theta_1 =_{(6)} \sigma_0(\theta_0 \uplus \eta_0) =_{(*)} \sigma_0\theta_0 \uplus \eta_0 =_{(0)} \theta_0 \uplus \eta_0 = \theta_1$.
$(*)$ Because $\mathrm{vran}(\sigma_0) \subseteq V_0$ and $\mathrm{dom}(\eta_0) \cap V_0 = \emptyset$.

Proof of (e). We have to see that ρ_1 satisfies every constraint in Δ_1:

1. $W = d \circ \bigsqcap\{W_1, \ldots, W_k\}$. This is satisfied by definition of ρ_1.

2. $d \circ \alpha \circ W_i \sqsupseteq \beta$ for $1 \leq i \leq k$. We know from (1) that $\alpha \circ W\rho_0 \sqsupseteq \beta$, and from (d) follows $W\rho_1 \sqsupseteq W\rho_0$. Therefore, $\alpha \circ W\rho_1 \sqsupseteq \beta$. Because of (8) we also know $W\rho_1 = d \circ \bigsqcap\{W_1\rho_1, \ldots, W_k\rho_1\}$ that implies $\alpha \circ W\rho_1 = \bigsqcap\{d \circ \alpha \circ W_1\rho_1, \ldots, d \circ \alpha \circ W_k\rho_1\}$ Hence $\alpha \circ W\rho_1 \sqsupseteq \beta$ implies that $d \circ \alpha \circ W_i\rho_1 \sqsupseteq \beta$ is satisfied for every $1 \leq i \leq k$.
3. Δ. From (7) follows that $\rho_1 = \rho_0\,[\mathrm{war}(\Delta)]$, and because of (1) $\rho_0 \in Sol(\Delta) \Longrightarrow \rho_1 \in Sol(\Delta)$.

Proof of (f). First we can deduce:

(9) $\mathcal{P} \vdash^{m_1}_{\mathrm{QHL}(\mathcal{D})} \overline{L_0}\hat{}(\theta_1, \rho_1)$ by (2), (6) and (8).
(10) $\mathcal{P} \vdash^{m_2-1}_{\mathrm{QHL}(\mathcal{D})} (B_1 \sharp W_1, \ldots, B_k \sharp W_k)\hat{}(\theta_1, \rho_1)$ by (5), (6) and (8).
(11) $\mathcal{P} \vdash^{m_3}_{\mathrm{QHL}(\mathcal{D})} \overline{R_0}\hat{}(\theta_1, \rho_1)$ by (4), (6) and (8).

Considering that $m_1 + (m_2 - 1) + m_3 = n - 1$, (9), (10) and (11) imply that $\mathcal{P} \vdash^{n-1}_{\mathrm{QHL}(\mathcal{D})} (\overline{L_0}, B_1 \sharp W_1, \ldots, B_k \sharp W_k, \overline{R_0})\hat{}(\theta_1, \rho_1)$; and this is (f) due to (b). □

Proof (of Completeness Theorem). As G_0 is a goal, we know that σ_0 is idempotent and that $\overline{A_0\sigma_0} = \overline{A_0}$. Now, as $(\theta_0, \rho_0) \in QSol_{\mathcal{P}}(G_0)$, we can choose a number $n \in \mathbb{N}$ such that $(\theta_0, \rho_0) \in QSol^n_{\mathcal{P}}(G_0)$ and therefore we have (1) $\sigma_0\theta_0 = \theta_0$, (2) $\rho_0 \in Sol(\Delta_0)$ and (3) $\mathcal{P} \vdash^n_{\mathrm{QHL}(\mathcal{D})} \overline{A_0}\hat{}(\theta_0, \rho_0)$. We can also choose a finite set of variables V_0 satisfying (4) $\mathrm{var}(G_0) \cup \mathrm{dom}(\theta_0) \subseteq V_0$. Given the conditions (1) to (4), we will prove the following:

(†) There exist some resolution computation $G_0 \Vdash^*_\sigma \sigma_0\sigma \,[]\, \Delta$ (that we can build making use of any selection strategy) ending in a solved goal, and some substitution θ satisfying (5) $\theta = \theta_0\,[V_0]$, (6) $\sigma\theta = \theta$ and (7) $\sigma_0\sigma\theta = \theta$.

From (†) follows the theorem's thesis (except (8) $\mu \sqsupseteq \rho_0\,[\mathrm{war}(G_0)]$) because:

– (6) $\Longrightarrow_{(5)} \sigma\theta = \theta_0\,[V_0] \Longrightarrow \sigma \preccurlyeq \theta_0\,[\mathrm{var}(G_0)]$
– (7) $\Longrightarrow_{(5)} \sigma_0\sigma\theta = \theta_0\,[V_0] \Longrightarrow \sigma_0\sigma \preccurlyeq \theta_0\,[\mathrm{var}(G_0)]$

We simultaneously prove (†) and (8) by induction on n:

Base. If $n = 0$, (2) implies that $\overline{A_0}$ is empty. Then, taking $\sigma = \epsilon$ and $\theta = \theta_0$ we have that $G_0 \Vdash^0_\epsilon \sigma_0 \,[]\, \Delta_0$ with is a trivial resolution of 0 steps; and in addition:
  – (5) reduces to $\theta_0 = \theta_0\,[V_0]$, which is trivial.
  – (6) reduces to $\theta_0 = \theta_0$, which also is trivial.
  – (7) reduces to $\sigma_0\theta_0 = \theta_0$ which is true given (1).
  – (8) is satisfied because $\mu = \rho_0\,[\mathrm{war}(G_0)]$ is true, given that $\mathrm{war}(G_0) = \mathrm{war}(\Delta_0)$. Now, as Δ_0 is solved, it only contains defining constraints and therefore (2) implies that for any $W \in \mathrm{war}(\Delta_0)$ it is true that $W\rho_0 = \omega_{\Delta_0}(W) = W\mu$.

Induction. If $n > 0$, (2) implies that $\overline{A_0}$ is not empty. Hence, selecting an atom $A \sharp W$ in $\overline{A_0}$ with any selection strategy and using the Completeness Lemma 2, we can perform a resolution step

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

having that there exists some solution $(\theta_1, \rho_1) \in QSol_{\mathcal{P}}^{n-1}(G_1)$ satisfying all 6 conditions guaranteed by the lemma: (10) $\theta_1 = \theta_0\,[V_0]$, (11) $\sigma_1\theta_1 = \theta_1$, (1') $\sigma_0\sigma_1\theta_1 = \theta_1$, (12) $\rho_1 \sqsupseteq \rho_0\,[\mathrm{war}(G_0)]$, (2') $\rho_1 \in Sol(\Delta_1)$, and (3') $\mathcal{P} \vdash^{n-1}_{\mathrm{QHL}(\mathcal{D})} \overline{A_1}\hat{}(\theta_1, \rho_1)$.
Let V_1 be any finite set of variables such that

$$(4') \quad V_0 \cup \mathrm{var}(G_1) \cup \mathrm{dom}(\theta_1) \subseteq V_1 \ .$$

Conditions (1'), (2'), (3') and (4') are similar, respectively, to (1), (2), (3) and (4), but now for $(\theta_1, \rho_1) \in QSol_{\mathcal{P}}^{n-1}(G_1)$. By induction hypothesis we can obtain a resolution computation

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

and a substitution θ such that
(5') $\theta = \theta_1\,[V_1]$ (6') $\sigma'\theta = \theta$ (7') $\sigma_0\sigma_1\sigma'\theta = \theta$
(8') $\mu' \sqsupseteq \rho_1[\mathrm{war}(G_1)]$ with $(\sigma_0\sigma_1\sigma', \mu')$ the associated solution to $\sigma_0\sigma_1\sigma' \,[\!]\, \Delta'$.
From (9) and (13) results

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

Now there is only left to check that (5), (6) and (7) are satisfied given the same θ that satisfies (5'), (6') and (7') and $\sigma = \sigma_1\sigma'$; and that (8) is also satisfied when $\mu = \mu'$. In fact:
- (5) trivially follows from (5'), (4') and (10).
- (6) comes from the following: by (4') we can assume $\theta = \theta_1 \uplus \eta'$, with η' such that $\mathrm{dom}(\eta') \cap (V_1) = \emptyset$. Then: $\underline{\sigma}\theta = \sigma_1\underline{\sigma'\theta} =_{(6')} \sigma_1\underline{\theta} = \sigma_1(\theta_1 \uplus \eta') =_{(*)} \underline{\sigma_1\theta_1} \uplus \eta' =_{(11)} \theta_1 \uplus \eta' = \theta$. The step $(*)$ is correct because $\mathrm{vran}(\sigma_1) \subseteq \overline{V_1}$ and $\mathrm{dom}(\eta') \cap V_1 = \emptyset$.
- (7) trivially follows from (7'), given that $\sigma = \sigma_1\sigma'$.
- (8) is consequence of (8') and (12), because $\mathrm{war}(G_0) \subseteq \mathrm{war}(G_1)$. □

Similarity-based Reasoning in Qualified Logic Programming

Revised Edition

Rafael Caballero Mario Rodríguez-Artalejo Carlos A. Romero-Díaz

Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid, Spain
{rafa,mario}@sip.ucm.es, cromdia@fdi.ucm.es

Abstract

Similarity-based Logic Programming (briefly, SLP) has been proposed to enhance the LP paradigm with a kind of approximate reasoning which supports flexible information retrieval applications. This approach uses a fuzzy similarity relation $\mathcal{R}$ between symbols in the program's signature, while keeping the syntax for program clauses as in classical LP. Another recent proposal is the $QLP(\mathcal{D})$ scheme for *Qualified Logic Programming*, an extension of the LP paradigm which supports approximate reasoning and more. This approach uses annotated program clauses and a parametrically given domain $\mathcal{D}$ whose elements qualify logical assertions by measuring their closeness to various users' expectations. In this paper we propose a more expressive scheme $SQLP(\mathcal{R}, \mathcal{D})$ which subsumes both SLP and $QLP(\mathcal{D})$ as particular cases. We also show that $SQLP(\mathcal{R}, \mathcal{D})$ programs can be transformed into semantically equivalent $QLP(\mathcal{D})$ programs. As a consequence, existing $QLP(\mathcal{D})$ implementations can be used to give efficient support for similarity-based reasoning.

Categories and Subject Descriptors D.1.6 [*Programming Techniques*]: Logic Programming; D.3.2 [*Programming Languages*]: Language Classifications—Constraint and logic languages; F.3.2 [*Theory of Computation*]: Logics and Meanings of Programs—Algebraic approaches to semantics

General Terms Algorithms, Languages, Theory

Keywords Qualification Domains, Similarity Relations

1. Introduction

The historical evolution of the research on uncertainty in *Logic Programming* (LP) has been described in a recent recollection by V. S. Subrahmanian [19]. Early approaches include the quantitative treatment of uncertainty in the spirit of fuzzy logic, as in van Emden's classical paper [20] and two subsequent papers by Subrahmanian [17, 18]. The main contribution of [20] was a rigorous declarative semantics for a LP language with program clauses of the form $A \leftarrow d - \overline{B}$, where the head A is an atom, the body $\overline{B}$ is a conjunction of atoms, and the so-called *attenuation* factor $d \in (0, 1]$ attached to the clause's implication is used to propagate to the head the certainty factor $d \times b$, where b is the minimum of the certainty factors $d_i \in (0, 1]$ previously computed for the various atoms occurring in the body. The papers [17, 18] proposed to use a special lattice $\mathcal{T}$ in place of the lattice of the real numbers in the interval $[0, 1]$ under their natural ordering. $\mathcal{T}$ includes two isomorphic copies of $[0, 1]$ whose elements are incomparable under $\mathcal{T}$'s ordering and can be used separately to represent degrees of *truth* and *falsity*, respectively, thus enabling a simple treatment of negation. Other main contributions of [17, 18] were the introduction of annotated program clauses and goals (later generalized to a much more expressive framework in [7]), as well as goal solving procedures more convenient and powerful than those given in [20].

A more recent line of research is *Similarity-based Logic Programming* (briefly, SLP) as presented in [16] and previous related works such as [3, 6, 5, 15]. This approach also uses the lattice $[0, 1]$ to deal with uncertainty in the spirit of fuzzy logic. In contrast to approaches based on annotated clauses, programs in SLP are just sets of definite Horn clauses as in classical LP. However, a *similarity relation* $\mathcal{R}$ (roughly, the fuzzy analog of an equivalence relation) between predicate and function symbols is used to enable the unification terms that would be not unifiable in the classical sense, measured by some degree $\lambda \in (0, 1]$. There are different proposals for the operational semantics of SLP programs. One possibility is to apply classical SLD resolution w.r.t. a transformation of the original program [6, 15, 16]. Alternatively, a $\mathcal{R}$-based SLD-resolution procedure relying on $\mathcal{R}$-unification can be applied w.r.t. to the original program, as proposed in [16]. Propositions 7.1 and 7.2 in [16] state a correspondence between the answers computed by $\mathcal{R}$-based SLD resolution w.r.t. a given logic program $\mathcal{P}$ and the answers computed by classical SLD resolution w.r.t. the two transformed programs $H_\lambda(\mathcal{P})$ (built by adding to $\mathcal{P}$ new clauses $\mathcal{R}$-similar to those in $\mathcal{P}$ up to the degree $\lambda \in (0, 1]$) and $\mathcal{P}_\lambda$ (built by replacing all the function and predicate symbols in $\mathcal{P}$ by new symbols that represent equivalence classes modulo $\mathcal{R}$-similarity up to λ). The $SiLog$ system [8] has been developed to implement SLP and to support applications related to flexible information retrieval from the web.

The aim of the present paper is to show that similarity-based reasoning can be expressed in $QLP(\mathcal{D})$, a programming scheme for *Qualified* LP over a parametrically given *Qualification Domain* $\mathcal{D}$ recently presented in [14] as a generalization and improvement of the classical approach by van Emden [20] to $Quantitative$ LP. Qualification domains are lattices satisfying certain natural axioms. They include the lattice $[0, 1]$ used both in [20] and in [16], as well as other lattices whose elements can be used to qualify logical assertions by measuring their closeness to different kinds of users' expectations. Programs in $QLP(\mathcal{D})$ use $\mathcal{D}$-attenuated clauses of the form $A \leftarrow d - \overline{B}$ where A is an atom, $\overline{B}$ a finite conjunction of atoms and $d \in D \setminus \{\bot\}$ is the *attenuation value*

PPDP'08, July 15–17, 2008, Valencia, Spain.

attached to the clause's implication, used to propagate to the head the *qualification value* $d \circ b$, where b is the infimum in $\mathcal{D}$ of the qualification values $d_i \in D \setminus \{\bot\}$ previously computed for the various atoms occurring in the body, and $\circ$ is an *attenuation operator* coming with $\mathcal{D}$. As reported in [14, 13], the classical results in LP concerning the existence of least Herbrand models of programs and the soundness and completeness of the SLD resolution procedure (see e.g.[21, 2, 1]) have been extended to the $QLP(\mathcal{D})$ scheme, and potentially useful instances of the scheme have been implemented on top of the *Constraint Functional Logic Programming* ($CFLP$) system $\mathcal{TOY}$ [4].

The results presented in this paper can be summarized as follows: we consider generalized similarity relations over a set S as mappings $\mathcal{R} : S \times S \to D$ taking values in the carrier set D of an arbitrarily given qualification domain $\mathcal{D}$, and we extend $QLP(\mathcal{D})$ to a more expressive scheme $SQLP(\mathcal{R}, \mathcal{D})$ with two parameters for programming modulo $\mathcal{R}$-similarity with $\mathcal{D}$-attenuated Horn clauses. We present a declarative semantics for $SQLP(\mathcal{R}, \mathcal{D})$ and a program transformation mapping each $SQLP(\mathcal{R}, \mathcal{D})$ program $\mathcal{P}$ into a $QLP(\mathcal{D})$ program $S_{\mathcal{R}}(\mathcal{P})$ whose least Herbrand model corresponds to that of $\mathcal{P}$. Roughly, $S_{\mathcal{R}}(\mathcal{P})$ is built adding to $\mathcal{P}$ new clauses obtained from the original clauses in $\mathcal{P}$ by computing various new heads $\mathcal{R}$-similar to a linearized version of the original head, adding also $\mathcal{R}$-similarity conditions $X_i \sim X_j$ to the body and suitable clauses for the new predicate $\sim$ to emulate $\mathcal{R}$-based unification. Thanks to the $S_{\mathcal{R}}(\mathcal{P})$ transformation, the sound and complete procedure for solving goals in $QLP(\mathcal{D})$ by $\mathcal{D}$-qualified SLD resolution and its implementation in the $\mathcal{TOY}$ system [14] can be used to implement $SQLP(\mathcal{R}, \mathcal{D})$ computations, including as a particular case SLP computations in the sense of [16].

Another recent proposal for reducing the SLP approach in [16] to a fuzzy LP paradigm can be found in [11], a paper which relies on the multi-adjoint framework for Logic Programming (*MALP* for short) previously proposed in [9, 10]. *MALP* is a quite general framework supporting LP with *weighted program rules* over different multi-adjoint lattices, each of which provides a particular choice of operators for implication, conjunction and aggregation of atoms in rule bodies. In comparison to the $QLP(\mathcal{D})$ scheme, the multi-adjoint framework differs in motivation and scope. Multi-adjoint lattices and qualification domains are two different classes of algebraic structures. Concerning declarative and operational semantics, there are also some significant differences between $QLP(\mathcal{D})$ and *MALP*. In particular, *MALP* 's goal solving procedure relies on a costly computation of *reductant clauses*, a technique borrowed from [7] which can be avoided in $QLP(\mathcal{D})$, as discussed in the concluding section of [14].

In spite of these differences, the results in [11] concerning the emulation of similarity-based can be compared to those in the present paper. Theorem 24 in [11] shows that every classical logic program $\mathcal{P}$ can be transformed into a *MALP* program $\mathcal{P}_{E,\mathcal{R}}$ which can be executed using only syntactical unification and emulates the successful computations of $\mathcal{P}$ using the SLD resolution with $\mathcal{R}$-based unification introduced in [16]. $\mathcal{P}_{E,\mathcal{R}}$ works over a particular multi-adjoint lattice $\mathcal{G}$ with carrier set $[0, 1]$ and implication and conjunction operators chosen according to the so-called Gödel's semantics [22]. $\mathcal{P}_{E,\mathcal{R}}$ also introduces clauses for a binary predicate $\sim$ which emulates $\mathcal{R}$-based unification, as in our transformation $S_{\mathcal{R}}(\mathcal{P})$. Nevertheless, $S_{\mathcal{R}}(\mathcal{P})$ is defined for a more general class of programs and uses the $\mathcal{R}$-similarity predicate $\sim$ only if the source program $\mathcal{P}$ has some clause whose head is non-linear. More detailed comparisons between the program transformations $S_{\mathcal{R}}(\mathcal{P})$, $H_\lambda(\mathcal{P})$, $\mathcal{P}_\lambda$ and $\mathcal{P}_{E,\mathcal{R}}$ will be given in Subsection 4.2.

The rest of the paper is structured as follows: In Section 2 we recall the qualification domains $\mathcal{D}$ first introduced in [14] and we define similarity relations $\mathcal{R}$ over an arbitrary qualification domain. In Section 3 we recall the scheme $QLP(\mathcal{D})$ and we introduce its extension $SQLP(\mathcal{R}, \mathcal{D})$ with its declarative semantics, given by a logical calculus which characterizes the least Herbrand model $\mathcal{M}_{\mathcal{P}}$ of each $SQLP(\mathcal{R}, \mathcal{D})$ program $\mathcal{P}$. In Section 4 we define the transformation $S_{\mathcal{R}}(\mathcal{P})$ of any given $SQLP(\mathcal{R}, \mathcal{D})$ program $\mathcal{P}$ into a $QLP(\mathcal{D})$ program $S_{\mathcal{R}}(\mathcal{P})$ such that $\mathcal{M}_{S_{\mathcal{R}}(\mathcal{P})} = \mathcal{M}_{\mathcal{P}}$, we give some comparisons to previously known program transformations, and we illustrate the application of $S_{\mathcal{R}}(\mathcal{P})$ to similarity-based computation by means of a simple example. Finally, in Section 5 we summarize conclusions and comparisons to related work and we point to planned lines of future work.

2. Qualification Domains and Similarity Relations

2.1 Qualification Domains

Qualification Domains were introduced in [14] with the aim of using their elements to qualify logical assertions in different ways. In this subsection we recall their axiomatic definition and some significant examples.

Definition 1. A *Qualification Domain* is any structure $\mathcal{D} = \langle D, \sqsubseteq, \bot, \top, \circ \rangle$ verifying the following requirements:

1. $\langle D, \sqsubseteq, \bot, \top \rangle$ is a lattice with extreme points $\bot$ and $\top$ w.r.t. the partial ordering $\sqsubseteq$. For given elements $d, e \in D$, we write $d \sqcap e$ for the *greatest lower bound* (*glb*) of d and e and $d \sqcup e$ for the *least upper bound* (*lub*) of d and e. We also write $d \sqsubset e$ as abbreviation for $d \sqsubseteq e \wedge d \neq e$.
2. $\circ : D \times D \to D$, called *attenuation operation*, verifies the following axioms:
   (a) $\circ$ is associative, commutative and monotonic w.r.t. $\sqsubseteq$.
   (b) $\forall d \in D : d \circ \top = d$.
   (c) $\forall d \in D : d \circ \bot = \bot$.
   (d) $\forall d, e \in D \setminus \{\bot, \top\} : d \circ e \sqsubset e$.
   (e) $\forall d, e_1, e_2 \in D : d \circ (e_1 \sqcap e_2) = d \circ e_1 \sqcap d \circ e_2$. □

In the rest of the paper, $\mathcal{D}$ will generally denote an arbitrary qualification domain. For any finite $S = \{e_1, e_2, \ldots, e_n\} \subseteq D$, the *glb* of S (noted as $\bigsqcap S$) exists and can be computed as $e_1 \sqcap e_2 \sqcap \cdots \sqcap e_n$ (which reduces to $\top$ in the case $n = 0$). As an easy consequence of the axioms, one gets the identity $d \circ \bigsqcap S = \bigsqcap \{d \circ e \mid e \in S\}$. The $QLP(\mathcal{D})$ scheme presented in [14] supports LP over a parametrically given qualification domain $\mathcal{D}$.

Example 1. *Some examples of qualification domains are presented below. Their intended use for qualifying logical assertions will become more clear in Subsection 3.1.*

1. $\mathcal{B} = (\{0, 1\}, \leq, 0, 1, \wedge)$*, where* 0 *and* 1 *stand for the two classical truth values* false *and* true*,* $\leq$ *is the usual numerical ordering over* $\{0, 1\}$*, and* $\wedge$ *stands for the classical conjunction operation over* $\{0, 1\}$*. Attaching* 1 *to an atomic formula* A *is intended to qualify* A *as 'true' in the sense of classical* LP*.*
2. $\mathcal{U} = (U, \leq, 0, 1, \times)$*, where* $U = [0, 1] = \{d \in \mathbb{R} \mid 0 \leq d \leq 1\}$*,* $\leq$ *is the usual numerical ordering, and* $\times$ *is the multiplication operation. In this domain, the top element* $\top$ *is* 1 *and the greatest lower bound* $\bigsqcap S$ *of a finite* $S \subseteq U$ *is the minimum value min(S), which is* 1 *if* $S = \emptyset$*. Attaching an element* $c \in U \setminus \{0\}$ *to an atomic formula* A *is intended to qualify* A *as 'true with certainty degree* c*' in the spirit of fuzzy logic, as done in the classical paper [20] by van Emden. The computation of qualifications* c *as certainty degrees in* $\mathcal{U}$ *is due to the interpretation of* $\sqcap$ *as* min *and* $\circ$ *as* $\times$*.*
3. $\mathcal{W} = (P, \geq, \infty, 0, +)$*, where* $P = [0, \infty] = \{d \in \mathbb{R} \cup \{\infty\} \mid d \geq 0\}$*,* $\geq$ *is the reverse of the usual numerical ordering (with* $\infty \geq d$ *for any* $d \in P$*), and* $+$ *is the addition operation (with*

$\infty + d = d + \infty = \infty$ for any $d \in P$). In this domain, the top element $\top$ is 0 and the greatest lower bound $\bigsqcap S$ of a finite $S \subseteq P$ is the maximum value max(S), which is 0 if $S = \emptyset$. Attaching an element $d \in P \setminus \{\infty\}$ to an atomic formula A is intended to qualify A as 'true with weighted proof depth d'. The computation of qualifications d as weighted proof depths in $\mathcal{W}$ is due to the interpretation of $\sqcap$ as max and $\circ$ as $+$.

4. *Given 2 qualification domains $\mathcal{D}_i = \langle D_i, \sqsubseteq_i, \bot_i, \top_i, \circ_i \rangle$ ($i \in \{1, 2\}$), their* cartesian product $\mathcal{D}_1 \times \mathcal{D}_2$ *is* $\mathcal{D} =_{\text{def}} \langle D, \sqsubseteq, \bot, \top, \circ \rangle$, *where* $D =_{\text{def}} D_1 \times D_2$, *the partial ordering* $\sqsubseteq$ *is defined as* $(d_1, d_2) \sqsubseteq (e_1, e_2) \Longleftrightarrow_{\text{def}} d_1 \sqsubseteq_1 e_1$ *and* $d_2 \sqsubseteq_2 e_2$, $\bot =_{\text{def}} (\bot_1, \bot_2)$, $\top =_{\text{def}} (\top_1, \top_2)$, *and the attenuation operator* $\circ$ *is defined as* $(d_1, d_2) \circ (e_1, e_2) =_{\text{def}} (d_1 \circ_1 e_1, d_2 \circ_2 e_2)$. *The product of two given qualification domains is always another qualification domain, as proved in [14]. Intuitively, each value (d_1, d_2) belonging to $\mathcal{D}_1 \times \mathcal{D}_2$ imposes the qualification d_1* and also *the qualification d_2. For instance, values (c, d) belonging to $\mathcal{U} \times \mathcal{W}$ impose two qualifications, namely: a certainty degree greater or equal than c and a weighted proof depth less or equal than d.* □

For technical reasons that will become apparent in Section 4, we consider the two structures $\mathcal{U}'$ resp. $\mathcal{W}'$ defined analogously to $\mathcal{U}$ resp. $\mathcal{W}$, except that $\circ$ behaves as min in $\mathcal{U}'$ and as max in $\mathcal{W}'$. Note that almost all the axioms for qualification domains enumerated in Definition 1 hold in $\mathcal{U}'$ and $\mathcal{W}'$, except that axiom 2.(d) holds only in the relaxed form $\forall d, e \in D : d \circ e \sqsubseteq e$. Therefore, we will refer to $\mathcal{U}'$ and $\mathcal{W}'$ as *quasi* qualification domains.

2.2 Similarity relations

Similarity relations over a given set S have been defined in [16] and related literature as mappings $\mathcal{R} : S \times S \to [0, 1]$ that satisfy three axioms analogous to those required for classical equivalence relations. Each value $\mathcal{R}(x, y)$ computed by a similarity relation $\mathcal{R}$ is called the *similarity degree* between x and y. In this paper we use a natural extension of the definition given in [16], allowing elements of an arbitrary qualification domain $\mathcal{D}$ to serve as similarity degrees. As in [16], we are especially interested in similarity relations over sets S whose elements are variables and symbols of a given signature.

Definition 2. Let a qualification domain $\mathcal{D}$ with carrier set D and a set S be given.

1. A *$\mathcal{D}$-valued similarity relation* over S is any mapping $\mathcal{R} : S \times S \to D$ such that the three following axioms hold for all $x, y, z \in S$:
   (a) *Reflexivity:* $\mathcal{R}(x, x) = \top$.
   (b) *Symmetry:* $\mathcal{R}(x, y) = \mathcal{R}(y, x)$.
   (c) *Transitivity:* $\mathcal{R}(x, z) \sqsupseteq \mathcal{R}(x, y) \sqcap \mathcal{R}(y, z)$.
2. The mapping $\mathcal{R} : S \times S \to D$ defined as $\mathcal{R}(x, x) = \top$ for all $x \in D$ and $\mathcal{R}(x, y) = \bot$ for all $x, y \in D$, $x \neq y$ is trivially a $\mathcal{D}$-valued similarity relation called the *identity*.
3. A $\mathcal{D}$-valued similarity relation $\mathcal{R}$ over S is called *admissible* iff $S = \mathcal{V}ar \cup CS \cup PS$ (where the three mutually disjoint sets $\mathcal{V}ar$, CS and PS stand for a countably infinite collection of *variables*, a set of *constructor symbols* and a set of *predicate symbols*, respectively) and the two following requirements are satisfied:
   (a) $\mathcal{R}$ restricted to $\mathcal{V}ar$ behaves as the identity, i.e. $\mathcal{R}(X, X) = \top$ for all $X \in \mathcal{V}ar$ and $\mathcal{R}(X, Y) = \bot$ for all $X, Y \in \mathcal{V}ar$, $X \neq Y$.
   (b) $\mathcal{R}(x, y) \neq \bot$ holds only if some of the following three cases holds x, y: either $x, y \in \mathcal{V}ar$ are both the same variable; or else $x, y \in CS$ are constructor symbols with the same arity; or else $x, y \in PS$ are predicate symbols with the same arity. □

The similarity degrees computed by a $\mathcal{D}$-valued similarity relation must be interpreted w.r.t. the intended role of $\mathcal{D}$-elements as qualification values. For example, let $\mathcal{R}$ be an admissible similarity relation, and let $c, d \in CS$ be two nullary constructor symbols (i.e., constants). If $\mathcal{R}$ is $\mathcal{U}$-valued, then $\mathcal{R}(c, d)$ can be interpreted as a *certainty degree* for the assertion that c and d are similar. On the other hand, if $\mathcal{R}$ is $\mathcal{W}$-valued, then $\mathcal{R}(c, d)$ can be interpreted as a *cost* to be paid for c to play the role of d. These two views are coherent with the different interpretations of the operators $\sqcap$ and $\circ$ in $\mathcal{U}$ and $\mathcal{W}$, respectively.

In the rest of the paper we assume that any admissible similarity relation $\mathcal{R}$ can be extended to act over terms, atoms and clauses. The extension, also called $\mathcal{R}$, can be recursively defined as in [16]. The following definition specifies the extension of $\mathcal{R}$ acting over terms. The case of atoms and clauses is analogous.

Definition 3. ($\mathcal{R}$ acting over terms).

1. For $X \in \mathcal{V}ar$ and for any term t different from X: $\mathcal{R}(X, X) = \top$ and $\mathcal{R}(X, t) = \mathcal{R}(t, X) = \bot$.
2. For $c, c' \in CS$ with different arities n, m: $\mathcal{R}(c(t_1, \ldots, t_n), c'(t'_1, \ldots, t'_m)) = \bot$.
3. For $c, c' \in CS$ with the same arity n: $\mathcal{R}(c(t_1, \ldots, t_n), c'(t'_1, \ldots, t'_n)) = \mathcal{R}(c, c') \sqcap \mathcal{R}(t_1, t'_1) \sqcap \ldots \sqcap \mathcal{R}(t_n, t'_n)$.

3. Similarity-based Qualified Logic Programming

In this section we extend our previous scheme $QLP(\mathcal{D})$ to a more expressive scheme called *Similarity-based Qualified Logic Programming* over $(\mathcal{R}, \mathcal{D})$ –abbreviated as $SQLP(\mathcal{R}, \mathcal{D})$– which supports both qualification over $\mathcal{D}$ in the sense of [14] and $\mathcal{R}$-based similarity in the sense of [16] and related research. Subsection 3.1 presents a quick review of the main results concerning syntax and declarative semantics of $QLP(\mathcal{D})$ already presented in [14], while the extensions needed to conform the new $SQLP(\mathcal{R}, \mathcal{D})$ scheme are presented in subsection 3.2.

3.1 Qualified Logic Programming

$QLP(\mathcal{D})$ was proposed in our previous work [14] as a generic scheme for qualified logic programming over a given qualification domain $\mathcal{D}$. In that scheme, a *signature* Σ providing constructor and predicate symbols with given arities is assumed. *Terms* are built from constructors and *variables* from a countably infinite set $\mathcal{V}ar$ (disjoint from Σ) and *Atoms* are of the form $p(t_1, \ldots, t_n)$ (shortened as $p(\overline{t_n})$ or simply $p(\overline{t})$) where p is a n-ary predicate symbol and t_i are terms. We write At_Σ, called the *open Herbrand base*, for the set of all atoms. A $QLP(\mathcal{D})$ program $\mathcal{P}$ is a finite set of *$\mathcal{D}$-qualified definite Horn clauses* of the form $A \leftarrow d- \overline{B}$ where A is an atom, $\overline{B}$ a finite conjunction of atoms and $d \in D \setminus \{\bot\}$ is the *attenuation value* attached to the clause's implication.

As explained in [14], in our aim to work with qualifications we are not only interested in just proving an atom, but in proving it along with a qualification value. For this reason, *$\mathcal{D}$-qualified atoms* ($A \sharp d$ where A is an atom and $d \in D \setminus \{\bot\}$) are introduced to represent the statement that the atom A holds for *at least* the qualification value d. For use in goals to be solved, *open $\mathcal{D}$-annotated atoms* ($A \sharp W$ where A is an atom and W a *qualification variable* intended to take values over $\mathcal{D}$) are also introduced, and a countably infinite set $\mathcal{W}ar$ of qualification variables (disjoint from $\mathcal{V}ar$ and Σ) is postulated. The *annotated Herbrand base* over $\mathcal{D}$ is defined as the set $\text{At}_\Sigma(\mathcal{D})$ of all $\mathcal{D}$-qualified atoms. A *$\mathcal{D}$-entailment relation* over $\text{At}_\Sigma(\mathcal{D})$, defined as $A \sharp d \succcurlyeq_\mathcal{D} A' \sharp d'$ iff there is some substitution θ such that $A' = A\theta$ and $d' \sqsubseteq d$, is used to for-

mally define an *open Herbrand interpretation* over $\mathcal{D}$ –from now on just an *interpretation*– as any subset $\mathcal{I} \subseteq \mathrm{At}_\Sigma(\mathcal{D})$ which is closed under $\mathcal{D}$-entailment. We write $\mathrm{Int}_\Sigma(\mathcal{D})$ for the family of all interpretations. The notion of model is such that given any clause $C \equiv A \leftarrow\!\!d\!\!- B_1, \ldots, B_k$ in the $QLP(\mathcal{D})$ program $\mathcal{P}$, an interpretation $\mathcal{I}$ is said to be a *model* of C iff for any substitution θ and any qualification values $d_1, \ldots, d_k \in D \setminus \{\bot\}$ such that $B_i\theta \sharp d_i \in \mathcal{I}$ for all $1 \leq i \leq k$, one has $A\theta \sharp (d \circ \bigsqcap\{d_1, \ldots, d_k\}) \in \mathcal{I}$. The interpretation $\mathcal{I}$ is also said to be a model of the $QLP(\mathcal{D})$ program $\mathcal{P}$ (written as $\mathcal{I} \models \mathcal{P}$) iff it happen to be a model of every clause in $\mathcal{P}$.

As technique to infer formulas (or in our case $\mathcal{D}$-qualified atoms) from a given $QLP(\mathcal{D})$ program $\mathcal{P}$, and following traditional ideas, we consider two alternative ways of formalizing an inference step which goes from the body of a clause to its head: both an interpretation transformer $\mathrm{T}_\mathcal{P} : \mathrm{Int}_\Sigma(\mathcal{D}) \to \mathrm{Int}_\Sigma(\mathcal{D})$, and a qualified variant of Horn Logic, noted as $QHL(\mathcal{D})$, called *Qualified Horn Logic* over $\mathcal{D}$. As both methods are equivalent and correctly characterize the least Herbrand model of a given program $\mathcal{P}$, we will only be recalling the logic $QHL(\mathcal{D})$, although we encourage the reader to see Section 3.2 in [14], where the fix-point semantics is explained.

The logic $QHL(\mathcal{D})$ is defined as a deductive system consisting just of one inference rule: QMP($\mathcal{D}$), called *Qualified Modus Ponens* over $\mathcal{D}$. Such rule allows us to give the following inference step given that there were some $(A \leftarrow\!\!d\!\!- B_1, \ldots, B_k) \in \mathcal{P}$, some substitution θ such that $A' = A\theta$ and $B'_i = B_i\theta$ for all $1 \leq i \leq k$ and some $d' \in D \setminus \{\bot\}$ such that $d' \sqsubseteq d \circ \bigsqcap\{d_1, \ldots, d_k\}$:

$$\frac{B'_1 \sharp d_1 \quad \cdots \quad B'_k \sharp d_k}{A' \sharp d'} \quad \text{QMP}(\mathcal{D})$$

Roughly, each QMP($\mathcal{D}$) inference step using an instance of a program clause $A \leftarrow\!\!d\!\!- B$ has the effect of propagating to the head the *qualification value* $d \circ b$, where b is the infimum in $\mathcal{D}$ of the qualification values $d_i \in D \setminus \{\bot\}$ previously computed for the various atoms occurring in the body. This helps to understand the claims made in Example 1 above about the intended use of elements of the domains $\mathcal{U}$ and $\mathcal{W}$ for qualifying logical assertions. We use the notations $\mathcal{P} \vdash_{\mathrm{QHL}(\mathcal{D})} A \sharp d$ (resp. $\mathcal{P} \vdash^n_{\mathrm{QHL}(\mathcal{D})} A \sharp d$) to indicate that $A \sharp d$ can be inferred from the clauses in program $\mathcal{P}$ in finitely many steps (resp. n steps). The *least Herbrand model of* $\mathcal{P}$ happens to be $\mathcal{M}_\mathcal{P} = \{A \sharp d \mid \mathcal{P} \vdash_{\mathrm{QHL}(\mathcal{D})} A \sharp d\}$, as proved in [14].

3.2 Similarity-based Qualified Logic Programming

The scheme $SQLP(\mathcal{R}, \mathcal{D})$ presented in this subsection has two parameters $\mathcal{R}$ and $\mathcal{D}$, where $\mathcal{D}$ can be any qualification domain and $\mathcal{R}$ can be any admissible $\mathcal{D}$-valued similarity relation, in the sense of Definition 2. The new scheme subsumes the approach in [14] by behaving as $QLP(\mathcal{D})$ in the case that $\mathcal{R}$ is chosen as the identity, and it also subsumes similarity-based LP by behaving as the approach in [16] and related papers in the case that $\mathcal{D}$ is chosen as $\mathcal{U}$.

Syntactically, $SQLP(\mathcal{R}, \mathcal{D})$ presents almost no changes w.r.t. $QLP(\mathcal{D})$, but the declarative semantics must be extended to account for the behavior of the parametrically given similarity relation $\mathcal{R}$. As in the previous subsection, we assume a signature Σ providing again constructor and predicate symbols. *Terms* and *Atoms* are built the same way they were in $QLP(\mathcal{D})$, and At_Σ will stand again for the set of all atoms, called the *open Herbrand base*. An atom A is called *linear* if there is no variable with multiple occurrences in A; otherwise A is called *non-linear*. A $SQLP(\mathcal{R}, \mathcal{D})$ program $\mathcal{P}$ is a finite set of *$\mathcal{D}$-qualified definite Horn clauses* with the same syntax as in $QLP(\mathcal{D})$, along with a $\mathcal{D}$-valued admissible similarity relation $\mathcal{R}$ in the sense of Definition 2, item 2. Figure 1 shows a simple $SQLP(\mathcal{R}, \mathcal{U})$ program built from the similarity

```
 1 wild(lynx) <-0.9-
 2 wild(boar) <-0.9-
 3 wild(snake) <-1.0-

 4 farm(cow) <-1.0-
 5 farm(pig) <-1.0-

 6 domestic(cat) <-0.8-
 7 domestic(snake) <-0.4-

 8 intelligent(A) <-0.9- domestic(A)
 9 intelligent(lynx) <-0.7-

10 pacific(A) <-0.9- domestic(A)
11 pacific(A) <-0.7- farm(A)

12 pet(A) <-1.0- pacific(A), intelligent(A)
```

```
R(farm,domestic) = 0.3
R(pig,boar) = 0.7
R(lynx,cat) = 0.8
```

Figure 1. $SQLP(\mathcal{R}, \mathcal{U})$ program.

relation $\mathcal{R}$ given in the same figure and the qualification domain $\mathcal{U}$ for certainty values. This program will be used just for illustrative purposes in the rest of the paper. The reader is referred to Section 2 for other examples of qualification domains, and to the references [8, 11] for suggestions concerning practical applications of similarity-based LP.

$\mathcal{D}$-qualified atoms ($A \sharp d$ with A an atom and $d \in D \setminus \{\bot\}$) and *open $\mathcal{D}$-annotated atoms* ($A \sharp W$ with A and atom and $W \in Var$ a qualification variable intended to take values in $D \setminus \{\bot\}$) will still be used here. Similarly, the *annotated open Herbrand base* over $\mathcal{D}$ is again defined as the set $\mathrm{At}_\Sigma(\mathcal{D})$ of all $\mathcal{D}$-qualified atoms. At this point, and before extending the notions of $\mathcal{D}$-entailment relation and interpretation to the $SQLP(\mathcal{R}, \mathcal{D})$ scheme, we need to define what an $\mathcal{R}$-instance of an atom is. Intuitively, when building $\mathcal{R}$-instances of an atom A, signature symbols occurring in A can be replaced by similar ones, and different occurrences of the same variable in A may be replaced by different terms, whose degree of similarity must be taken into account. Technically, $\mathcal{R}$-instances of an atom $A \in \mathrm{At}_\Sigma$ are built from a linearized version of A which has the form $\mathrm{lin}(A) = (A_\ell, \mathcal{S}_\ell)$ and is constructed as follows: A_ℓ is a linear atom built from A by replacing each n additional occurrences of a variable X by new fresh variables X_i ($1 \leq i \leq n$); and $\mathcal{S}_\ell$ is a set of *similarity conditions* $X \sim X_i$ (with $1 \leq i \leq n$) asserting the similarity of all variables in A_ℓ that correspond to the same variable X in A. As a concrete illustration, let us show the linearization of two atoms. Note what happens when the atom A is already linear as in the first case: A_ℓ is just the same as A and $\mathcal{S}_\ell$ is empty.

- $H_1 = p(c(X), Y)$
  $\mathrm{lin}(H_1) = (p(c(X), Y),\ \{\})$
- $H_2 = p(c(X), X, Y)$
  $\mathrm{lin}(H_2) = (p(c(X), X_1, Y),\ \{X \sim X_1\})$

Now we are set to formally define the $\mathcal{R}$-instances of an atom.

Definition 4. ($\mathcal{R}$-instance of an atom). Assume an atom $A \in \mathrm{At}_\Sigma$ and its linearized version $\mathrm{lin}(A) = (A_\ell, \mathcal{S}_\ell)$. Then, an atom A' is

said to be an $\mathcal{R}$-instance of A with similarity degree δ, noted as $(A', \delta) \in [A]_{\mathcal{R}}$, iff there are some atom A^S and some substitution θ such that $A' = A^S\theta$ and $\delta = \mathcal{R}(A_\ell, A^S) \sqcap \bigsqcap\{\mathcal{R}(X_i\theta, X_j\theta) \mid (X_i \sim X_j) \in \mathcal{S}_\ell\} \neq \bot$.

Next, the $(\mathcal{R}, \mathcal{D})$-*entailment relation* over $\mathrm{At}_\Sigma(\mathcal{D})$ is defined as follows: $A \sharp d \succcurlyeq_{(\mathcal{R},\mathcal{D})} A' \sharp d'$ iff there is some similarity degree δ such that $(A', \delta) \in [A]_{\mathcal{R}}$ and $d' \sqsubseteq d \circ \delta$. Finally, an *open Herbrand interpretation* –just *interpretation* from now on– over $(\mathcal{R}, \mathcal{D})$ is defined as any subset $\mathcal{I} \in \mathrm{At}_\Sigma(\mathcal{D})$ which is closed under $(\mathcal{R}, \mathcal{D})$-entailment. That is, an interpretation $\mathcal{I}$ including a given $\mathcal{D}$-qualified atom $A \sharp d$ is required to include all the 'similar instances' $A' \sharp d'$ such that $A \sharp d \succcurlyeq_{(\mathcal{R},\mathcal{D})} A' \sharp d'$, because we intend to formalize a semantics in which all such similar instances are valid whenever $A \sharp d$ is valid. This complements the intuition given for the $\mathcal{D}$-entailment relation in $QLP(\mathcal{D})$ to include the similar instances (obtainable due to $\mathcal{R}$) of each atom, and not only those which are true because we can prove them for a better (i.e. higher in $\mathcal{D}$) qualification. Note that $(\mathcal{R}, \mathcal{D})$-entailment is a refinement of $\mathcal{D}$-entailment, since: $A \sharp d \succcurlyeq_{\mathcal{D}} A' \sharp d' \Longrightarrow$ there is some substitution θ such that $A' = A\theta$ and $d' \sqsubseteq d \Longrightarrow (A', \top) \in [A]_{\mathcal{R}}$ and $d' \sqsubseteq d \circ \top \Longrightarrow A \sharp d \succcurlyeq_{(\mathcal{R},\mathcal{D})} A' \sharp d'$.

As an example of the closure of interpretations w.r.t. $(\mathcal{R}, \mathcal{D})$-entailment, consider the $\mathcal{U}$-qualified atom `domestic(cat)#0.8`. As a trivial consequence of Proposition 2 below, this atom belongs to the least Herbrand model of the program in Figure 1. On the other hand, we also know that `lynx` is similar to `cat` with a similarity degree of 0.8 w.r.t. the similarity relation $\mathcal{R}$ in Figure 1. Therefore, `domestic(lynx)` is a $\mathcal{R}$-instance of `domestic(cat)` to the degree 0.8. Then, by definition of $(\mathcal{R}, \mathcal{U})$-entailment, it turns out that `domestic(cat)#0.8` $\succcurlyeq_{(\mathcal{R},\mathcal{U})}$ `domestic(lynx)#0.64`, and the $\mathcal{U}$-qualified atom `domestic(lynx)#0.64` does also belong to the least model of the example program. Intuitively, $0.64 = 0.8 \times 0.8$ is the best $\mathcal{U}$-qualification which can be inferred from the $\mathcal{U}$-qualification 0.8 for `domestic(cat)` and the $\mathcal{R}$-similarity 0.8 between `domestic(cat)` and `domestic(lynx)`.

We will write $\mathrm{Int}_\Sigma(\mathcal{R}, \mathcal{D})$ for the family of all interpretations over $(\mathcal{R}, \mathcal{D})$, a family for which the following proposition can be easily proved from the definition of an interpretation and the definitions of the union and intersection of a family of sets.

Proposition 1. *The family* $\mathrm{Int}_\Sigma(\mathcal{R}, \mathcal{D})$ *of all interpretations over* $(\mathcal{R}, \mathcal{D})$ *is a complete lattice under the inclusion ordering* $\subseteq$*, whose extreme points are* $\mathrm{Int}_\Sigma(\mathcal{R}, \mathcal{D})$ *as maximum and* $\emptyset$ *as minimum. Moreover, given any family of interpretations* $I \subseteq \mathrm{Int}_\Sigma(\mathcal{R}, \mathcal{D})$*, its lub and glb are* $\bigsqcap I = \bigcup\{\mathcal{I} \in \mathrm{Int}_\Sigma(\mathcal{R}, \mathcal{D}) \mid \mathcal{I} \in I\}$ *and* $\bigsqcup I = \bigcap\{\mathcal{I} \in \mathrm{Int}_\Sigma(\mathcal{R}, \mathcal{D}) \mid \mathcal{I} \in I\}$*, respectively.*

Similarly as we did for the $\mathcal{R}$-instances of an atom, we will define what the $\mathcal{R}$-instances of a clause are. The following definition tells us so.

Definition 5. ($\mathcal{R}$-instance of a clause). Assume a clause $C \equiv A \leftarrow d- B_1, \ldots, B_k$ and the linearized version of its head atom $\mathrm{lin}(A) = (A_\ell, \mathcal{S}_\ell)$. Then, a clause C' is said to be an $\mathcal{R}$-instance of C with similarity degree δ, noted as $(C', \delta) \in [C]_{\mathcal{R}}$, iff there are some atom A^S and some substitution θ such that $\delta = \mathcal{R}(A_\ell, A^S) \sqcap \bigsqcap\{\mathcal{R}(X_i\theta, X_j\theta) \mid (X_i \sim X_j) \in \mathcal{S}_\ell\} \neq \bot$ and $C' \equiv A^S\theta \leftarrow d- B_1\theta, \ldots, B_k\theta$.

Note that as an immediate consequence from Definitions 4 and 5 it is true that given two clauses C and C' such that $(C', \delta) \in [C]_{\mathcal{R}}$, and assuming A to be head atom of C and A' to be the head atom of C', then we have that $(A', \delta) \in [A]_{\mathcal{R}}$.

Let C be any clause $A \leftarrow d- B_1, \ldots, B_k$ in the program $\mathcal{P}$, and $\mathcal{I} \in \mathrm{Int}_\Sigma(\mathcal{R}, \mathcal{D})$ any interpretation over $(\mathcal{R}, \mathcal{D})$. We say that $\mathcal{I}$ is a model of C iff for any clause $C' \equiv H' \leftarrow d- B'_1, \ldots, B'_k$ such that $(C', \delta) \in [C]_{\mathcal{R}}$ and any qualification values $d_1, \ldots, d_k \in D \setminus \{\bot\}$ such that $B'_i \sharp d_i \in \mathcal{I}$ for all $1 \leq i \leq k$, one has $H' \sharp d' \in \mathcal{I}$ where $d' = d \circ \bigsqcap\{e, d_1, \ldots, d_k\}$. And we say that $\mathcal{I}$ is a model of the $SQLP(\mathcal{R}, \mathcal{D})$ program $\mathcal{P}$ (also written $\mathcal{I} \models \mathcal{P}$) iff $\mathcal{I}$ is a model of each clause in $\mathcal{P}$.

We will provide now a way to perform an inference step from the body of a clause to its head. As in the case of $QLP(\mathcal{D})$, this can be formalized in two alternative ways, namely an interpretation transformer and a variant of Horn Logic. Both approaches lead to equivalent characterizations of least program models. Here we focus on the second approach, defining what we will call *Similarity-based Qualified Horn Logic* over $(\mathcal{R}, \mathcal{D})$ – abbreviated as $SQHL(\mathcal{R}, \mathcal{D})$–, another variant of Horn Logic and an extension of the previous $QHL(\mathcal{D})$. The logic $SQHL(\mathcal{R}, \mathcal{D})$ is also defined as a deductive system consisting just of one inference rule $SQMP(\mathcal{R}, \mathcal{D})$, called *Similarity-based Qualified Modus Ponens* over $(\mathcal{R}, \mathcal{D})$:

If $((A' \leftarrow d- B'_1, \ldots, B'_k), \delta) \in [C]_{\mathcal{R}}$ for some clause $C \in \mathcal{P}$ with attenuation value d, then the following inference step is allowed for any $d' \in D \setminus \{\bot\}$ such that $d' \sqsubseteq d \circ \bigsqcap\{\delta, d_1, \ldots, d_k\}$:

$$\frac{B'_1 \sharp d_1 \quad \cdots \quad B'_k \sharp d_k}{A' \sharp d'} \quad \text{SQMP}(\mathcal{R}, \mathcal{D}) \ .$$

We will use the notations $\mathcal{P} \vdash_{\mathrm{SQHL}(\mathcal{R},\mathcal{D})} A \sharp d$ (respectively $\mathcal{P} \vdash^n_{\mathrm{SQHL}(\mathcal{R},\mathcal{D})} A \sharp d$) to indicate that $A \sharp d$ can be inferred from the clauses in program $\mathcal{P}$ in finitely many steps (respectively n steps). Note that $SQHL(\mathcal{R}, \mathcal{D})$ proofs can be naturally represented as upwards growing *proof trees* with $\mathcal{D}$-qualified atoms at their nodes, each node corresponding to one inference step having the children nodes as premises.

The following proposition contains the main result concerning the declarative semantics of the $SQLP(\mathcal{R}, \mathcal{D})$ scheme. A full proof can be developed in analogy to the $QLP(\mathcal{D})$ case presented in [14, 13].

Proposition 2. *Given any* $SQLP(\mathcal{R}, \mathcal{D})$ *program* $\mathcal{P}$*. The* least Herbrand model *(*$\mathcal{M}_\mathcal{P}$*) of* $\mathcal{P}$ *is*

$$\{A \sharp d \mid \mathcal{P} \vdash_{\mathrm{SQHL}(\mathcal{R},\mathcal{D})} A \sharp d\} \ .$$

The following example serves as an illustration of how the logic $SQHL(\mathcal{R}, \mathcal{D})$ works over $(\mathcal{R}, \mathcal{U})$ using the example program displayed in Figure 1.

Example 2. *The following proof tree proves that the atom* `pet(lynx)` *can be inferred for at least a qualification value of* 0.50 *in the* $SQLP(\mathcal{R}, \mathcal{U})$ *program* $\mathcal{P}$ *of Figure 1. Let's see it:*

$$\dfrac{\dfrac{\dfrac{}{\texttt{domestic(lynx)\#0.64}}\,(4)}{\texttt{pacific(lynx)\#0.57}}\,(2) \quad \dfrac{}{\texttt{intelligent(lynx)\#0.70}}\,(3)}{\texttt{pet(lynx)\#0.50}}\,(1)$$

where the clauses and qualification values used for each inference step are:

(1) `pet(lynx) <-1.0- pacific(lynx),intelligent(lynx)` *is an instance of clause* 12 *in* $\mathcal{P}$ *and* $0.50 \leq 1.0 \times \min\{1.0, 0.57, 0.70\}$*. Note that the first* 1.0 *in the minimum is the one which comes from the similarity relation as for this step we are just using a plain instance of clause* 12 *in* $\mathcal{P}$*.*

(2) `pacific(lynx) <-0.9- domestic(lynx)` *is a plain instance of clause* 10 *in* $\mathcal{P}$ *and* $0.57 \leq 0.9 \times \min\{1.0, 0.64\}$*.*

(3) `intelligent(lynx) <-0.7-` *is clause* 9 *in* $\mathcal{P}$ *and* $0.70 \leq 0.70 \times \min\{1.0\}$*.*

(4) The clause `domestic(lynx) <-0.8-` *is an* $\mathcal{R}$*-instance of clause* 6 *with a similarity degree of* 0.8 *and we have* $0.64 \leq 0.8 \times \min\{0.8\}$*.* □

4. Reducing Similarities to Qualifications

4.1 A Program Transformation

In this section we prove that any $SQLP(\mathcal{R},\mathcal{D})$ program $\mathcal{P}$ can be transformed into an equivalent $QLP(\mathcal{D})$ program which will be denoted by $S_{\mathcal{R}}(\mathcal{P})$. The program transformation is defined as follows:

Definition 6. Let $\mathcal{P}$ be a $SQLP(\mathcal{R},\mathcal{D})$ program. We define the transformed program $S_{\mathcal{R}}(\mathcal{P})$ as:

$$S_{\mathcal{R}}(\mathcal{P}) = \mathcal{P}_S \cup \mathcal{P}_\sim \cup \mathcal{P}_{\text{pay}}$$

where the auxiliary sets of clauses $\mathcal{P}_S, \mathcal{P}_\sim, \mathcal{P}_{\text{pay}}$ are defined as:

- For each clause $(H \leftarrow d - \overline{B}) \in \mathcal{P}$ and for each H' such that $\mathcal{R}(H_\ell, H') \neq \bot$
$$(H' \leftarrow d - pay_{\mathcal{R}(H_\ell,H')}, S_\ell, \overline{B}) \in \mathcal{P}_S$$
where $(H_\ell, S_\ell) = lin(H)$.
- $\mathcal{P}_\sim = \{X \sim X \leftarrow \top -\} \cup \{(c(\overline{X}_n) \sim c'(\overline{Y}_n) \leftarrow \top - pay_{\mathcal{R}(c,c')}, X_1 \sim Y_1, \ldots, X_n \sim Y_n) \mid c, c' \in CS \text{ of arity } n, \mathcal{R}(c,c') \neq \bot\}$
- $\mathcal{P}_{\text{pay}} = \{(pay_w \leftarrow w -) \mid \text{for each atom } pay_w \text{ occurring in } \mathcal{P}_\sim \cup \mathcal{P}_S\}$

Note that the linearization of clause heads in this transformation is motivated by the role of linearized atoms in the $SQHL(\mathcal{R},\mathcal{D})$ logic defined in Subsection 3.2 to specify the declarative semantics of $SQLP(\mathcal{R},\mathcal{D})$ programs. For instance, assume a $SQLP(\mathcal{R},\mathcal{U})$ program $\mathcal{P}$ including the clause $p(X,X) \leftarrow 1.0-$ and two nullary constructors c, d such that $\mathcal{R}(c,d) = 0.8$. Then, $SQHL(\mathcal{R},\mathcal{U})$ supports the derivation $\mathcal{P} \vdash_{\text{SQHL}(\mathcal{R},\mathcal{U})} p(c,d) \sharp 0.8$, and the transformed program $S_{\mathcal{R}}(\mathcal{P})$ will include the clauses

$$\begin{array}{ll} p(X,X_1) & \leftarrow 1.0 - pay_{1.0}, X \sim X_1, \\ X \sim X & \leftarrow 1.0-, \\ c \sim d & \leftarrow 1.0 - pay_{0.8}, \\ pay_{1.0} & \leftarrow 1.0-, \\ pay_{0.8} & \leftarrow 0.8- \end{array}$$

thus enabling the corresponding derivation $S_{\mathcal{R}}(\mathcal{P}) \vdash_{\text{QHL}(\mathcal{U})} p(c,d) \sharp 0.8$ in $QHL(\mathcal{U})$.

In general, $\mathcal{P}$ and $S_{\mathcal{R}}(\mathcal{P})$ are semantically equivalent in the sense that $\mathcal{P} \vdash_{\text{SQHL}(\mathcal{R},\mathcal{D})} A \sharp d \iff S_{\mathcal{R}}(\mathcal{P}) \vdash_{\text{QHL}(\mathcal{D})} A \sharp d$ holds for any $\mathcal{D}$-qualified atom $A \sharp d$, as stated in Theorem 1 below. The next technical lemma will be useful for the proof of this theorem.

Lemma 1. *Let $\mathcal{P}$ be a $SQLP(\mathcal{R},\mathcal{D})$ program and $S_{\mathcal{R}}(\mathcal{P})$ its transformed program according to Definition 6. Let t, s be two terms in $\mathcal{P}$'s signature and $d \in D \setminus \{\bot\}$. Then:*

1. $S_{\mathcal{R}}(\mathcal{P}) \vdash_{\text{QHL}(\mathcal{D})} (t \sim s) \sharp d \Longrightarrow d \sqsubseteq \mathcal{R}(t,s)$
2. $\mathcal{R}(t,s) = d \Longrightarrow S_{\mathcal{R}}(\mathcal{P}) \vdash_{\text{QHL}(\mathcal{D})} (t \sim s) \sharp d$

Proof. We prove the two items separately.

1. Let T be a $QHL(\mathcal{D})$ proof tree witnessing
$$S_{\mathcal{R}}(\mathcal{P}) \vdash_{\text{QHL}(\mathcal{D})} (t \sim s) \sharp d$$
We prove by induction on number of nodes of T that $d \sqsubseteq \mathcal{R}(t,s)$. The basis case, with T consisting of just one node, must correspond to some inference without premises, i.e., a clause with empty body for $\sim$. Checking $P_\sim$ we observe that $X \sim X \leftarrow \top -$ is the only possibility. In this case t and s must be the same term and by the reflexivity of $\mathcal{R}$ (Def. 2), $\mathcal{R}(t,s) = \top$, which means $d \sqsubseteq \mathcal{R}(t,s)$ for every d. In the inductive step, we consider T with more than one node. Then the inference step at the root of T uses some clause $(c(\overline{X}_n) \sim c'(\overline{X}'_n) \leftarrow \top - pay_{\mathcal{R}(c,c')}, X_1 \sim X'_1, \ldots, X_n \sim X'_n) \in \mathcal{P}_\sim$, and must be of the form:
$$\frac{pay_w \sharp v \;\; (t_1 \sim s_1) \sharp e_1 \;\; \ldots \;\; (t_n \sim s_n) \sharp e_n}{c(\overline{t}_n) \sim c'(\overline{s}_n) \sharp d}$$
where $w = \mathcal{R}(c,c')$, $v \in \mathcal{D}$, $v \sqsubseteq w$, $t = c(\overline{t}_n)$, $s = c'(\overline{s}_n)$, and $e_1, \ldots, e_n$ s.t. $d \sqsubseteq \top \circ \bigsqcap\{v, e_1, \ldots, e_k\}$, i.e., $d \sqsubseteq \bigsqcap\{v, e_1, \ldots, e_k\}$. By induction hypothesis $e_i \sqsubseteq \mathcal{R}(t_i, s_i)$ for $i = 1 \ldots n$. Then $d \sqsubseteq \bigsqcap\{v, e_1, \ldots, e_n\}$ implies $d \sqsubseteq \bigsqcap\{w, \mathcal{R}(t_1,s_1), \ldots, \mathcal{R}(t_n,s_n)\}$ and hence $d \sqsubseteq \mathcal{R}(t,s)$ (Def. 3, item 3).
2. If $\mathcal{R}(t,s) = d$, $d \neq \bot$, we prove that $S_{\mathcal{R}}(\mathcal{P}) \vdash_{\text{QHL}(\mathcal{D})} (t \sim s) \sharp d$ by induction on the syntactic structure of t. The basis corresponds to the case $t = c$ for some constant c, or $t = Y$ for some variable Y. If $t = c$ then $s = c'$ for some other constant c'. By Definition 6 there is a clause in $\mathcal{P}_\sim$ of the form $(c \sim c' \leftarrow \top - pay_d)$. Using this clause and the identity substitution we can write the root inference step of a proof for $S_{\mathcal{R}}(\mathcal{P}) \vdash_{\text{QHL}(\mathcal{D})} (c \sim c') \sharp d$ as follows:
$$\frac{pay_d \sharp d}{c \sim c' \sharp d}$$
The condition required by the inference rule QMP($\mathcal{D}$) is in this particular case $d \sqsubseteq \top \circ \bigsqcap\{d\}$, and $\top \circ \bigsqcap\{d\} = d$. Proving the only premise $pay_d \sharp d$ in $QHL(\mathcal{D})$ is direct from its definition. If $t = Y$, with Y a variable, then $s = Y$ and $d = \top$ (otherwise $\mathcal{R}(t,s) = \bot$). Then $S_{\mathcal{R}}(\mathcal{P}) \vdash_{\text{QHL}(\mathcal{D})} (Y \sim Y) \sharp \top$ can be proved by using the clause $(X \sim X \leftarrow \top -) \in \mathcal{P}_\sim$ with substitution $\theta = \{X \mapsto Y\}$.
In the inductive step, t must be of the form $c(\overline{t}_n)$, with $n \geq 1$, and then s must be of the form $c'(\overline{s}_n)$ (otherwise $\mathcal{R}(t,s) = \bot$). From $d = \mathcal{R}(t,s) \neq \bot$ (hypotheses of the lemma) and Definition 3 we have that $\mathcal{R}(c,c') \neq \bot$. Then, by Definition 6, there is a clause in $\mathcal{P}_\sim$ of the form:
$$c(\overline{X}_n) \sim c'(\overline{Y}_n) \leftarrow \top - pay_{\mathcal{R}(c,c')}, X_1 \sim Y_1, \ldots, X_n \sim Y_n$$
By using the substitution $\theta = \{X_1 \mapsto t_1, \ldots, X_n \mapsto t_n, Y_1 \mapsto s_1, \ldots, Y_n \mapsto s_n\}$ we can write the root inference step in $QHL(\mathcal{D})$ as:
$$\frac{pay_{\mathcal{R}(c,c')} \sharp \mathcal{R}(c,c') \; (t_i \sim s_i \sharp \mathcal{R}(t_i,s_i))_{i=1\ldots n}}{c(\overline{t}_n) \sim c'(\overline{s}_n) \sharp d}$$
The inference can be applied because the condition
$$d \sqsubseteq \top \circ \bigsqcap\{\mathcal{R}(c,c'), \mathcal{R}(t_1,s_1), \ldots, \mathcal{R}(t_n,s_n)\}$$
reduces to
$$d \sqsubseteq \bigsqcap\{\mathcal{R}(c,c'), \mathcal{R}(t_1,s_1), \ldots, \mathcal{R}(t_n,s_n)\}$$
which holds by Definition 3, item 3. Moreover, the premises $t_i \sim s_i \sharp \mathcal{R}(t_i,s_i)$, $i = 1 \ldots n$, hold in $QHL(\mathcal{D})$ due to the inductive hypotheses, and proving
$$pay_{\mathcal{R}(c,c')} \sharp \mathcal{R}(c,c')$$
is straightforward from its definition. □

Now we can prove the equivalence between semantic inferences in $QHL(\mathcal{D})$ w.r.t. $\mathcal{P}$ and semantic inferences in $SQHL(\mathcal{R},\mathcal{D})$ w.r.t. $S_{\mathcal{R}}(\mathcal{P})$.

Theorem 1. *Let $\mathcal{P}$ be a $SQLP(\mathcal{R},\mathcal{D})$ program, A an atom in $\mathcal{P}$'s signature and $d \in D \setminus \{\bot\}$. Then:*

$$\mathcal{P} \vdash_{\text{SQHL}(\mathcal{R},\mathcal{D})} A \sharp d \iff S_{\mathcal{R}}(\mathcal{P}) \vdash_{\text{QHL}(\mathcal{D})} A \sharp d \;.$$

Proof. Let T be a $SQHL(\mathcal{R},\mathcal{D})$ proof tree for some annotated atom $A\,\sharp\,d$ in $\mathcal{P}$'s signature witnessing $\mathcal{P} \vdash_{\mathrm{SQHL}(\mathcal{R},\mathcal{D})} A\,\sharp\,d$. We prove that $S_\mathcal{R}(\mathcal{P}) \vdash_{\mathrm{QHL}(\mathcal{D})} A\,\sharp\,d$ by induction on the number of nodes of T.

The inference step at the root of T must be of the form

$$\frac{B'_1\,\sharp\,d_1 \quad \cdots \quad B'_k\,\sharp\,d_k}{A\,\sharp\,d} \qquad (1)$$

with $((A \leftarrow e - B'_1,\ldots,B'_k),\delta) \in [C]_\mathcal{R}$ for some clause $C \equiv (H \leftarrow e - B_1,\ldots,B_k) \in \mathcal{P}$ (observe that the case $k = 0$ corresponds to the induction basis). By Definition 5, $A = H'\theta$, $B'_i = B_i\theta$ for some substitution θ and atom H' such that $\delta = \mathcal{R}(H_\ell,H') \sqcap \bigsqcap\{\mathcal{R}(X_i\theta,\,X_j\theta) \mid (X_i \sim X_j) \in \mathcal{S}_\ell\} \neq \bot$, with $\mathrm{lin}(H) = (H_\ell,\mathcal{S}_\ell)$. This means in particular that $w = \mathcal{R}(H_\ell,H') \neq \bot$, which by Definition 6 implies that there is a clause C' in $S_\mathcal{R}(\mathcal{P})$ of the form $C' \equiv (H' \leftarrow e - pay_w, S_\ell, B_1,\ldots B_k)$. Then the root inference step of the deduction proving $\mathcal{P} \vdash_{\mathrm{QHL}(\mathcal{D})} A\,\sharp\,d$ will use the inference rule QMP($\mathcal{D}$) with C' and substitution θ (such that $H'\theta = A$) as follows:

$$\frac{pay_w\theta\,\sharp\,w \quad ((u_i \sim v_i)\theta\,\sharp\,e_i)_{1\le i\le m} \quad B'_1\,\sharp\,d_1 \cdots B'_k\,\sharp\,d_k}{A\,\sharp\,d} \qquad (2)$$

where $S_\ell = \{u_1 \sim v_1,\ldots,u_m \sim v_m\}$, and $e_i = \mathcal{R}(u_i\theta,v_i\theta)$ for $i = 1\ldots m$.

Next we check that the premises can be proved from $S_\mathcal{R}(\mathcal{P})$ in $QHL(\mathcal{D})$:

- $pay_w\theta = pay_w$, since pay_w is a nullary predicate for every w. Therefore $S_\mathcal{R}(\mathcal{P}) \vdash_{\mathrm{QHL}(\mathcal{D})} pay_w\,\sharp\,w$ is immediate from the definition of pay_w in Definition 6.
- For each $1 \le i \le m$, we observe that $\mathcal{R}(u_i\theta,v_i\theta) \neq \bot$ because $\delta \neq \bot$ has been computed above as the infimum of a set including $\mathcal{R}(u_i\theta,v_i\theta)$ among its members. Then $S_\mathcal{R}(\mathcal{P}) \vdash_{\mathrm{QHL}(\mathcal{D})} (u_i \sim v_i)\theta$ holds by Lemma 1, item 2.
- For each $1 \le i \le k$, (1) shows that $\mathcal{P} \vdash_{\mathrm{SQHL}(\mathcal{R},\mathcal{D})} B'_i\,\sharp\,d_i$ with a proof tree having less nodes that T. Therefore, $S_\mathcal{R}(\mathcal{P}) \vdash_{\mathrm{QHL}(\mathcal{D})} B'_i\,\sharp\,d_i$ by induction hypothesis.

In order to perform the inference step (2), the QMP($\mathcal{D}$) inference rule also requires that $d \sqsubseteq e \circ \bigsqcap\{w,e_1\ldots,e_m,\,d_1,\ldots,d_k\}$. This follows from the associativity of $\sqcap$ since:

- As defined above, $\delta = \mathcal{R}(H_\ell,H') \sqcap \bigsqcap\{\mathcal{R}(X_i\theta,X_j\theta) \mid (X_i \sim X_j) \in \mathcal{S}_\ell\}$, i.e. $\delta = w \sqcap \bigsqcap\{e_1\ldots e_m\}$.
- By the SQMP($\mathcal{R},\mathcal{D}$) inference (1) we know that $d \sqsubseteq e \circ \bigsqcap\{\delta,d_1,\ldots,d_k\}$.

Let T be a $QHL(\mathcal{D})$ proof tree witnessing $S_\mathcal{R}(\mathcal{P}) \vdash_{\mathrm{QHL}(\mathcal{D})} A\,\sharp\,d$ for some atom A in $\mathcal{P}$'s signature. We prove by induction on the number of nodes of T that $\mathcal{P} \vdash_{\mathrm{SQHL}(\mathcal{R},\mathcal{D})} A\,\sharp\,d$.

Since A is in $\mathcal{P}$'s signature, the clause employed at the inference step at the root of T must be in the set $\mathcal{P}_S$ of Definition 6, and the inference step at the root of T have of the form of the inference (2) above. Hence this clause must have been constructed from a clause $C \equiv (H \leftarrow e - B_1,\ldots,B_k) \in \mathcal{P}$ and some atom H' such that $A = H'\theta$ and $\mathcal{R}(H_\ell,H') \neq \bot$, where $lin(H) = (H_\ell,S_\ell)$.

Then we can use C and θ to prove $\mathcal{P} \vdash_{\mathrm{SQHL}(\mathcal{R},\mathcal{D})} A\,\sharp\,d$ by a SQMP($\mathcal{R},\mathcal{D}$) inference like (1) using the $\mathcal{R}$-instance $C' \equiv A \leftarrow e - B'_1,\ldots,B'_k$ of C. The premises can be proved in $SQHL(\mathcal{R},\mathcal{D})$ by induction hypotheses, since all of them are also premises in (2). Finally, we must check that the conditions required by (1) hold: $(C',\delta) \in [C]_\mathcal{R}$ for some $\delta \in \mathcal{D}$, $\delta \neq \bot$ s.t. $d \sqsubseteq e \circ \bigsqcap\{\delta,d_1,\ldots,d_k\}$. This is true for $\delta = \bigsqcap\{w,e'_1,\ldots,e'_n\}$, with $e'_i = \mathcal{R}(u_i\theta,v_i\theta)$ for $i = 1\ldots m$. Observe that in the premises of (2) we have $QHL(\mathcal{D})$ proofs of $u_i\theta \sim v_i\theta\,\sharp\,e_i$ for $i = 1\ldots m$. Therefore $e_i \sqsubseteq e'_i$, by Lemma 1, item 1. Then

$$\begin{array}{lll} d & \sqsubseteq e \circ \bigsqcap\{w,e_1\ldots,e_m,\,d_1,\ldots,d_k\} & \text{(by(2))} \\ & \sqsubseteq e \circ \bigsqcap\{w,e'_1\ldots,e'_m,\,d_1,\ldots,d_k\} & (e_i \sqsubseteq e'_i) \\ & = e \circ \bigsqcap\{\delta,d_1,\ldots,d_k\} & \end{array}$$

We must still prove that $\delta \neq \bot$. Observe that by the distributivity of $\circ$ w.r.t. $\sqcap$ (Def. 1, axiom 2.(e)):

$$e \circ \bigsqcap\{\delta,d_1,\ldots,d_k\} = (e \circ \delta) \sqcap (e \circ \bigsqcap\{d_1,\ldots,d_k\})\ .$$

Therefore

$$d \sqsubseteq (e \circ \delta) \sqcap (e \circ \bigsqcap\{d_1,\ldots,d_k\})$$

and from $d \neq \bot$ we obtain $(e \circ \delta) \neq \bot$ which implies $\delta \neq \bot$ due to axiom 2.(c) in Definition 1. This completes the proof. □

4.2 Comparison to Related Approaches

Other program transformations have been proposed in the literature with the aim of supporting $\mathcal{R}$-based reasoning while avoiding explicit $\mathcal{R}$-based unification. Here we draw some comparisons between the program transformation $S_\mathcal{R}(\mathcal{P})$ presented in the previous subsection, the program transformations $H_\lambda(\mathcal{P})$ and $\mathcal{P}_\lambda$ proposed in [16], and the program transformation $\mathcal{P}_{E,\mathcal{R}}$ proposed in [11]. These three transformations are applied to a classical logic program $\mathcal{P}$ w.r.t. a fuzzy similarity relation $\mathcal{R}$ over symbols in the program's signature. Both $H_\lambda(\mathcal{P})$ and $\mathcal{P}_\lambda$ are classical logic programs to be executed by SLD resolution, and their construction depends on a fixed similarity degree $\lambda \in (0,1]$. On the other hand, $\mathcal{P}_{E,\mathcal{R}}$ is a multi-adjoint logic program over a particular multi-adjoint lattice $\mathcal{G}$, providing the uncertain truth values in the interval $[0,1]$ and two operators for conjunction and disjunction in the sense of Gödel's fuzzy logic (see [22] for technical details). As in the case of our own transformation $S_\mathcal{R}(\mathcal{P})$, the construction of $\mathcal{P}_{E,\mathcal{R}}$ does not depend on any fixed similarity degree. The transformation $S_\mathcal{R}(\mathcal{P})$ proposed in this paper is more general in that it can be applied to an arbitrary $SQLP(\mathcal{R},\mathcal{D})$ program $\mathcal{P}$, yielding a $QLP(\mathcal{D})$ program $S_\mathcal{R}(\mathcal{P})$ whose least Herbrand model is the same as that of $\mathcal{P}$.

We will restrict our comparisons to the case that $\mathcal{P}$ is chosen as a similarity-based logic program in the sense of [16]. As an illustrative example, consider the simple logic program $\mathcal{P}$ consisting of the following four clauses:

- $C_r:\ r(X,Y) \leftarrow p(X),\,q(Y),\,s(X,Y)$
- $C_p:\ p(c(U)) \leftarrow$
- $C_q:\ q(d(V)) \leftarrow$
- $C_s:\ s(Z,Z) \leftarrow$

Assume an admissible similarity relation defined by $\mathcal{R}(c,d) = 0.9$ and consider the goal $G : \leftarrow r(X,Y)$ for $\mathcal{P}$. Then, $\mathcal{R}$-based SLD-resolution as defined in [16] computes the answer substitution $\sigma = \{X \mapsto c(U),\,Y \mapsto d(U)\}$ with similarity degree 0.9. This computation succeeds because $\mathcal{R}$-based unification can compute the $m.g.u.$ $\{Z \mapsto c(U),\,V \mapsto U\}$ with similarity degree 0.9 to unify the two atoms $s(c(U),d(V))$ and $s(Z,Z)$. Let us now examine the behavior of the the transformed programs $H_{0.9}(\mathcal{P})$, $\mathcal{P}_{0.9}$, $S_\mathcal{R}(\mathcal{P})$ and $\mathcal{P}_{E,\mathcal{R}}$ and when working to emulate this computation without explicit use of a $\mathcal{R}$-based unification procedure.

1. $H_{0.9}(\mathcal{P})$ is defined in [16] as the set of all clauses C' such that $\mathcal{R}(C,C') \ge 0.9$ for some clause $C \in \mathcal{P}$. In this case $H_{0.9}(\mathcal{P})$ includes the four clauses of $\mathcal{P}$ and the two additional clauses $p(d(U)) \leftarrow$ and $q(c(V)) \leftarrow$, derived by similarity from C_p and C_q, respectively. Solving G w.r.t. $H_{0.9}(\mathcal{P})$ by means of

classical SLD resolution produces two possible answer substitutions, namely $\sigma_1 = \{X \mapsto c(U), Y \mapsto c(U)\}$ and $\sigma_2 = \{X \mapsto d(U), Y \mapsto d(U)\}$. They are both similar to σ to a degree greater or equal than 0.9, but none of them is σ itself, contrary to the claim in Proposition 7.1 (i) from [16]. Therefore, this Proposition seems to hold only in a somewhat weaker sense than the statement in [16]. This problem is due to the possible non-linearity of a clause's head, which is properly taken into account by our transformation $S_{\mathcal{R}}(\mathcal{P})$.

2. According to [16], $\mathcal{P}_{0.9}$ is computed from $\mathcal{P}$ by replacing all the constructor and predicate symbols by new symbols that represent the equivalence classes of the original ones modulo $\mathcal{R}$-similarity to a degree greater or equal than 0.9. In our example these classes are $\{r\}$, $\{p\}$, $\{q\}$, $\{s\}$ and $\{c, d\}$, that can be represented by the symbols r, p, q, s an e, respectively. Then, $\mathcal{P}_{0.9}$ replaces the two clauses C_p and C_q by $p(e(U)) \leftarrow$ and $q(e(V)) \leftarrow$, respectively, leaving the other two clauses unchanged. Solving G w.r.t. $\mathcal{P}_{0.9}$ by means of classical SLD resolution produces the answer substitution $\sigma' = \{X \mapsto e(U), Y \mapsto e(U)\}$, which corresponds to σ modulo the replacement of the symbols in the original program by their equivalence classes. This is consistent with the claims in Proposition 7.2 from [16].

3. Note that $\mathcal{P}$ can be trivially converted into a semantically equivalent a $SQLP(\mathcal{R},\mathcal{U})$ program, just by replacing each occurrence of the implication sign $\leftarrow$ in $\mathcal{P}$'s clauses by $\leftarrow 1.0-$. Then $S_{\mathcal{R}}(\mathcal{P})$ can be built as a $QLP(\mathcal{U})$ program by the method explained in Subsection 4.1. It includes three clauses corresponding to C_r, C_p and C_q of $\mathcal{P}$ plus the following three new clauses:

   - $C'_p:\ p(d(U)) \leftarrow 1.0 - pay_{0.9}$
   - $C'_q:\ q(c(V)) \leftarrow 1.0 - pay_{0.9}$
   - $C'_s:\ s(Z_1, Z_2) \leftarrow 1.0 - Z_1 \sim Z_2$

   where C'_p resp. C'_q come from replacing the linear heads of C_p resp. C_q by similar heads, and C_s comes from linearizing the head of C_s, which allows no replacements by similarity. $S_{\mathcal{R}}(\mathcal{P})$ includes also the proper clauses for $\mathcal{P}_{\sim}$ and $\mathcal{P}_{\mathrm{pay}}$, in particular the following three ones:

   - $I:\ X \sim X \leftarrow 1.0-$
   - $S:\ c(X_1) \sim d(Y_1) \leftarrow 1.0 - pay_{0.9}, X_1 \sim Y_1$
   - $P:\ pay_{0.9} \leftarrow 0.9-$

   Solving goal G w.r.t. $S_{\mathcal{R}}(\mathcal{P})$ by means of the $\mathcal{U}$-qualified SLD resolution procedure described in [14] can compute the answer substitution σ with qualification degree 0.9. More precisely, the initial goal can be stated as $r(X,Y)\#W \,[\!]\, W \geq 0.9$, and the computed answer is $(\sigma, \{W \mapsto 0.9\})$. The computation emulates $\mathcal{R}$-based unification of $s(c(U), c(V))$ and $s(Z, Z)$ to the similarity degree 0.9 by solving $s(c(U), c(V))$ with the clauses C'_s, I, S and P.

4. The semantics of the *MALP* framework depending on the chosen multi-adjoint lattice is presented in [11]. A comparison with the semantics of the $QLP(\mathcal{D})$ scheme (see [14] and Subsection 3.1 above) shows that *MALP* programs over the multi-adjoint lattice $\mathcal{G}$ behave as $QLP(\mathcal{U}')$ programs, where $\mathcal{U}'$ is the quasi qualification domain analogous to $\mathcal{U}$ introduced at the end of Subsection 2.1 above. For this reason, we can think of the transformed program $\mathcal{P}_{E,\mathcal{R}}$ as presented with he syntax of a $QLP(\mathcal{U}')$ program. The original program $\mathcal{P}$ can also be written as a $QLP(\mathcal{U}')$ program just by replacing each the implication sign $\leftarrow$ occurring in $\mathcal{P}$ by $\leftarrow 1.0-$. As explained in [11], $\mathcal{P}_{E,\mathcal{R}}$ is built by extending $\mathcal{P}$ with clauses for a new binary predicate $\sim$ intended to emulate the behaviour of $\mathcal{R}$-based unification between terms. In our example, $\mathcal{P}_{E,\mathcal{R}}$ will include (among others) the following clause for $\sim$:

   - $S':\ c(X_1) \sim d(Y_1) \leftarrow 0.9 - X_1 \sim Y_1$

   In comparison to the clause S in $S_{\mathcal{R}}(\mathcal{P})$, clause S' needs no call to a $pay_{0.9}$ predicate at its body, because the similarity degree $0.9 = \mathcal{R}(c, d)$ can be attached directly to the clause's implication. This difference corresponds to the different interpretations of $\circ$, which behaves as $\times$ in $\mathcal{U}$ and as min in $\mathcal{U}'$.

   Moreover, $\mathcal{P}_{E,\mathcal{R}}$ is defined to include a clause of the following form for each pair of n-ary predicate symbols pd and pd' such that $\mathcal{R}(pd, pd') \neq 0$:

   - $C_{pd,pd'}:\ pd(Y_1, \ldots, Y_n) \leftarrow \mathcal{R}(pd, pd') - pd'(X_1, \ldots, X_n), X_1 \sim Y_1, \ldots, X_n \sim Y_n$

   In our simple example, all the clauses of this form correspond to the trivial case where pd and pd' are the same predicate symbol and $\mathcal{R}(pd, pd') = 1.0$. Solving goal G w.r.t.$S_{\mathcal{R}}(\mathcal{P})$ by means of the procedural semantics described in Section 4 of [11] can compute the answer substitution σ to the similarity degree 0.9. More generally, Theorem 24 in [11] claims that for any choice of $\mathcal{P}$, $\mathcal{P}_{E,\mathcal{R}}$ can emulate any successful computation performed by $\mathcal{P}$ using $\mathcal{R}$-based SLD resolution.

In conclusion, the main difference between $S_{\mathcal{R}}(\mathcal{P})$ and $\mathcal{P}_{E,\mathcal{R}}$ pertains to the techniques used by both program transformations in order to emulate the effect of replacing the head of a clause in the original program by a similar one. $\mathcal{P}_{E,\mathcal{R}}$ always relies on the clauses of the form $C_{pd,pd'}$ *and* the clauses for $\sim$, while $S_{\mathcal{R}}(\mathcal{P})$ can avoid to use the clauses for $\sim$ as long as all the clauses involved in the computation have linear heads. In comparison to the two transformations $H_\lambda(\mathcal{P})$ and $\mathcal{P}_\lambda$, our transformation $S_{\mathcal{R}}(\mathcal{P})$ does not depend on a fixed similarity degree λ and does not replace the atoms in clause bodies by similar ones.

4.3 A Goal Solving Example

In order to illustrate the use of the transformed program $S_{\mathcal{R}}(\mathcal{P})$ for golving goals w.r.t. the original program $\mathcal{P}$, we consider the case where $\mathcal{P}$ is the $SQLP(\mathcal{R},\mathcal{U})$ program displayed in Figure 1. The transformed program $S_{\mathcal{R}}(\mathcal{P})$ obtained by applying Definition 6 is shown in Figure 2. The following observations are useful to understand how the transformation has worked in this simple case:

- The value $\top$ in the domain $\mathcal{U}$ corresponds to the real number 1 and hence by reflexivity $\mathcal{R}(A, A) = 1$ for any atom in the signature of the program. Therefore, and as a consequence of Definition 6, every clause in the original program gives rise to a clause in the transformed program with the same head and with the same body except for a new, first atom $pay_{1.0}$. For instance, clauses 1, 2 and 3 in Figure 2 correspond to the same clause numbers in Figure 1.
- Apart of the clauses corresponding directly to the original clauses, the program of Figure 2 contains new clauses obtained by similarity with some clause heads in the original program. For instance, lines 4 and 5 are obtained by similarity with clauses at lines 1 and 2 in the original program, respectively. The subindexes at literal pay correspond to $\mathcal{R}(\texttt{lynx}, \texttt{cat}) = 0.8$, $\mathcal{R}(\texttt{boar}, \texttt{pig}) = 0.7$, respectively.
- Analogously, for instance the clause at line 10 (with head `farm(lynx)`) is obtained by head-similarity with the clause of line 6 in the $SQLP(\mathcal{R},\mathcal{U})$ program (head `domestic(cat)`),

```
1  wild(lynx) <-0.9- pay_1.0
2  wild(boar) <-0.9- pay_1.0
3  wild(snake) <-1.0- pay_1.0
4  wild(cat) <-0.9- pay_0.8
5  wild(pig) <-0.9- pay_0.7

6  farm(cow) <-1.0- pay_1.0
7  farm(pig) <-1.0- pay_1.0
8  farm(boar) <-1.0- pay_0.7
9  farm(cat) <-0.8- pay_0.3
10 farm(lynx) <-0.8- pay_0.3
11 farm(snake) <-0.4- pay_0.3

12 domestic(cat) <-0.8- pay_1.0
13 domestic(snake) <-0.4- pay_1.0
14 domestic(lynx) <-0.8- pay_0.8
15 domestic(cow) <-1.0- pay_0.3
16 domestic(pig) <-1.0- pay_0.3
17 domestic(boar) <-1.0- pay_0.3

18 intelligent(A) <-0.9- pay_1.0,domestic(A)
19 intelligent(lynx) <-0.7- pay_1.0
20 intelligent(cat) <-0.7- pay_0.8

21 pacific(A) <-0.9- pay_1.0,domestic(A)
22 pacific(A) <-0.7- pay_1.0,farm(A)

23 pet(A) <-1.0- pay_1.0,pacific(A),intelligent(A)

24 pay_1.0 <-1.0-
25 pay_0.8 <-0.8-
26 pay_0.7 <-0.7-
27 pay_0.3 <-0.3-
```

Figure 2. Example of transformed program. (Note: no clauses for $\sim$ are needed because the original program was left-linear).

and the subindex at pay is obtained from

$$\begin{array}{ll} \mathcal{R}(\texttt{domestic(cat)},\texttt{farm(lynx)}) & = \\ \mathcal{R}(\texttt{domestic},\texttt{farm}) \sqcap \mathcal{R}(\texttt{cat},\texttt{lynx}) & = \\ 0.3 \sqcap 0.8 & = \\ 0.3 & \end{array}$$

- There is no clause for predicate $\sim$ since all the heads in the original program were already linear and therefore $\mathcal{P}_\sim$ can be left empty in practice.
- The clauses for pay correspond to the fragment $\mathcal{P}_{\text{pay}}$ in Definition 6.

In the rest of this subsection, we will show an execution for the goal `pet(A)#W | W >= 0.50` over the program $S_\mathcal{R}(\mathcal{P})$ (see Figure 2) with the aim of obtaining all those animals that could be considered a `pet` for at least a qualification value of 0.50.

We are trying this execution in the prototype developed along with [14] for the instances $QLP(\mathcal{U})$ and $QLP(\mathcal{W})$. Although this prototype hasn't been released as an integrated part of $\mathcal{TOY}$, you can download[1] the prototype to try this execution. Please notice that the prototype does not automatically do the translation process from a given $SQLP(\mathcal{R},\mathcal{D})$ program $\mathcal{P}$ to its transformed program $S_\mathcal{R}(\mathcal{P})$, because it was developed mainly for [14]. Therefore, the transformed program shown in Figure 2 has been computed manually.

We will start running $\mathcal{TOY}$ and loading the $QLP(\mathcal{U})$ instance with the command `/qlp(u)`:

```
Toy> /qlp(u)
```

this will have the effect of loading the *Real Domain Constraints library* and the $QLP(\mathcal{U})$ library into the system, the prompt `QLP(U)>` will appear. Now we have to compile our example program (assume we have it in a text file called `animals.qlp` in `C:/examples/`) with the command `/qlptotoy` (this command will behave differently based on the actual instance loaded).

```
QLP(U)> /qlptotoy(c:/examples/animals)
```

Note that we didn't write the extension of the file because it *must* be `.qlp`. This will create the file `animals.toy` in the same directory as our former file. And this one will be an actual $\mathcal{TOY}$ program. We run the program with `/run(c:/examples/animals)` (again without the extension –although this time we are assuming `.toy` as extension–) and we should get the following message:

```
PROCESS COMPLETE
```

And finally we are set to launch our goal with the command `/qlpgoal`. The solutions found for this program and goal are:

```
QLP(U)> /qlpgoal(pet(A)#W | W>=0.50)
      { A -> cat,
        W -> 0.5599999999999999 }

sol.1, more solutions (y/n/d/a) [y]?
      { A -> cat,
        W -> 0.7200000000000001 }

sol.2, more solutions (y/n/d/a) [y]?
      { A -> lynx,
        W -> 0.5760000000000002 }

sol.3, more solutions (y/n/d/a) [y]?
      { A -> lynx,
        W -> 0.5760000000000002 }

sol.4, more solutions (y/n/d/a) [y]?
      no
```

At this point and if you remember the inference we did in Example 2 for `pet(lynx)#0.50`, we have found a better solution (as you can see there are two solutions for `lynx`, and this is due to the two different ways of proving `intelligent(lynx)`: `intelligent(lynx)#0.7` using clause 19, and `intelligent (lynx)#0.576` using clauses 18 and 14.

5. Conclusions

Similarity-based LP has been proposed in [16] and related works to enhance the LP paradigm with a kind of approximate reasoning which supports flexible information retrieval applications, as argued in [8, 11]. This approach keeps the syntax for program clauses as in classical LP, and supports uncertain reasoning by using a fuzzy similarity relation $\mathcal{R}$ between symbols in the program's signature. We have shown that similarity-based LP as presented in [16] can be reduced to Qualified LP in the $QLP(\mathcal{D})$ scheme introduced in [14], which supports logic programming with attenuated program clauses over a parametrically given domain $\mathcal{D}$ whose

[1] Available at: `http://gpd.sip.ucm.es/cromdia/qlpd`. There you will also find specific instructions on how to install and run it as well as text files with the program examples tried in here.

elements qualify logical assertions by measuring their closeness to various users' expectations. Using generalized similarity relations taking values in the carrier set of an arbitrarily given qualification domain $\mathcal{D}$, we have extended $QLP(\mathcal{D})$ to a more expressive scheme $SQLP(\mathcal{R},\mathcal{D})$ with two parameters, for programming modulo $\mathcal{R}$-similarity with $\mathcal{D}$-attenuated Horn clauses. We have presented a declarative semantics for $SQLP(\mathcal{R},\mathcal{D})$ programs and a semantics-preserving program transformation which embeds $SQLP(\mathcal{R},\mathcal{D})$ into $QLP(\mathcal{D})$. As a consequence, the sound and complete procedure for solving goals in $QLP(\mathcal{D})$ by $\mathcal{D}$-qualified SLD resolution and its implementation in the $\mathcal{TOY}$ system [14] can be used to implement $SQLP(\mathcal{R},\mathcal{D})$ computations via the transformation.

Our framework is quite general due to the availability of different qualification domains, while the similarity relations proposed in [16] take fuzzy values in the interval $[0,1]$. In comparison to the multi-adjoint framework proposed in [11], the $QLP(\mathcal{D})$ and $SQLP(\mathcal{R},\mathcal{D})$ schemes have a different motivation and scope, due to the differences between multi-adjoint algebras and qualification domains as algebraic structures. In contrast to the goal solving procedure used in the multi-adjoint framework, $\mathcal{D}$-qualified SLD resolution does not rely on costly computations of reductant clauses and has been efficiently implemented.

As future work, we plan to investigate an extension of the $\mathcal{R}$-based SLD resolution procedure proposed in [16] to be used within the $SQLP(\mathcal{R},\mathcal{D})$ scheme, and to develop an extension of this scheme which supports lazy functional programming and constraint programming facilities. The idea of similarity-based unification has been already applied in [12] to obtain an extension of *needed narrowing*, the main goal solving procedure of functional logic languages. As in the case of [16], the similarity relations considered in [12] take fuzzy values in the real interval $[0,1]$.

Acknowledgments

The authors have been partially supported by the Spanish National Projects MERIT-FORMS (TIN2005-09027-C03-03) and PROMESAS–CAM (S-0505/TIC/0407).

References

[1] K.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] K.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.

[3] 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.

[4] 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 2.3.1, 2007. R. Caballero and J. Sánchez (Eds.), available at `http://toy.sourceforge.net`.

[5] F. Formato, G. Gerla and M.I. Sessa. Similarity-based unification. *Fundamenta Informaticae*, 41(4):393-414, 2000.

[6] G. Gerla and M.I. Sessa. Similarity in logic programming. In G. Chen, M. Ying and K. Cai, editors, *Fuzzy Logic and Soft Computing*, pages 19-31. Kluwer Academic Publishers, 1999.

[7] 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.

[8] 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.

[9] J. Medina, M. Ojeda-Aciego and P. Vojtáš. Multi-adjoint logic programming with continuous semantics. In T. Eiter, W. Faber and M. Truszczyinski, editors, *Logic Programming and Non-Monotonic Reasoning (LPNMR'01)*, volume 2173 of *LNAI*, pages 351-364. Springer-Verlag, 2001.

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

[11] J. Medina, M. Ojeda-Aciego and P. Vojtáš. Similarity-based unification: A multi-adjoint approach. *Fuzzy Sets and Systems*, 146:43-62, 2004.

[12] G. Moreno and V. Pascual. Programming with fuzzy logic and mathematical functions. In A.P.I. Bloch and A. Tettamanzi, editors, *Proceedings of the 6th International Workshop on Fuzzy Logic and Applications (WILF'05)*, volume 3849 of *LNAI*, pages 89-98. Springer-Verlag, 2006.

[13] M. Rodríguez-Artalejo and C.A. Romero-Díaz. A generic scheme for qualified logic programming (Technical Report SIC-1-08). Technical Report, Universidad Complutense, Departamento de Sistemas Informáticos y Computación, Madrid, Spain, 2008.

[14] M. Rodríguez-Artalejo and C.A. Romero-Díaz. Quantitative logic programming revisited. In J. Garrigue and M. Hermenegildo, editors, *Functional and Logic Programming (FLOPS'08)*, volume 4989 of *LNCS*, pages 272-288. Springer-Verlag, 2008.

[15] M.I. Sessa. Translations and similarity-based logic programming. *Soft Computing*, 5(2), 2001.

[16] M.I. Sessa. Approximate reasoning by similarity-based SLD resolution. *Theoretical Computer Science*, 275(1&2):389-426, 2002.

[17] 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.

[18] 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.

[19] V.S. Subrahmanian. Uncertainty in logic programming: Some recollections. *Association for Logic Programming Newsletter*, 20(2), 2007.

[20] M.H. van Emden. Quantitative deduction and its fixpoint theory. *Journal of Logic Programming*, 3(1):37-53, 1986.

[21] 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.

[22] P. Vojtáš. Fuzzy logic programming. *Fuzzy Sets and Systems*, 124:361:370, 2001.

A Generic Scheme for Qualified Constraint Functional Logic Programming*

Technical Report SIC-1-09

Rafael Caballero, Mario Rodríguez-Artalejo and Carlos A. Romero-Díaz

Departamento de Sistemas Informáticos y Computación, Universidad Complutense, Facultad de Informática, 28040 Madrid, Spain
{rafa,mario}@sip.ucm.es and cromdia@fdi.ucm.es

Abstract. Qualification has been recently introduced as a generalization of uncertainty in the field of Logic Programming. In this report we investigate a more expressive language for First-Order Functional Logic Programming with Constraints and Qualification. We present a Rewriting Logic which characterizes the intended semantics of programs, and a prototype implementation based on a semantically correct program transformation. Potential applications of the resulting language include flexible information retrieval. As a concrete illustration, we show how to write program rules to compute qualified answers for user queries concerning the books available in a given library.

Keywords: Constraints, Functional Logic Programming, Program Transformation, Qualification, Rewriting Logic.

1 Introduction

Various extensions of Logic Programming with uncertain reasoning capabilities have been widely investigated during the last 25 years. The recent recollection [21] reviews the evolution of the subject from the viewpoint of a committed researcher. All the proposals in the field replace classical two-valued logic by some kind of many-valued logic with more than two truth values, which are attached to computed answers and interpreted as truth degrees.

In a recent work [19,18] we have presented a *Qualified Logic Programming* scheme QLP($\mathcal{D}$) parameterized by a *qualification domain* $\mathcal{D}$, a lattice of so-called *qualification values* that are attached to computed answers and interpreted as a measure of the satisfaction of certain user expectations. QLP($\mathcal{D}$)-programs are sets of clauses of the form $A \stackrel{\alpha}{\leftarrow} \overline{B}$, where the head A is an atom, the body $\overline{B}$ is a conjunction of atoms, and $\alpha \in \mathcal{D}$ is called *attenuation* factor. Intuitively, α measures the maximum confidence placed on an inference performed by the clause. More precisely, any successful application of the clause attaches to the

* Research partially supported by projects MERIT–FORMS (TIN2005-09027-C03-03), PROMESAS–CAM(S-0505/TIC/0407) and STAMP (TIN2008-06622-C03-01).

head a qualification value which cannot exceed the infimum of $\alpha \circ \beta_i \in \mathcal{D}$, where β_i are the qualification values computed for the body atoms and $\circ$ is a so-called *attenuation operator*, provided by $\mathcal{D}$.

Uncertain Logic Programming can be expressed by particular instances of QLP($\mathcal{D}$), where the user expectation is understood as a lower bound for the *truth degree* of the computed answer and $\mathcal{D}$ is chosen to formalize a lattice of non-classical truth values. Other choices of $\mathcal{D}$ can be designed to model other kinds of user expectations, as e.g. an upper bound for the *size* of the logical proof underlying the computed answer. As shown in [4], the QLP($\mathcal{D}$) scheme is also well suited to deal with Uncertain Logic Programming based on similarity relations in the line of [20]. Therefore, Qualified Logic Programming has a potential for flexible information retrieval applications, where the answers computed for user queries may match the user expectations only to some degree. As shown in [19], several useful instances of QLP($\mathcal{D}$) can be conveniently implemented by using constraint solving techniques.

In this report we investigate an extension of QLP($\mathcal{D}$) to a more expressive scheme, supporting computation with first-order lazy functions and constraints. More precisely, we consider the first-order fragment of CFLP($\mathcal{C}$), a generic scheme for functional logic programming with constraints over a parametrically given domain $\mathcal{C}$ presented in [13]. We propose an extended scheme QCFLP($\mathcal{D},\mathcal{C}$) where the additional parameter $\mathcal{D}$ stands for a qualification domain. QCFLP($\mathcal{D},\mathcal{C}$)-programs are sets of conditional rewrite rules of the form $f(\bar{t}_n) \xrightarrow{\alpha} r \Leftarrow \Delta$, where the condition Δ is a conjunction of $\mathcal{C}$-constraints that may involve user defined functions, and $\alpha \in \mathcal{D}$ is an attenuation factor. As in the logic programming case, α measures the maximum confidence placed on an inference performed by the rule: any successful application of the rule attaches to the computed result a qualification value which cannot exceed the infimum of $\alpha \circ \beta_i \in \mathcal{D}$, where β_i are the qualification values computed for r and Δ, and $\circ$ is $\mathcal{D}$'s attenuation operator. QLP($\mathcal{D}$) program clauses can be easily formulated as a particular case of QCFLP($\mathcal{D},\mathcal{C}$) program rules.

As far as we know, no related work covers the expressivity of our approach. Guadarrama et al. [8] have proposed to use real arithmetic constraints as an implementation tool for a Fuzzy Prolog, but their language does not support constraint programming as such. Starting from the field of natural language processing, Riezler [15,16] has developed quantitative and probabilistic extensions of the classical CLP($\mathcal{C}$) scheme with the aim of computing good parse trees for constraint logic grammars, but his work bears no relation to functional programming. Moreno and Pascual [14] have investigated similarity-based unification in the context of *needed narrowing* [1], a narrowing strategy using so-called *definitional trees* that underlies the operational semantics of functional logic languages such as Curry [9] and $\mathcal{TOY}$ [3], but they use neither constraints nor attenuation factors and they provide no declarative semantics. The approach of the present report is quite different. We work with a class of programs more general and expressive than the *inductively sequential* term rewrite systems used in [14], and our results focus on a rewriting logic used to characterize declarative semantics

and to prove the correctness of an implementation technique based on a program transformation. Similarity relations could be easily incorporated to our scheme by using the techniques presented in [4] for the Logic Programming case. Moreover, the good properties of needed narrowing as a computation model are not spoiled by our implementation technique, because our program transformation preserves the structure of the definitional trees derived from the user-given program rules.

```
%% Data types:
type pages, id = int
type title, author, language, genre = [char]
data vocabularyLevel = easy | medium | difficult
data readerLevel = basic | intermediate | upper | proficiency
data book = book(id, title, author, language, genre, vocabularyLevel, pages)

%% Simple library, represented as list of books:
library :: [book]
library --> [ book(1, "Tintin", "Herge", "French", "Comic", easy, 65),
              book(2, "Dune", "F. P. Herbert", "English", "SciFi", medium, 345),
              book(3, "Kritik der reinen Vernunft", "Immanuel Kant", "German",
                   "Philosophy", difficult, 1011),
              book(4, "Beim Hauten der Zwiebel", "Gunter Grass", "German",
                   "Biography", medium, 432) ]

%% Auxiliary function for computing list membership:
member(B,[]) --> false
member(B,H:_T) --> true <== B == H
member(B,H:T) --> member(B,T) <== B /= H

%% Functions for getting the explicit attributes of a given book:
getId(book(Id,_Title,_Author,_Lang,_Genre,_VocLvl,_Pages)) --> Id
getTitle(book(_Id,Title,_Author,_Lang,_Genre,_VocLvl,_Pages)) --> Title
getAuthor(book(_Id,_Title,Author,_Lang,_Genre,_VocLvl,_Pages)) --> Author
getLanguage(book(_Id,_Title,_Author,Lang,_Genre,_VocLvl,_Pages)) --> Lang
getGenre(book(_Id,_Title,_Author,_Lang,Genre,_VocLvl,_Pages)) --> Genre
getVocabularyLevel(book(_Id,_Title,_Author,_Lang,_Genre,VocLvl,_Pages)) --> VocLvl
getPages(book(_Id,_Title,_Author,_Lang,_Genre,_VocLvl,Pages)) --> Pages

%% Function for guessing the genre of a given book:
guessGenre(B) --> getGenre(B)
guessGenre(B) -0.9-> "Fantasy" <== guessGenre(B) == "SciFi"
guessGenre(B) -0.8-> "Essay" <== guessGenre(B) == "Philosophy"
guessGenre(B) -0.7-> "Essay" <== guessGenre(B) == "Biography"
guessGenre(B) -0.7-> "Adventure" <== guessGenre(B) == "Fantasy"

%% Function for guessing the reader level of a given book:
guessReaderLevel(B) --> basic <== getVocabularyLevel(B) == easy, getPages(B) < 50
guessReaderLevel(B) -0.8-> intermediate <== getVocabularyLevel(B) == easy, getPages(B) >= 50
guessReaderLevel(B) -0.9-> basic <== guessGenre(B) == "Children"
guessReaderLevel(B) -0.9-> proficiency <== getVocabularyLevel(B) == difficult,
                                           getPages(B) >= 200
guessReaderLevel(B) -0.8-> upper <== getVocabularyLevel(B) == difficult, getPages(B) < 200
guessReaderLevel(B) -0.8-> intermediate <== getVocabularyLevel(B) == medium
guessReaderLevel(B) -0.7-> upper <== getVocabularyLevel(B) == medium

%% Function for answering a particular kind of user queries:
search(Language,Genre,Level) --> getId(B) <== member(B,library),
                                              getLanguage(B) == Language,
                                              guessReaderLevel(B) == Level,
                                              guessGenre(B) == Genre
```

Fig. 1. Library with books in different languages

Figure 1 shows a small set of QCFLP($\mathcal{U},\mathcal{R}$) program rules, called the *library program* in the rest of the report. The concrete syntax is inspired by the functional logic language $\mathcal{TOY}$, but the ideas and results of this report could be also applied to Curry and other similar languages. In this example, $\mathcal{U}$ stands for a particular qualification domain which supports uncertain truth values in the real interval $[0,1]$, while $\mathcal{R}$ stands for a particular constraint domain which supports arithmetic constraints over the real numbers; see Section 2 for more details.

The program rules are intended to encode expert knowledge for computing qualified answers to user queries concerning the books available in a simplified library, represented as a list of objects of type `book`. The various `get` functions extract the explicit values of book attributes. Functions `guessGenre` and `guessReaderLevel` infer information by performing qualified inferences, relying on analogies between different genres and heuristic rules to estimate reader levels on the basis of other features of a given book, respectively. Some program rules, as e.g. those of the auxiliary function `member`, have attached no explicit attenuation factor. By convention, this is understood as the implicit attachment of the attenuation factor `1.0`, the top value of $\mathcal{U}$. For any instance of the QCFLP($\mathcal{D},\mathcal{C}$) scheme, a similar convention allows to view CFLP($\mathcal{C}$)-program rules as QCFLP($\mathcal{D},\mathcal{C}$)-program rules whose attached qualification is optimal.

The last rule for function `search` encodes a method for computing qualified answers to a particular kind of user queries. Therefore, the queries can be formulated as goals to be solved by the program fragment. For instance, answering the query of a user who wants to find a book of genre `"Essay"`, language `"German"` and user level `intermediate` with a certainty degree of at least 0.65 can be formulated as the goal:

```
(search("German","Essay",intermediate) == R) # W | W >= 0.65
```

The techniques presented in Section 4 can be used to translate the QCFLP($\mathcal{U},\mathcal{R}$) program rules and goal into the CFLP($\mathcal{R}$) language, which is implemented in the $\mathcal{TOY}$ system. Solving the translated goal in $\mathcal{TOY}$ computes the answer $\{R \mapsto 4\}\{0.65 \leq W, W \leq 0.7\}$, ensuring that the library book with `id 4` satisfies the query's requirements with any certainty degree in the interval [0.65,0.7], in particular 0.7. The computation uses the 4th program rule of `guessGenre` to obtain `"Essay"` as the book's genre with qualification 0.7, and the 6th program rule of `guessReaderLevel` to obtain `intermediate` as the reader level with qualification 0.8.

The rest of the report is organized as follows. In Section 2 we recall known proposals concerning qualification and constraint domains, and we introduce a technical notion needed to relate both kinds of domains for the purposes of this report. In Section 3 we present the generic scheme QCFLP($\mathcal{D},\mathcal{C}$) announced in this introduction, and we formalize a special Rewriting Logic which characterizes the declarative semantics of QCFLP($\mathcal{D},\mathcal{C}$)-programs. In Section 4 we present a semantically correct program transformation converting QCFLP($\mathcal{D},\mathcal{C}$) programs and goals into the qualification-free CFLP($\mathcal{C}$) programming scheme, which is supported by existing systems such as $\mathcal{TOY}$. Section 5 concludes and points to some lines of planned future work.

2 Qualification and Constraint Domains

Qualification Domains were introduced in [19]. Their intended use has been already explained in the Introduction. In this section we recall and slightly improve their axiomatic definition.

Definition 1 (Qualification Domains). *A* Qualification Domain *is any structure* $\mathcal{D} = \langle D, \preccurlyeq, \mathbf{b}, \mathbf{t}, \circ \rangle$ *verifying the following requirements:*

1. *D, noted as $D_{\mathcal{D}}$ when convenient, is a set of elements called* qualification values.
2. *$\langle D, \preccurlyeq, \mathbf{b}, \mathbf{t} \rangle$ is a lattice with extreme points $\mathbf{b}$ and $\mathbf{t}$ w.r.t. the partial ordering $\preccurlyeq$. For given elements $d, e \in D$, we write $d \sqcap e$ for the* greatest lower bound *(glb) of d and e, and $d \sqcup e$ for the* least upper bound *(lub) of d and e. We also write $d \lhd e$ as abbreviation for $d \preccurlyeq e \wedge d \neq e$.*
3. *$\circ : D \times D \longrightarrow D$, called* attenuation operation*, verifies the following axioms:*
   (a) $\circ$ is associative, commutative and monotonic w.r.t. $\preccurlyeq$.
   (b) $\forall d \in D : d \circ \mathbf{t} = d$.
   (c) $\forall d, e \in D \setminus \{\mathbf{b}, \mathbf{t}\} : d \circ e \lhd e$.
   (d) $\forall d, e_1, e_2 \in D : d \circ (e_1 \sqcap e_2) = d \circ e_1 \sqcap d \circ e_2$. □

As an easy consequence of the previous definition one can prove the following proposition. [1]

Proposition 1 (Additional properties of qualification domains). *Any qualification domain $\mathcal{D}$ satisfies the following properties:*

1. *$\forall d, e \in D : d \circ e \preccurlyeq e$.*
2. *$\forall d \in D : d \circ \mathbf{b} = \mathbf{b}$.*

Proof. Since $\mathbf{t}$ is the top element of the lattice, we know $d \preccurlyeq \mathbf{t}$ for any $d \in D$. As $\circ$ is monotonic w.r.t. $\preccurlyeq$, $d \circ e \preccurlyeq \mathbf{t} \circ e$ also holds for any $e \in D$ which, due to commutativity and axiom (b) of $\circ$, yields $d \circ e \preccurlyeq e$. Therefore *1.* holds. Now, taking $e = \mathbf{b}$, one has $d \circ \mathbf{b} \preccurlyeq \mathbf{b}$ which implies $d \circ \mathbf{b} = \mathbf{b}$ as $\mathbf{b}$ is the bottom element of the lattice. Hence *2.* also holds. □

The examples in this report will use a particular qualification domain $\mathcal{U}$ whose values represent certainty degrees in the sense of fuzzy logic. Formally, $\mathcal{U} = \langle U, \leq, 0, 1, \times \rangle$, where $U = [0, 1] = \{d \in \mathbb{R} \mid 0 \leq d \leq 1\}$, $\leq$ is the usual numerical ordering, and $\times$ is the multiplication operation. In this domain, the bottom and top elements are $\mathbf{b} = 0$ and $\mathbf{t} = 1$, and the infimum of a finite $S \subseteq U$ is the minimum value $\min(S)$, understood as 1 if $S = \emptyset$. The class of qualification domains is closed under cartesian products. For a proof of this fact and other examples of qualification domains, the reader is referred to [19,18].

Constraint domains are used in Constraint Logic Programming and its extensions as a tool to provide data values, primitive operations and constraints

[1] The authors are thankful to G. Gerla for pointing out this fact.

tailored to domain-oriented applications. Various formalizations of this notion are known. In this report, constraint domains are related to signatures of the form $\Sigma = \langle DC, PF, DF \rangle$ where $DC = \bigcup_{n\in\mathbb{N}} DC^n$, $PF = \bigcup_{n\in\mathbb{N}} PF^n$ and $DF = \bigcup_{n\in\mathbb{N}} DF^n$ are mutually disjoint sets of *data constructor* symbols, *primitive function* symbols and *defined function* symbols, respectively, ranked by arities. Given a signature Σ, a symbol $\bot$ to note the *undefined value*, a set B of *basic values* u and a countably infinite set $\mathcal{V}ar$ of variables X, we define the notions listed below, where $\overline{o}_n$ abbreviates the n-tuple of syntactic objects $o_1, \ldots, o_n$.

- *Expressions* $e \in \mathrm{Exp}_\bot(\Sigma, B, \mathcal{V}ar)$ have the syntax $e ::= \bot | X | u | h(\overline{e}_n)$, where $h \in DC^n \cup PF^n \cup DF^n$. In the case $n = 0$, $h(\overline{e}_n)$ is written simply as h.
- *Constructor Terms* $t \in \mathrm{Term}_\bot(\Sigma, B, \mathcal{V}ar)$ have the syntax $e ::= \bot | X | u | c(\overline{t}_n)$, where $c \in DC^n$. They will be called just terms in the sequel.
- *Total Expressions* $e \in \mathrm{Exp}(\Sigma, B, \mathcal{V}ar)$ and *Total Terms* $t \in \mathrm{Term}(\Sigma, B, \mathcal{V}ar)$ have a similar syntax, with the $\bot$ case omitted.
- An expression or term (total or not) is called *ground* iff it includes no occurrences of variables. $\mathrm{Exp}_\bot(\Sigma, B)$ stands for the set of all ground expressions. The notations $\mathrm{Term}_\bot(\Sigma, B)$, $\mathrm{Exp}(\Sigma, B)$ and $\mathrm{Term}(\Sigma, B)$ have a similar meaning.
- We note as $\sqsubseteq$ the *information ordering*, defined as the least partial ordering over $\mathrm{Exp}_\bot(\Sigma, B, \mathcal{V}ar)$ compatible with contexts and verifying $\bot \sqsubseteq e$ for all $e \in \mathrm{Exp}_\bot(\Sigma, B, \mathcal{V}ar)$.
- Substitutions are defined as mappings $\sigma : \mathcal{V}ar \to \mathrm{Term}_\bot(\Sigma, B, \mathcal{V}ar)$ assigning not necessarily total terms to variables. They can be represented as sets of bindings $X \mapsto t$ and extended to act over other syntactic objects o. The *domain* $\mathrm{vdom}(\sigma)$ and *variable range* $\mathrm{vran}(\sigma)$ are defined in the usual way. We will write $o\sigma$ for the result of applying σ to o. The *composition* $\sigma\sigma'$ of two substitutions is such that $o(\sigma\sigma')$ equals $(o\sigma)\sigma'$.

By adapting the definition found in Section 2.2 of [13] to a first-order setting, we obtain: [2]

Definition 2 (Constraint Domains). *A* Constraint Domain *of signature Σ is any algebraic structure of the form $\mathcal{C} = \langle C, \{p^{\mathcal{C}} \mid p \in PF\}\rangle$ such that:*

1. *The carrier set C is $Term_\bot(\Sigma, B)$ for a certain set B of* basic values. *When convenient, we note B and C as $B_{\mathcal{C}}$ and $C_{\mathcal{C}}$, respectively.*
2. *$p^{\mathcal{C}} \subseteq C^n \times C$, written simply as $p^{\mathcal{C}} \subseteq C$ in the case $n = 0$, is called the* interpretation *of p in $\mathcal{C}$. We will write $p^{\mathcal{C}}(\overline{t}_n) \to t$ (or simply $p^{\mathcal{C}} \to t$ if $n = 0$) to indicate that $(\overline{t}_n, t) \in p^{\mathcal{C}}$.*
3. *Each primitive interpretation $p^{\mathcal{C}}$ has* monotonic *and* radical *behavior w.r.t. the information ordering $\sqsubseteq$. More precisely:*
   (a) **Monotonicity***: For all $p \in PF^n$, $p^{\mathcal{C}}(\overline{t}_n) \to t$ behaves monotonically w.r.t. the arguments $\overline{t}_n$ and antimonotonically w.r.t. the result t. Formally: For all $\overline{t}_n, \overline{t'}_n, t, t' \in C$ such that $p^{\mathcal{C}}(\overline{t}_n) \to t$, $\overline{t}_n \sqsubseteq \overline{t'}_n$ and $t \sqsupseteq t'$, $p^{\mathcal{C}}(\overline{t'}_n) \to t'$ also holds.*

[2] We slightly modify the statement of the *radicality* property, rendering it simpler than in [13] but sufficient for practical purposes.

(b) **Radicality***: For all $p \in PF^n$, as soon as the arguments given to $p^{\mathcal{C}}$ have enough information to return a result other than $\bot$, the same arguments suffice already for returning a simple total result. Formally: For all $\overline{t}_n, t \in C$, if $p^{\mathcal{C}}(\overline{t}_n) \to t$ then $t = \bot$ or else $t \in B \cup DC^0$.*

Note that symbols $h \in DC \cup DF$ are given no interpretation in $\mathcal{C}$. As we will see in Section 3, symbols in $c \in DC$ are interpreted as free constructors, and the interpretation of symbols $f \in DF$ is program-dependent. We assume that any signature Σ includes two nullary constructors $true$ and $false$ for the boolean values, and a binary symbol $== \in PF^2$ used in infix notation and interpreted as *strict equality*; see [13] for details. For the examples in this report we will use a constraint domain $\mathcal{R}$ whose set of basic elements is $C_{\mathcal{R}} = \mathbb{R}$ and whose primitives functions correspond to the usual arithmetic operations $+, \times, \ldots$ and the usual boolean-valued comparison operations $\leq, <, \ldots$ over $\mathbb{R}$. Other useful instances of constraint domains can be found in [13].

Atomic constraints over $\mathcal{C}$ have the form $p(\overline{e}_n) == v$ [3] with $p \in PF^n$, $e_i \in \mathrm{Exp}_\bot(\Sigma, B, \mathcal{V}ar)$ and $v \in \mathcal{V}ar \cup DC^0 \cup B_{\mathcal{C}}$. Atomic constraints of the form $p(\overline{e}_n) == true$ are abbreviated as $p(\overline{e}_n)$. In particular, $(e_1 == e_2) == true$ is abbreviated as $e_1 == e_2$. Atomic constraints of the form $(e_1 == e_2) == false$ are abbreviated as $e_1 /= e_2$.

Compound constraints are built from atomic constraints using logical conjunction, existential quantification, and sometimes other logical operations. Constraints without occurrences of symbols $f \in DF$ are called *primitive*. We will note atomic constraints as δ, sets of atomic constraints as Δ, atomic primitive constraints as π, and sets of atomic primitive constraints as Π. When interpreting set of constraints, we will treat them as the conjunction of their members.

Ground substitutions η such that $X\eta \in \mathrm{Term}_\bot(\Sigma, B)$ for all $X \in \mathrm{vdom}(\eta)$ are called *variable valuations* over $\mathcal{C}$. The set of all possible variable valuations is noted $\mathrm{Val}_{\mathcal{C}}$. The *solution set* $\mathrm{Sol}_{\mathcal{C}}(\Pi) \subseteq \mathrm{Val}_{\mathcal{C}}$ includes as members those valuations η such that $\pi\sigma$ is true in $\mathcal{C}$ for all $\pi \in \Pi$; see [13] for a formal definition. In case that $\mathrm{Sol}_{\mathcal{C}}(\Pi) = \emptyset$ we say that Π is *unsatisfiable* and we write $\mathrm{Unsat}_{\mathcal{C}}(\Pi)$. In case that $\mathrm{Sol}_{\mathcal{C}}(\Pi) \subseteq \mathrm{Sol}_{\mathcal{C}}(\pi)$ we say that π is *entailed* by Π in $\mathcal{C}$ and we write $\Pi \models_{\mathcal{C}} \pi$. Note that the notions defined in this paragraph only make sense for primitive constraints.

In this report we are interested in pairs consisting of a qualification domain and a constraint domain that are related in the following technical sense:

Definition 3 (Expressing $\mathcal{D}$ in $\mathcal{C}$). *A qualification domain $\mathcal{D}$ with carrier set $D_{\mathcal{D}}$ is expressible in a constraint domain $\mathcal{C}$ with carrier set $C_{\mathcal{C}}$ if $D_{\mathcal{D}} \setminus \{\mathbf{b}\} \subseteq C_{\mathcal{C}}$ and the two following requirements are satisfied:*

1. *There is a primitive $\mathcal{C}$-constraint $\mathsf{qVal}(X)$ depending on the variable X, such that $Sol_{\mathcal{C}}(\mathsf{qVal}(X)) = \{\eta \in Val_{\mathcal{C}} \mid \eta(X) \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}\}$.*
2. *There is a primitive $\mathcal{C}$-constraint $\mathsf{qBound}(X, Y, Z)$ depending on the variables X, Y, Z, such that any $\eta \in Val_{\mathcal{C}}$ such that $\eta(X)$, $\eta(Y)$, $\eta(Z) \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ verifies $\eta \in Sol_{\mathcal{C}}(\mathsf{qBound}(X, Y, Z)) \Longleftrightarrow \eta(X) \trianglelefteq \eta(Y) \circ \eta(Z)$.* □

[3] Written as $p(\overline{e}_n) \to! v$ in [13].

Intuitively, $\mathsf{qBound}(X, Y, Z)$ encodes the $\mathcal{D}$-statement $X \trianglelefteq Y \circ Z$ as a $\mathcal{C}$-constraint. As convenient notations, we will write $\ulcorner X \trianglelefteq Y \circ Z \urcorner$, $\ulcorner X \trianglelefteq Y \urcorner$ and $\ulcorner X \trianglerighteq Y \urcorner$ in place of $\mathsf{qBound}(X, Y, Z)$, $\mathsf{qBound}(X, \mathbf{t}, Y)$ and $\mathsf{qBound}(Y, \mathbf{t}, X)$, respectively. In the sequel, $\mathcal{C}$-constraints of the form $\ulcorner \kappa \urcorner$ are called *qualification constraints*, and Ω is used as notation for sets of qualification constraints. We also write $\mathrm{Val}_{\mathcal{D}}$ for the set of all $\mu \in \mathrm{Val}_{\mathcal{C}}$ such that $X\mu \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ for all $X \in \mathrm{vdom}(\mu)$, called *$\mathcal{D}$-valuations*.

Note that $\mathcal{U}$ can be expressed in $\mathcal{R}$, because $D_{\mathcal{U}} \setminus \{0\} = (0, 1] \subseteq \mathbb{R} \subseteq C_{\mathcal{R}}$, $\mathsf{qVal}(X)$ can be built as the $\mathcal{R}$-constraint $0 < X \wedge X \leq 1$ and $\ulcorner X \trianglelefteq Y \circ Z \urcorner$ can be built as the $\mathcal{R}$-constraint $X \leq Y \times Z$. Other instances of qualification domains presented in [19] are also expressible in $\mathcal{R}$.

3 A Qualified Declarative Programming Scheme

In this section we present the scheme QCFLP$(\mathcal{D}, \mathcal{C})$ announced in the Introduction, and we develop alternative characterizations of its declarative semantics using an interpretation transformer and a rewriting logic. The parameters $\mathcal{D}$ and $\mathcal{C}$ respectively stand for a qualification domain and a constraint domain with certain signature Σ. By convention, we only allow those instances of the scheme verifying that $\mathcal{D}$ is expressible in $\mathcal{C}$ in the sense of Definition 3. For example, QCFLP$(\mathcal{U}, \mathcal{R})$ is an allowed instance.

Technically, the results presented here extend similar ones known for the CFLP$(\mathcal{C})$ sheme [13], omitting higher-order functions and adding a suitable treatment of qualifications. In particular, the qc-interpretations for QCFLP$(\mathcal{D}, \mathcal{C})$-programs are a natural extension of the c-interpretations for CFLP$(\mathcal{C})$-programs introduced in [13]. In turn, these were inspired by the π-interpretations for the CLP$(\mathcal{C})$ scheme proposed by Dore, Gabbrielli and Levi [7,6].

3.1 Programs, Interpretations and Models

A QCFLP$(\mathcal{D}, \mathcal{C})$-program is a set $\mathcal{P}$ of program rules. A program rule has the form $f(\bar{t}_n) \xrightarrow{\alpha} r \Leftarrow \Delta$ where $f \in DF^n$, $\bar{t}_n$ is a lineal sequence of Σ-terms, $\alpha \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ is an attenuation factor, r is a Σ-expression and Δ is a sequence of atomic $\mathcal{C}$-constraints δ_j $(1 \leq j \leq m)$, interpreted as conjunction. The undefined symbol $\bot$ is not allowed to occur in program rules.

The library program shown in Figure 1 is an example of QCFLP$(\mathcal{U}, \mathcal{R})$-program. Leaving aside the attenuation factors, this is clearly not a confluent conditional term rewriting system. Certain program rules, as e.g. those for `guessGenre`, are intended to specify the behavior of *non-deterministic functions*. As argued elsewhere [17], the semantics of non-deterministic functions for the purposes of Functional Logic Programming is not suitably described by ordinary rewriting. Inspired by the approach in [13], we will overcome this difficulty by designing special inference mechanisms to derive semantically meaningful statements from programs. The kind of statements that we will consider are defined below:

Definition 4 (qc-Statements). *Assume partial Σ-expression e, partial Σ-terms $t, t', \bar{t}_n$, a qualification value $d \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$, an atomic $\mathcal{C}$-constraint δ and a finite set of atomic primitive $\mathcal{C}$-constraints Π. A* qualified constrained statement *(briefly, qc-statement) φ must have one of the following two forms:*

1. qc-production $(e \to t)\sharp d \Leftarrow \Pi$. *Such a qc-statement is called* trivial *iff either t is $\bot$ or else $Unsat_{\mathcal{C}}(\Pi)$. Its intuitive meaning is that a rewrite sequence $e \to^* t'$ using program rules and with attached qualification value d is allowed in our intended semantics for some $t' \sqsupseteq t$, under the assumption that Π holds. By convention, qc-productions of the form $(f(\bar{t}_n) \to t)\sharp d \Leftarrow \Pi$ with $f \in DF^n$ are called* qc-facts.
2. qc-atom $\delta\sharp d \Leftarrow \Pi$. *Such a qc-statement is called* trivial *iff $Unsat_{\mathcal{C}}(\Pi)$. Its intuitive meaning is that δ is entailed by the program rules with attached qualification value d, under the assumption that Π holds.* □

Our semantics will use program interpretations defined as sets of qc-facts with certain closure properties. As an auxiliary tool we need the following technical notion:

Definition 5 ($(\mathcal{D},\mathcal{C})$-Entailment). *Given two qc-statements φ and φ', we say that φ $(\mathcal{D},\mathcal{C})$-entails φ' (in symbols, $\varphi \succcurlyeq_{\mathcal{D},\mathcal{C}} \varphi'$) iff one of the following two cases hold:*

1. *$\varphi = (e \to t)\sharp d \Leftarrow \Pi$, $\varphi' = (e' \to t')\sharp d' \Leftarrow \Pi'$, and there is some substitution σ such that $\Pi' \models_{\mathcal{C}} \Pi\sigma$, $d \trianglerighteq d'$, $e\sigma \sqsubseteq e'$ and $t\sigma \sqsupseteq t'$.*
2. *$\varphi = \delta\sharp d \Leftarrow \Pi$, $\varphi' = \delta'\sharp d' \Leftarrow \Pi'$, and there is some substitution σ such that $\Pi' \models_{\mathcal{C}} \Pi\sigma$, $d \trianglerighteq d'$, $\delta\sigma \sqsubseteq \delta'$.* □

The intended meaning of $\varphi \succcurlyeq_{\mathcal{D},\mathcal{C}} \varphi'$ is that φ' follows from φ, regardless of the interpretation of the defined function symbols $f \in DF$ occurring in φ, φ'. Intuitively, this is the case because the interpretations of defined function symbols are expected to satisfy the monotonicity properties stated for the case of primitive function symbols in Definition 2. The following example may help to understand the idea:

Example 1 ($(\mathcal{U},\mathcal{R})$-entailment). Let φ, φ' be defined as:

$$\begin{aligned} \varphi &: (f(X\!:\!Xs) \to Xs)\sharp 0.8 \Leftarrow X \times X \neq 0 \\ \varphi' &: (f(A\!:\!(B\!:\![\,])) \to \bot\!:\!\bot)\sharp 0.7 \Leftarrow A < 0 \end{aligned}$$

Then $\varphi \succcurlyeq_{\mathcal{U},\mathcal{R}} \varphi'$ with $\sigma = \{X \mapsto A,\ Xs \mapsto B\!:\!\bot\}$ because:

- $\Pi' \models_{\mathcal{R}} \Pi\sigma$, since $\Pi' = \{A < 0\}$, $\Pi\sigma = \{X \times X \neq 0\}\sigma = \{A \times A \neq 0\}$, and $A \times A \neq 0$ is entailed by $A < 0$ in $\mathcal{R}$.
- $d \trianglerighteq d'$ holds in $\mathcal{U}$, since $d = 0.8 \geq 0.7 = d'$.
- $e\sigma \sqsubseteq e'$, since $e\sigma = f(X\!:\!Xs)\sigma = f(A\!:\!(B\!:\!\bot)) \sqsubseteq f(A\!:\!(B\!:\![\,])) = e'$.
- $t\sigma \sqsupseteq t'$, since $t\sigma = Xs\sigma = B\!:\!\bot \sqsupseteq \bot\!:\!\bot = t'$. □

Now we can define program interpretations as follows:

Definition 6 (qc-Interpretations). *A* qualified constrained interpretation *(or* qc-interpretation*) over $\mathcal{D}$ and $\mathcal{C}$ is a set $\mathcal{I}$ of qc-facts including all trivial and entailed qc-facts. In other words, a set $\mathcal{I}$ of qc-facts such that $cl_{\mathcal{D},\mathcal{C}}(\mathcal{I}) \subseteq \mathcal{I}$, where the closure over $\mathcal{D}$ and $\mathcal{C}$ of $\mathcal{I}$ is defined as:*

$$cl_{\mathcal{D},\mathcal{C}}(\mathcal{I}) =_{\text{def}} \{\varphi \mid \varphi \textit{ trivial}\} \cup \{\varphi' \mid \varphi \succcurlyeq_{\mathcal{D},\mathcal{C}} \varphi' \textit{ for some } \varphi \in \mathcal{I}\} \ .$$

We write $Int_{\mathcal{D},\mathcal{C}}$ for the set of all qc-interpretations over $\mathcal{D}$ and $\mathcal{C}$.

QTI $\dfrac{}{\varphi}$ if φ is a trivial qc-statement.

QRR $\dfrac{}{(v \to v)\sharp d \Leftarrow \Pi}$ if $v \in \mathcal{V}ar \cup B_{\mathcal{C}}$ and $d \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$.

QDC $\dfrac{(\ (e_i \to t_i)\sharp d_i \Leftarrow \Pi\ )_{i=1\ldots n}}{(c(\overline{e}_n) \to c(\overline{t}_n))\sharp d \Leftarrow \Pi}$ if $c \in DC^n$ and $d \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ verifies $d \trianglelefteq d_i$ $(1 \leq i \leq n)$.

QDF$_{\mathcal{I}}$ $\dfrac{(\ (e_i \to t_i)\sharp d_i \Leftarrow \Pi\ )_{i=1\ldots n}}{(f(\overline{e}_n) \to t)\sharp d \Leftarrow \Pi}$

if $f \in DF^n$, non-trivial $((f(\overline{t}_n) \to t)\sharp d_0 \Leftarrow \Pi) \in \mathcal{I}$
and $d \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ verifies $d \trianglelefteq d_i$ $(0 \leq i \leq n)$.

QPF $\dfrac{(\ (e_i \to t_i)\sharp d_i \Leftarrow \Pi\ )_{i=1\ldots n}}{(p(\overline{e}_n) \to v)\sharp d \Leftarrow \Pi}$ if $p \in PF^n$, $v \in \mathcal{V}ar \cup DC^0 \cup B_{\mathcal{C}}$,

$\Pi \models_{\mathcal{C}} p(\overline{t}_n) \to v$ and $d \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ verifies $d \trianglelefteq d_i$ $(1 \leq i \leq n)$.

QAC $\dfrac{(\ (e_i \to t_i)\sharp d_i \Leftarrow \Pi\ )_{i=1\ldots n}}{(p(\overline{e}_n) == v)\sharp d \Leftarrow \Pi}$ if $p \in PF^n$, $v \in \mathcal{V}ar \cup DC^0 \cup B_{\mathcal{C}}$,

$\Pi \models_{\mathcal{C}} p(\overline{t}_n) == v$ and $d \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ verifies $d \trianglelefteq d_i$ $(1 \leq i \leq n)$.

Fig. 2. Qualified Constrained Rewriting Logic for Interpretations

Given a qc-interpretation $\mathcal{I}$, the inference rules displayed in Fig. 2 are used to derive qc-statements from the qc-facts belonging to $\mathcal{I}$. The inference system consisting of these rules is called *Qualified Constrained Rewriting Logic for Interpretations* and noted as $\mathcal{I}$-QCRWL$(\mathcal{D},\mathcal{C})$. The notation $\mathcal{I} \vdash_{\mathcal{D},\mathcal{C}} \varphi$ is used to indicate that φ can be derived from $\mathcal{I}$ in $\mathcal{I}$-QCRWL$(\mathcal{D},\mathcal{C})$. By convention, we agree that no other inference rule is used whenever **QTI** is applicable. Therefore, trivial qc-statements can only be inferred by rule **QTI**. As usual in formal inference systems, $\mathcal{I}$-QCRWL$(\mathcal{D},\mathcal{C})$ proofs can be represented as trees whose nodes correspond to inference steps.

In the sequel, the inference rules $\mathbf{QDF}_{\mathcal{I}}$, **QPF** and **QAC** will be called *crucial*. The notation $|\mathcal{T}|$ will denote the number of inference steps within the proof tree $\mathcal{T}$ that are *not crucial*. Proof trees with no crucial inferences (i.e. such that $|\mathcal{T}| = 0$) will be called *easy*. The following lemma states some technical properties of $\mathcal{I}$-QCRWL$(\mathcal{D},\mathcal{C})$.

Lemma 1 (Some properties of $\mathcal{I}$-QCRWL$(\mathcal{D},\mathcal{C})$).

1. *Approximation property: For any non-trivial φ of the form $(t \to t')\sharp d \Leftarrow \Pi$ where $t, t' \in Term_\perp(\Sigma, B, \mathcal{V}ar)$, the three following affirmations are equivalent: (a) $t \sqsupseteq t'$; (b) $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} \varphi$ with an easy proof tree; and (c) $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} \varphi$.*
2. *Primitive c-atoms: For any primitive c-atom $p(\bar{t}_n) == v$, one has $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} (p(\bar{t}_n) == v)\sharp d \Leftarrow \Pi \iff \Pi \models_{\mathcal{C}} p(\bar{t}_n) == v$.*
3. *Entailment property: $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} \varphi$ with a proof tree $\mathcal{T}$ and $\varphi \succcurlyeq_{\mathcal{D},\mathcal{C}} \varphi' \implies \mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} \varphi'$ with a proof tree $\mathcal{T}'$ such that $|\mathcal{T}'| \leq |\mathcal{T}|$.*
4. *Conservation property: For any qc-fact φ, one has $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} \varphi \iff \varphi \in \mathcal{I}$.*

Proof. We argue separately for each of the four properties:

[*1.*] (*Approximation property*). The terms t and t' involve neither defined nor primitive function symbols. Due to the form of the $\mathcal{I}$-QCRWL$(\mathcal{D},\mathcal{C})$ inference rules, a proof of the qc-statement $(t \to t')\sharp d \Leftarrow \Pi$ will involve no crucial inferences and it will succeed iff $t \sqsupseteq t'$. A formal proof can be easily obtained reasoning by induction on the syntactic size of t, similarly as in item 3. of Lemma 1 from [13].

[*2.*] (*Primitive c-atoms*). Let φ be $(p(\bar{t}_n) == v)\sharp d \Leftarrow \Pi$. If φ is trivial, then $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} (p(\bar{t}_n) == v)\sharp d \Leftarrow \Pi$ can be proved with just one **QTI** inference, and $\Pi \models_{\mathcal{C}} p(\bar{t}_n) == v$ also holds because of $\mathrm{Unsat}_{\mathcal{C}}(\Pi)$. If φ is not trivial, then:

- ($\Leftarrow$) Assume $\Pi \models_{\mathcal{C}} p(\bar{t}_n) == v$. Then $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} (p(\bar{t}_n) == v)\sharp d \Leftarrow \Pi$ can be obtained with a proof of the form

$$\frac{(\ (t_i \to t_i)\sharp \mathbf{t} \Leftarrow \Pi\ )_{i=1\ldots n}}{(p(\bar{t}_n) == v)\sharp d \Leftarrow \Pi}\ \text{QAC}$$

  where each of the n premises has an easy $\mathcal{I}$-QCRWL$(\mathcal{D},\mathcal{C})$-proof due to the approximation property (since $t_i \sqsupseteq t_i$).
- ($\Rightarrow$) Assume now $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} (p(\bar{t}_n) == v)\sharp d \Leftarrow \Pi$. The $\mathcal{I}$-QCRWL$(\mathcal{D},\mathcal{C})$-proof will have the form

$$\frac{(\ (t_i \to t'_i)\sharp d_i \Leftarrow \Pi\ )_{i=1\ldots n}}{(p(\bar{t}_n) == v)\sharp d \Leftarrow \Pi}\ \text{QAC}$$

  where $\Pi \models_{\mathcal{C}} p(\overline{t'}_n) == v$ and $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} (t_i \to t_i)\sharp d_i \Leftarrow \Pi$, $d \trianglelefteq d_i$ hold for all $1 \leq i \leq n$. Due to the approximation property, we can conclude that $t_i \sqsupseteq t'_i$ holds for $1 \leq i \leq n$, which implies $\Pi \models_{\mathcal{C}} p(\bar{t}_n) == v$ because of the monotonic behavior of primitive functions in constraint domains.

[*3.*] (*Entailment property*). Assume $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} \varphi$ with a $\mathcal{I}$-QCRWL$(\mathcal{D},\mathcal{C})$-proof tree $\mathcal{T}$. We must prove that $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} \varphi'$ with some proof tree $\mathcal{T}'$ such that $|\mathcal{T}'| \leq |\mathcal{T}|$. If φ' results trivial, then it is proved with just one **QTI** inference step, and therefore $|\mathcal{T}'| = 0 \leq |\mathcal{T}|$. In the sequel, we assume φ' non-trivial and we reason by induction on the number of inference steps within $\mathcal{T}$. We distinguish cases according to the inference step at the root of $\mathcal{T}$:

- **QTI**: From Definition 5 it is easy to check that φ' must be trivial whenever $\varphi \succcurlyeq_{\mathcal{D},\mathcal{C}} \varphi'$ and φ is trivial. Since we are assuming that φ' is not trivial, this case cannot happen.
- **QRR**: In this case φ is of the form $(v \to v)\sharp d \Leftarrow \Pi$ with either $v \in B_{\mathcal{C}}$ or $v \in \mathcal{V}ar$. Since $\varphi \succcurlyeq_{\mathcal{D},\mathcal{C}} \varphi'$, we assume $\varphi' : (v' \to v')\sharp d' \Leftarrow \Pi'$ with $\Pi' \models_{\mathcal{C}} \Pi\sigma$, $d \trianglerighteq d'$ and $v\sigma = v'$ for some substitution σ. If $v \in B_{\mathcal{C}}$, then also $v' \in B_{\mathcal{C}}$ and $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} \varphi'$ can be proved with a proof tree $\mathcal{T}'$ consisting of just one **QRR** inference step. If $v \in \mathcal{V}ar$, then $v' \in \mathrm{Term}_{\perp}(\Sigma, B, \mathcal{V}ar)$, and $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} \varphi'$ can be proved with a proof tree $\mathcal{T}'$ consisting only of **QDC** and **QRR** inferences. In both cases, $|\mathcal{T}'| = 0 \leq |\mathcal{T}|$.
- **QDC**: In this case $\varphi : (c(\overline{e}_n) \to c(\overline{t}_n))\sharp d \Leftarrow \Pi$ and $\mathcal{T}$ has the form

$$\frac{(\ (e_i \to t_i)\sharp d_i \Leftarrow \Pi\ )_{i=1\ldots n}}{(c(\overline{e}_n) \to c(\overline{t}_n))\sharp d \Leftarrow \Pi}\ \mathrm{QDC}$$

  where $c \in DC^n$, $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} (e_i \to t_i)\sharp d_i \Leftarrow \Pi$ with proof tree $\mathcal{T}_i$, and $d \trianglelefteq d_i$ $(1 \leq i \leq n)$. Since $\varphi \succcurlyeq_{\mathcal{D},\mathcal{C}} \varphi'$, we can assume that φ' has the form $(c(\overline{e'}_n) \to c(\overline{t'}_n))\sharp d' \Leftarrow \Pi'$ with $e_i\sigma \sqsubseteq e'_i$ $(1 \leq i \leq n)$, $c(\overline{t}_n)\sigma \sqsupseteq c(\overline{t'}_n)$, $d \trianglerighteq d'$ and $\Pi' \models_{\mathcal{C}} \Pi\sigma$ for some substitution σ. For $1 \leq i \leq n$, we clearly obtain $(e_i \to t_i)\sharp d_i \Leftarrow \Pi \succcurlyeq_{\mathcal{D},\mathcal{C}} (e'_i \to t'_i)\sharp d_i \Leftarrow \Pi'$, and by induction hypothesis we can assume $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} (e'_i \to t'_i)\sharp d_i \Leftarrow \Pi'$ with proof tree $\mathcal{T}'_i$ such that $|\mathcal{T}'_i| \leq |\mathcal{T}_i|$. Then we get $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} (c(\overline{e'}_n) \to c(\overline{t'}_n))\sharp d' \Leftarrow \Pi'$ with a proof tree $\mathcal{T}'$ such that $|\mathcal{T}'| \leq |\mathcal{T}|$. More precisely, $\mathcal{T}'$ has the form

$$\frac{(\ (e'_i \to t'_i)\sharp d_i \Leftarrow \Pi'\ )_{i=1\ldots n}}{(c(\overline{e'}_n) \to c(\overline{t'}_n))\sharp d' \Leftarrow \Pi'}\ \mathrm{QDC}$$

  where $d' \trianglelefteq d_i$ follows from $d' \trianglelefteq d \trianglelefteq d_i$ $(1 \leq i \leq n)$ and each premise is proved by $\mathcal{T}'_i$.
- **QDF**$_{\mathcal{I}}$: In this case $\varphi : (f(\overline{e}_n) \to t)\sharp d \Leftarrow \Pi$ and $\mathcal{T}$ has the form

$$\frac{(\ (e_i \to t_i)\sharp d_i \Leftarrow \Pi\ )_{i=1\ldots n}}{(f(\overline{e}_n) \to t)\sharp d \Leftarrow \Pi}\ \mathrm{QDF}_{\mathcal{I}}$$

  where $f \in DF^n$ and there is some non-trivial $\psi = (f(\overline{t}_n) \to t)\sharp d_0 \Leftarrow \Pi)$ such that $\psi \in \mathcal{I}$, $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} (e_i \to t_i)\sharp d_i \Leftarrow \Pi$ with proof tree $\mathcal{T}_i$ and $d \trianglelefteq d_i$ $(0 \leq i \leq n)$. Since $\varphi \succcurlyeq_{\mathcal{D},\mathcal{C}} \varphi'$, we can assume $\varphi' = (f(\overline{e'}_n) \to t')\sharp d' \Leftarrow \Pi'$ with $e_i\sigma \sqsubseteq e'_i$ $(1 \leq i \leq n)$, $t\sigma \sqsupseteq t'$, $d \trianglerighteq d'$ and $\Pi' \models_{\mathcal{C}} \Pi\sigma$ for some substitution σ. For $1 \leq i \leq n$, we get $(e_i \to t_i)\sharp d_i \Leftarrow \Pi \succcurlyeq_{\mathcal{D},\mathcal{C}} (e'_i \to t_i\sigma)\sharp d_i \Leftarrow \Pi'$, and by induction hypothesis we can assume $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} (e'_i \to t_i\sigma)\sharp d_i \Leftarrow \Pi'$ with proof

tree $\mathcal{T}'_i$ such that $|\mathcal{T}'_i| \leq |\mathcal{T}_i|$. Consider now $\psi' = ((f(\bar{t}_n)\sigma \to t')\sharp d_0 \Leftarrow \Pi')$. Clearly, $\psi \succcurlyeq_{\mathcal{D},\mathcal{C}} \psi'$ and therefore $\psi' \in \mathcal{I}$ because $\mathcal{I}$ is closed under $(\mathcal{D},\mathcal{C})$-entailment. Using this ψ' we get $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} (f(\overline{e'}_n) \to t')\sharp d' \Leftarrow \Pi'$ with a proof tree $\mathcal{T}'$ such that $|\mathcal{T}'| \leq |\mathcal{T}|$. More precisely, $\mathcal{T}'$ has the form

$$\frac{(\ (e'_i \to t_i\sigma)\sharp d_i \Leftarrow \Pi'\ )_{i=1\ldots n}}{(f(\overline{e'}_n) \to t')\sharp d' \Leftarrow \Pi'}\ \mathrm{QDF}_{\mathcal{I}}$$

where $d' \trianglelefteq d_i$ follows from $d' \trianglelefteq d \trianglelefteq d_i$ $(0 \leq i \leq n)$ and each premise is proved by $\mathcal{T}'_i$.

– **QPF**: In this case $\varphi : (p(\bar{e}_n) \to v)\sharp d \Leftarrow \Pi$ and $\mathcal{T}$ has the form

$$\frac{(\ (e_i \to t_i)\sharp d_i \Leftarrow \Pi\ )_{i=1\ldots n}}{(p(\bar{e}_n) \to v)\sharp d \Leftarrow \Pi}\ \mathrm{QPF}$$

where $p \in PF^n$, $v \in \mathcal{V}ar \cup DC^0 \cup B_{\mathcal{C}}$, $\Pi \models_{\mathcal{C}} p(\bar{t}_n) \to v$, $d \trianglelefteq d_i$ and $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} (e_i \to t_i)\sharp d_i \Leftarrow \Pi$ with proof tree $\mathcal{T}_i$ $(1 \leq i \leq n)$. Since $\varphi \succcurlyeq_{\mathcal{D},\mathcal{C}} \varphi'$, we can assume φ' to be of the form $(p(\overline{e'}_n) \to v')\sharp d' \Leftarrow \Pi'$ with $e_i\sigma \sqsubseteq e'_i$ $(1 \leq i \leq n)$, $v\sigma \sqsupseteq v'$, $d \trianglerighteq d'$ and $\Pi' \models_{\mathcal{C}} \Pi\sigma$ for some substitution σ. For $1 \leq i \leq n$, we get $(e_i \to t_i)\sharp d_i \Leftarrow \Pi \succcurlyeq_{\mathcal{D},\mathcal{C}} (e'_i \to t_i\sigma)\sharp d_i \Leftarrow \Pi'$, and by induction hypothesis we can assume $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} (e'_i \to t_i\sigma)\sharp d_i \Leftarrow \Pi'$ with proof tree $\mathcal{T}'_i$ such that $|\mathcal{T}'_i| \leq |\mathcal{T}_i|$. Moreover, we can also assume $v' \in \mathcal{V}ar \cup DC^0 \cup B_{\mathcal{C}}$ because p is a primitive function symbol and φ' is not trivial. From $v, v' \in \mathcal{V}ar \cup DC^0 \cup B_{\mathcal{C}}$ and $v\sigma \sqsupseteq v'$ we can conclude that $v\sigma = v'$. Then, from $\Pi \models_{\mathcal{C}} p(\bar{t}_n) \to v$ and $\Pi' \models_{\mathcal{C}} \Pi\sigma$ we can deduce $\Pi' \models_{\mathcal{C}} p(\bar{t}_n)\sigma \to v'$. Putting everything together, we get $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} (p(\overline{e'}_n) \to v')\sharp d' \Leftarrow \Pi'$ with a proof tree $\mathcal{T}'$ such that $|\mathcal{T}'| \leq |\mathcal{T}|$. More precisely, $\mathcal{T}'$ has the form

$$\frac{(\ (e'_i \to t_i\sigma)\sharp d_i \Leftarrow \Pi'\ )_{i=1\ldots n}}{(p(\overline{e'}_n) \to v')\sharp d' \Leftarrow \Pi'}\ \mathrm{QPF}$$

where $d' \trianglelefteq d_i$ follows from $d' \trianglelefteq d \trianglelefteq d_i$ $(1 \leq i \leq n)$ and each premise is proved by $\mathcal{T}'_i$.

– **QAC**: Similar to the case for **QPF**.

[*4.*] (*Conservation property*). Assume $\varphi : (f(\bar{t}_n) \to t)\sharp d \Leftarrow \Pi$. In the case that φ is a trivial qc-fact, it is true by definition of qc-interpretation that $\varphi \in \mathcal{I}$, and $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} \varphi$ follows by rule **QTI**. Therefore the property is satisfied for trivial qc-facts. If φ is not trivial, we prove each implication as follows:

– $(\Leftarrow)$ Assume $\varphi \in \mathcal{I}$. Then $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} \varphi$ with a $\mathcal{I}$-QCRWL$(\mathcal{D},\mathcal{C})$-proof tree of the form:

$$\frac{(\ (t_i \to t_i)\sharp \mathbf{t} \Leftarrow \Pi\ )_{i=1\ldots n}}{(f(\bar{t}_n) \to t)\sharp d \Leftarrow \Pi}\ \mathrm{QDF}_{\mathcal{I}} \text{ using } \varphi \in \mathcal{I}$$

where each premise has an easy $\mathcal{I}$-QCRWL$(\mathcal{D},\mathcal{C})$-proof tree due to the approximation property, and $d \trianglelefteq d, \mathbf{t}$ hold trivially.

– ($\Rightarrow$) Assume $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} \varphi$. As φ is not trivial, there is a $\mathcal{I}$-QCRWL$(\mathcal{D},\mathcal{C})$-proof tree of the form:

$$\frac{(\ (t_i \to t'_i)\sharp d_i \Leftarrow \Pi\ )_{i=1\ldots n}}{(f(\overline{t}_n) \to t)\sharp d \Leftarrow \Pi}\ \mathrm{QDF}_{\mathcal{I}}\ \text{using}\ \varphi' = (f(\overline{t'}_n) \to t)\sharp d' \Leftarrow \Pi) \in \mathcal{I}$$

where $d \trianglelefteq d', d_i$ and $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} (t_i \to t'_i)\sharp d_i \Leftarrow \Pi\, (1 \leq i \leq n)$. For each $1 \leq i \leq n$, we claim that $t'_i \sqsubseteq t_i$. If $t'_i = \bot$ the claim is trivial. If $t'_i \neq \bot$, then $(t_i \to t'_i)\sharp d_i \Leftarrow \Pi$ is a non-trivial qc-production and the claim follows from $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} (t_i \to t'_i)\sharp d_i \Leftarrow \Pi$ and the approximation property. Now, the claim together with $\Pi \models_{\mathcal{C}} \Pi$, $d' \trianglerighteq d$ and $t \sqsupseteq t$ yields $\varphi' \succcurlyeq_{\mathcal{D},\mathcal{C}} \varphi$. Since $\varphi' \in \mathcal{I}$ and $\mathcal{I}$ is closed under $(\mathcal{D},\mathcal{C})$-entailment, we can conclude that $\varphi \in \mathcal{I}$. □

Next, we can define program models and semantic consequence, adapting ideas from the so-called *strong semantics* of [13]. [4]

Definition 7 (Models and semantic consequence). *Let a QCFLP$(\mathcal{D},\mathcal{C})$-program $\mathcal{P}$ be given.*

1. *A qc-interpretation $\mathcal{I}$ is a model of $R_l : (f(\overline{t}_n) \xrightarrow{\alpha} r \Leftarrow \overline{\delta}_m) \in \mathcal{P}$ (in symbols, $\mathcal{I} \models_{\mathcal{D},\mathcal{C}} R_l$) iff for every substitution θ, for every set of atomic primitive $\mathcal{C}$-constraints Π, for every c-term $t \in Term_\bot(\Sigma, B, \mathcal{V}ar)$ and for all $d, d_0, \ldots, d_m \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ such that $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} \delta_i\theta\sharp d'_i \Leftarrow \Pi\, (1 \leq i \leq m)$, $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} (r\theta \to t)\sharp d'_0 \Leftarrow \Pi$ and $d \trianglelefteq \alpha \circ d_i\, (0 \leq i \leq m)$, one has $((f(\overline{t}_n)\theta \to t)\sharp d \Leftarrow \Pi) \in \mathcal{I}$.*
2. *A qc-interpretation $\mathcal{I}$ is a* model *of $\mathcal{P}$ (in symbols, $\mathcal{I} \models_{\mathcal{D},\mathcal{C}} \mathcal{P}$) iff $\mathcal{I}$ is a model of every program rule belonging to $\mathcal{P}$.*
3. *A qc-statement φ is a semantic consequence of $\mathcal{P}$ (in symbols, $\mathcal{P} \models_{\mathcal{D},\mathcal{C}} \varphi$) iff $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} \varphi$ holds for every qc-interpretation $\mathcal{I}$ such that $\mathcal{I} \models_{\mathcal{D},\mathcal{C}} \mathcal{P}$.* □

3.2 Least Models

We will now present two different characterizations for the least model of a given program $\mathcal{P}$: in the first place as a least fixpoint of an interpretation transformer and in the second place as the set of qc-facts derivable from $\mathcal{P}$ in a special rewriting logic.

A fixpoint characterization of least models.

A well-known way of characterizing least program models is to exploit the lattice structure of the family of all program interpretations and to obtain the least model of a give program $\mathcal{P}$ as the least fixpoint of an interpretation transformer related to $\mathcal{P}$. Such characterizations are know for logic programming [11,2], constraint logic programming [7,6,10], constraint functional logic programming [13]

[4] Weak models and weak semantic consequence could be also defined similarly as in [13], but strong semantics suffices for the purposes of this report.

and qualified logic programming [19]. Our approach here extends that in [13] by adding qualification values.

The next result, whose easy proof is omitted, provides a lattice structure of program interpretations:

Proposition 2 (Interpretations Lattice). *$Int_{\mathcal{D},\mathcal{C}}$ defined as the set of all qc-interpretations over the qualification domain $\mathcal{D}$ and the constraint domain $\mathcal{C}$ is a complete lattice w.r.t. the set inclusion ordering ($\subseteq$). Moreover, the bottom element $\perp\!\!\!\perp$ and the top element $\top\!\!\!\top$ of this lattice are characterized as $\perp\!\!\!\perp = cl_{\mathcal{D},\mathcal{C}}(\{\varphi \mid \varphi$ is a trivial qc-fact$\})$ and $\top\!\!\!\top = \{\varphi \mid \varphi$ is any qc-fact$\}$.*

Now we define an *interpretations transformer* $ST_{\mathcal{P}}$ intended to formalize the computation of immediate consequences from the qc-facts belonging to a given qc-interpretation.

Definition 8 (Interpretations transformers). *Assuming a QCFLP$(\mathcal{D},\mathcal{C})$-program $\mathcal{P}$ and a qc-interpretation $\mathcal{I}$, $ST_{\mathcal{P}} : Int_{\mathcal{D},\mathcal{C}} \to Int_{\mathcal{D},\mathcal{C}}$ is defined as $ST_{\mathcal{P}}(\mathcal{I}) =_{\text{def}} cl_{\mathcal{D},\mathcal{C}}(preST_{\mathcal{P}}(\mathcal{I}))$ where the closure operator $cl_{\mathcal{D},\mathcal{C}}()$ is defined as in Def. 6 and the auxiliary interpretation pre-transformer $preST_{\mathcal{P}}$ acts as follows:*

$$
\begin{array}{l}
preST_{\mathcal{P}}(\mathcal{I}) =_{\text{def}} \{(f(\overline{t}_n)\theta \to t)\sharp d \Leftarrow \Pi \mid \text{ there are} \\
\quad \text{some } (f(\overline{t}_n) \xrightarrow{\alpha} r \Leftarrow \overline{\delta}_m) \in \mathcal{P}, \\
\quad \text{some substitution } \theta, \\
\quad \text{some set } \Pi \text{ of primitive atomic } \mathcal{C}\text{-constraints}, \\
\quad \text{some c-term } t \in Term_{\perp}(\Sigma, B, \mathcal{V}ar), \text{ and} \\
\quad \text{some qualification values } d_0, d_1, \ldots, d_m \in D_{\mathcal{D}} \setminus \{\mathbf{b}\} \text{ such that} \\
\quad - \mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} \delta_i\theta\sharp d_i \Leftarrow \Pi \ (1 \le i \le m), \\
\quad - \mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} (r\theta \to t)\sharp d_0 \Leftarrow \Pi, \text{ and} \\
\quad - d \trianglelefteq \alpha \circ d_i \ (0 \le i \le m) \\
\}.
\end{array}
$$

Proposition 3 below shows that $preST_{\mathcal{P}}(\mathcal{I})$ is closed under $(\mathcal{D},\mathcal{C})$-entailment. Its proof relies on the next technical, but easy result:

Lemma 2 (Auxiliary Result). *Given terms $t, t' \in Term_{\perp}(\Sigma, B, \mathcal{V}ar)$ and a substitution η such that t is linear and $t\eta \sqsubseteq t'$, there is some substitution η' such that:*

1. *$t\eta' = t'$,*
2. *$\eta \sqsubseteq \eta'$ (i.e. $X\eta \sqsubseteq X\eta'$ for all $X \in \mathcal{V}ar$) , and*
3. *$\eta = \eta' \ [\setminus var(t)]$.*

Proof. Since t is linear, for each variable X occurring in t there is one single position p such that X occurs in t at position p. Let p_X be this position. Since $t\theta \sqsubseteq t'$, there must be a subterm t'_X occurring in t' at position p_X such that $X\eta \sqsubseteq t'_X$. Let η' be a substitution such that $X\eta' = t'_X$ for each variable X occurring in t, and $Y\eta' = Y\theta$ for each variable Y not occurring in t. It is easy to check that η' has all the desired properties. □

Proposition 3 ($preST_{\mathcal{P}}(\mathcal{I})$ is closed under $(\mathcal{D},\mathcal{C})$-entailment). *Assume two qc-facts φ and φ'. If $\varphi \in preST_{\mathcal{P}}(\mathcal{I})$ and $\varphi \succcurlyeq_{\mathcal{D},\mathcal{C}} \varphi'$, then $\varphi' \in preST_{\mathcal{P}}(\mathcal{I})$.*

Proof. Since $\varphi \in preST_{\mathcal{P}}(\mathcal{I})$, there are some $R_l : (f(\overline{t}_n) \xrightarrow{\alpha} r \Leftarrow \overline{\delta}_m) \in \mathcal{P}$ and some substitution θ such that $\varphi : (f(\overline{t}_n)\theta \to t)\sharp d \Leftarrow \Pi$ and

- (1) $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} \delta_i\theta\sharp d_i \Leftarrow \Pi$ $(1 \leq i \leq m)$,
- (2) $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} (r\theta \to t)\sharp d_0 \Leftarrow \Pi$, and
- (3) $d \trianglelefteq \alpha \circ d_i$ $(0 \leq i \leq m)$.

Since $\varphi \succcurlyeq_{\mathcal{D},\mathcal{C}} \varphi'$, we can assume $\varphi' : (f(\overline{t'}_n) \to t')\sharp d' \Leftarrow \Pi'$ and a substitution σ such that $t_i\theta\sigma \sqsubseteq t'_i$ $(1 \leq i \leq n)$, $t\sigma \sqsupseteq t'$, (4) $d \trianglerighteq d'$ and $\Pi' \models_{\mathcal{C}} \Pi\sigma$.

Given that $\overline{t}_n$ is a linear tuple of terms, and applying Lemma 2 with $\eta = \theta\sigma$, we obtain a substitution η' satisfying $t_i\eta' = t'_i$ $(1 \leq i \leq n)$, $\theta\sigma \sqsubseteq \eta'$ and $\theta\sigma = \eta'$ $[\backslash \mathrm{var}(\overline{t}_n)]$. Now, in order to prove $\varphi' \in preST_{\mathcal{P}}(\mathcal{I})$ it suffices to consider R_l, η' and some some $d'_0, d'_1, \ldots, d'_m \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ satisfying:

- (1') $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} \delta_i\eta'\sharp d'_i \Leftarrow \Pi'$ $(1 \leq i \leq m)$,
- (2') $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} (r\eta' \to t')\sharp d'_0 \Leftarrow \Pi'$, and
- (3') $d' \trianglelefteq \alpha \circ d'_i$ $(0 \leq i \leq m)$.

Let us see that (1'), (2') and (3') hold when choosing $d'_i = d_i$ $(0 \leq i \leq m)$:

[1'] For any $1 \leq i \leq m$ we have $\delta_i\theta\sharp d_i \Leftarrow \Pi \succcurlyeq_{\mathcal{D},\mathcal{C}} \delta_i\eta'\sharp d_i \Leftarrow \Pi'$ using σ, because $\delta_i\theta\sigma \sqsubseteq \delta_i\eta'$, $d_i \trianglerighteq d_i$ and $\Pi' \models_{\mathcal{C}} \Pi\sigma$. Therefore (1) $\Rightarrow$ (1') by the entailment property (Lemma 1(3)).

[2'] Similarly as for (1'), $(r\theta \to t)\sharp d_0 \Leftarrow \Pi \succcurlyeq_{\mathcal{D},\mathcal{C}} (r\theta' \to t')\sharp d_0\mu \Leftarrow \Pi'$ using σ, because $r\theta\sigma \sqsubseteq r\eta'$, $t\sigma \sqsupseteq t'$, $d_0 \trianglerighteq d_0$ and $\Pi' \models_{\mathcal{C}} \Pi\sigma$. Therefore (2) $\Rightarrow$ (2') again by the entailment property (Lemma 1(3)).

[3'] From (3) and (4) we trivially get $d' \trianglelefteq \alpha \circ d_i$ $(0 \leq i \leq m)$. Therefore, (3') holds when choosing $d'_i = d_i$ $(0 \leq i \leq m)$. □

As a consequence of the previous proposition, we can establish a stronger relation between $ST_{\mathcal{P}}(\mathcal{I})$ and $preST_{\mathcal{P}}(\mathcal{I})$ for non-trivial qc-facts, as given in the following lemma.

Lemma 3 ($ST_{\mathcal{P}}(\mathcal{I})$ versus $preST_{\mathcal{P}}(\mathcal{I})$). *For any non-trivial qc-fact φ one has: $\varphi \in ST_{\mathcal{P}}(\mathcal{I}) \Longrightarrow \varphi \in preST_{\mathcal{P}}(\mathcal{I})$.*

Proof. From $\varphi \in ST_{\mathcal{P}}(\mathcal{I})$ it follows by definition of $ST_{\mathcal{P}}$ that $\varphi \in \mathrm{cl}_{\mathcal{D},\mathcal{C}}(preST_{\mathcal{P}}(\mathcal{I}))$. As we are assuming that φ is not trivial, there must be some $\psi \in preST_{\mathcal{P}}(\mathcal{I})$ such that $\psi \succcurlyeq_{\mathcal{D},\mathcal{C}} \varphi$. Then $\varphi \in preST_{\mathcal{P}}(\mathcal{I})$ follows from Proposition 3. □

The main properties of the interpretation transformer $ST_{\mathcal{P}}$ are given in the following proposition.

Proposition 4 (Properties of interpretation transformers). *Let $\mathcal{P}$ be a QCFLP$(\mathcal{D},\mathcal{C})$-program. Then:*

1. *$ST_{\mathcal{P}}$ is monotonic and continuous.*

2. For any $\mathcal{I} \in Int_{\mathcal{D},\mathcal{C}}$*:* $\mathcal{I} \models_{\mathcal{D},\mathcal{C}} \mathcal{P} \iff ST_{\mathcal{P}}(\mathcal{I}) \subseteq \mathcal{I}$.

Proof. Monotonicity and continuity are well-known results for similar semantics; see e.g. Prop. 3 in [13]. Item 2 can be proved as follows: as an easy consequence of Def. 7, $\mathcal{I} \models_{\mathcal{D},\mathcal{C}} \mathcal{P} \iff preST_{\mathcal{P}}(\mathcal{I}) \subseteq \mathcal{I}$. Moreover, $preST_{\mathcal{P}}(\mathcal{I}) \subseteq \mathcal{I} \iff \mathrm{cl}_{\mathcal{D},\mathcal{C}}(preST_{\mathcal{P}}(\mathcal{I})) \subseteq \mathrm{cl}_{\mathcal{D},\mathcal{C}}(\mathcal{I}) \iff ST_{\mathcal{P}}(\mathcal{I}) \subseteq \mathcal{I}$, where the first equivalence is obvious and the second equivalence is due to the equalities $\mathrm{cl}_{\mathcal{D},\mathcal{C}}(preST_{\mathcal{P}}(\mathcal{I})) = ST_{\mathcal{P}}(\mathcal{I})$ and $\mathrm{cl}_{\mathcal{D},\mathcal{C}}(\mathcal{I}) = \mathcal{I}$. Therefore, $\mathcal{I} \models_{\mathcal{D},\mathcal{C}} \mathcal{P} \iff ST_{\mathcal{P}}(\mathcal{I}) \subseteq \mathcal{I}$, as desired. □

Finally, we can conclude that the least fixpoint of $ST_{\mathcal{P}}$ characterizes the least model of any given QCFLP$(\mathcal{D},\mathcal{C})$-program $\mathcal{P}$, as stated in the following theorem.

Theorem 1. *For every QCFLP*$(\mathcal{D},\mathcal{C})$*-program* $\mathcal{P}$ *there exists the* least model $S_{\mathcal{P}} = lfp(ST_{\mathcal{P}}) = \bigcup_{k\in\mathbb{N}} ST_{\mathcal{P}}{\uparrow^k}(\bot\!\!\!\bot)$.

Proof. Due to a well-known theorem by Knaster and Tarski [22], a monotonic mapping from a complete lattice into itself always has a least fixpoint which is also its least pre-fixpoint. In the case that the mapping is continuous, its least fixpoint can be characterized as the lub of the sequence of lattice elements obtained by reiterated application of the mapping to the bottom element. Combining these results with Prop. 4 trivially proves the theorem. □

A qualified constraint rewriting logic.

In order to obtain a logical view of program semantics and an alternative characterization of least program models, we define the *Qualified Constrained Rewriting Logic for Programs* QCRWL$(\mathcal{D},\mathcal{C})$ as the formal system consisting of the six inference rules displayed in Fig. 3. Note that QCRWL$(\mathcal{D},\mathcal{C})$ is very similar Qualified Constrained Rewriting Logic for Interpretations $\mathcal{I}$-QCRWL$(\mathcal{D},\mathcal{C})$ (see Fig. 2), except that the inference rule $\mathbf{QDF}_{\mathcal{I}}$ from $\mathcal{I}$-QCRWL$(\mathcal{D},\mathcal{C})$ is replaced by the inference rule $\mathbf{QDF}_{\mathcal{P}}$ in QCRWL$(\mathcal{D},\mathcal{C})$. The inference rules in QCRWL$(\mathcal{D},\mathcal{C})$ formalize provability of qc-statements from a given program $\mathcal{P}$ according to their intuitive meanings. In particular, $\mathbf{QDF}_{\mathcal{P}}$ formalizes the behavior of program rules and attenuation factors that was informally explained in the Introduction, using the set $[\mathcal{P}]_{\perp}$ of *program rule instances*.

In the sequel we use the notation $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} \varphi$ to indicate that φ can be inferred from $\mathcal{P}$ in QCRWL$(\mathcal{D},\mathcal{C})$. By convention, we agree that no other inference rule is used whenever **QTI** is applicable. Therefore, trivial qc-statements can only be inferred by rule **QTI**. As usual in formal inference systems, QCRWL$(\mathcal{D},\mathcal{C})$ proofs can be represented as trees whose nodes correspond to inference steps. For example, if $\mathcal{P}$ is the library program, Π is empty, and ψ is

```
(guessGenre(book(4,"Beim Hauten der Zwiebel","Gunter Grass",
    "German","Biography", medium, 432)) --> "Essay")#0.7
```

QTI $\dfrac{}{\varphi}$ if φ is a trivial qc-statement.

QRR $\dfrac{}{(v \to v) \sharp d \Leftarrow \Pi}$ if $v \in \mathcal{V}ar \cup B_{\mathcal{C}}$ and $d \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$.

QDC $\dfrac{(\ (e_i \to t_i) \sharp d_i \Leftarrow \Pi\ )_{i=1\ldots n}}{(c(\overline{e}_n) \to c(\overline{t}_n)) \sharp d \Leftarrow \Pi}$ if $c \in DC^n$ and $d \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ verifies $d \trianglelefteq d_i$ $(1 \leq i \leq n)$.

QDF$_{\mathcal{P}}$ $\dfrac{(\ (e_i \to t_i) \sharp d_i \Leftarrow \Pi\ )_{i=1\ldots n} \quad (r \to t) \sharp d'_0 \Leftarrow \Pi \quad (\delta_j \sharp d'_j \Leftarrow \Pi)_{j=1\ldots m}}{(f(\overline{e}_n) \to t) \sharp d \Leftarrow \Pi}$

if $f \in DF^n$ and $(f(\overline{t}_n) \xrightarrow{\alpha} r \Leftarrow \delta_1, \ldots, \delta_m) \in [\mathcal{P}]_\perp$
where $[\mathcal{P}]_\perp = \{R_l\theta \mid R_l$ is a rule in $\mathcal{P}$ and θ is a substitution$\}$,
and $d \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ verifies $d \trianglelefteq d_i$ $(1 \leq i \leq n)$, $d \trianglelefteq \alpha \circ d'_j$ $(0 \leq j \leq m)$.

QPF $\dfrac{(\ (e_i \to t_i) \sharp d_i \Leftarrow \Pi\ )_{i=1\ldots n}}{(p(\overline{e}_n) \to v) \sharp d \Leftarrow \Pi}$ if $p \in PF^n$, $v \in \mathcal{V}ar \cup DC^0 \cup B_{\mathcal{C}}$,

$\Pi \models_{\mathcal{C}} p(\overline{t}_n) \to v$ and $d \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ verifies $d \trianglelefteq d_i$ $(1 \leq i \leq n)$.

QAC $\dfrac{(\ (e_i \to t_i) \sharp d_i \Leftarrow \Pi\ )_{i=1\ldots n}}{(p(\overline{e}_n) == v) \sharp d \Leftarrow \Pi}$ if $p \in PF^n$, $v \in \mathcal{V}ar \cup DC^0 \cup B_{\mathcal{C}}$,

$\Pi \models_{\mathcal{C}} p(\overline{t}_n) == v$ and $d \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ verifies $d \trianglelefteq d_i$ $(1 \leq i \leq n)$.

Fig. 3. Qualified Constrained Rewriting Logic for Programs

then $\mathcal{P} \vdash_{\mathcal{U},\mathcal{R}} \psi \Leftarrow \Pi$ with a proof tree whose root inference may be chosen as **QDF**$_{\mathcal{P}}$ using a suitable instance of the fourth program rule for `guessGenre`.

The following lemma states the main properties of QCRWL$(\mathcal{D},\mathcal{C})$. The proof is similar to that of Lemma 1 and omitted here. The interested reader is also referred to the proof of Lemma 2 in [13].

Lemma 4 (Some properties of QCRWL$(\mathcal{D},\mathcal{C})$). *The three first items of Lemma 1 also hold for QCRWL$(\mathcal{D},\mathcal{C})$, with the natural reformulation of their statements. More precisely:*

1. *Approximation property: For any non-trivial φ of the form $(t \to t') \sharp d \Leftarrow \Pi$ where $t, t' \in Term_\perp(\Sigma, B, \mathcal{V}ar)$, the three following affirmations are equivalent: (a) $t \sqsupseteq t'$; (b) $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} \varphi$ with an easy proof tree; and (c) $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} \varphi$.*
2. *Primitive c-atoms: For any primitive c-atom $p(\overline{t}_n) == v$, one has $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} (p(\overline{t}_n) == v) \sharp d \Leftarrow \Pi \iff \Pi \models_{\mathcal{C}} p(\overline{t}_n) == v$.*
3. *Entailment property: $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} \varphi$ with a proof tree $\mathcal{T}$ and $\varphi \succcurlyeq_{\mathcal{D},\mathcal{C}} \varphi' \implies \mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} \varphi'$ with a proof tree $\mathcal{T}'$ such that $|\mathcal{T}'| \leq |\mathcal{T}|$.*

The next theorem is the main result in this section. It provides a nice equivalence between QCRWL$(\mathcal{D},\mathcal{C})$-derivability and semantic consequence in the sense

of Definition 7 (*soundness* and *completeness* properties), as well as a characterization of least program models in terms of QCRWL($\mathcal{D},\mathcal{C}$)-derivability (*canonicity property*).

Theorem 2 (QCRWL($\mathcal{D},\mathcal{C}$) characterizes program semantics). *For any QCFLP($\mathcal{D},\mathcal{C}$)-program $\mathcal{P}$ and any qc-statement φ, the following three conditions are equivalent:*

$$(a)\quad \mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} \varphi \qquad (b)\quad \mathcal{P} \models_{\mathcal{D},\mathcal{C}} \varphi \qquad (c)\quad S_{\mathcal{P}} \Vdash_{\mathcal{D},\mathcal{C}} \varphi$$

Moreover, we also have:

1. *Soundness: for any qc-statement φ, $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} \varphi \Longrightarrow \mathcal{P} \models_{\mathcal{D},\mathcal{C}} \varphi$.*
2. *Completeness: for any qc-statement φ, $\mathcal{P} \models_{\mathcal{D},\mathcal{C}} \varphi \Longrightarrow \mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} \varphi$.*
3. *Canonicity: $S_{\mathcal{P}} = \{\varphi \mid \varphi \text{ is a qc-fact and } \mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} \varphi\}$.*

Proof. Assuming the equivalence between (a), (b) and (c), soundness and completeness are a trivial consequence of the equivalence between (a) and (b), and canonicity holds because of the equivalences $\varphi \in S_{\mathcal{P}} \iff S_{\mathcal{P}} \Vdash_{\mathcal{D},\mathcal{C}} \varphi \iff \mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} \varphi$, which follow from the conservation property from Lemma 1 and the equivalence between (c) and (a). The rest of the proof consists of separate proofs for the three implications $(a) \Rightarrow (b)$, $(b) \Rightarrow (c)$ and $(c) \Rightarrow (a)$.

$[(a) \Rightarrow (b)]$ We assume (a), i.e., $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} \varphi$ with a QCRWL($\mathcal{D},\mathcal{C}$)-proof tree $\mathcal{T}_{\mathcal{P}}$ including $k \geq 1$ QCRWL($\mathcal{D},\mathcal{C}$)-inference steps. In order to prove (b) we also assume a qc-interpretation $\mathcal{I}$ such that $\mathcal{I} \models_{\mathcal{D},\mathcal{C}} \mathcal{P}$. We must prove $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} \varphi$ with some QCRWL($\mathcal{D},\mathcal{C}$)-proof tree $\mathcal{T}_{\mathcal{I}}$. This follows easily by induction on k, using the fact that each QCRWL($\mathcal{D},\mathcal{C}$)-inference rule **QRL** is *sound* in the following sense: each inference step

$$\frac{\varphi_1 \ \cdots \ \varphi_n}{\varphi}\ \text{QRL}$$

verifying $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} \varphi_i$ $(1 \leq i \leq n)$ (i.e., the premises are valid in $\mathcal{I}$) also verifies $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} \varphi$ (i.e., the conclusion is valid in $\mathcal{I}$). For **QRL** other than $\mathbf{QDF}_{\mathcal{P}}$, soundness of **QRL** does not depend on the assumption $\mathcal{I} \models_{\mathcal{D},\mathcal{C}} \mathcal{P}$; it can be easily proved by using the homonomous $\mathcal{I}$-QCRWL($\mathcal{D},\mathcal{C}$)-inference rule **QRL**. In the case of $\mathbf{QDF}_{\mathcal{P}}$, φ has the form $f(\bar{e}_n) \to t)\sharp d \Leftarrow \Pi$ and the validity of the premises in $\mathcal{I}$ means the following:

- (1) $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} (e_i \to t_i)\sharp d_i \Leftarrow \Pi$ $(1 \leq i \leq n)$,
- (2) $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} (r \to t)\sharp d'_0 \Leftarrow \Pi$, and
- (3) $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} \delta_j \sharp d'_j \Leftarrow \Pi$ $(1 \leq j \leq m)$

with $f \in DF^n$, $(f(\bar{t}_n) \xrightarrow{\alpha} r \Leftarrow \delta_1, \cdots, \delta_m) \in [\mathcal{P}]_\perp$, $d \trianglelefteq d_i$ $(1 \leq i \leq n)$ and $d \trianglelefteq \alpha \circ d'_j$ $(0 \leq j \leq m)$. Then, from the assumption $\mathcal{I} \models_{\mathcal{D},\mathcal{C}} \mathcal{P}$ and Def. 7 we obtain

- (4) $((f(\bar{t}_n) \to t)\sharp d \Leftarrow \Pi) \in \mathcal{I}$.

Finally, from (1), (4) we conclude that $(f(\overline{e}_n) \to t)\sharp d \Leftarrow \Pi$ can de derived by means of a $\mathbf{QDF}_{\mathcal{I}}$-inference step from premises $(e_i \to t_i)\sharp d_i \Leftarrow \Pi$ $(1 \leq i \leq n)$. Therefore, $\mathcal{I} \Vdash_{\mathcal{D},\mathcal{C}} (f(\overline{e}_n) \to t)\sharp d \Leftarrow \Pi$, as desired.

$[(b) \Rightarrow (c)]$ Straightforward, given that $S_{\mathcal{P}} \models_{\mathcal{D},\mathcal{C}} \mathcal{P}$, as proved in Th. 1.

$[(c) \Rightarrow (a)]$ Let φ be any c-statement and assume $S_{\mathcal{P}} \Vdash_{\mathcal{D},\mathcal{C}} \varphi$ with proof tree $\mathcal{T}$. Note that $\mathcal{T}$ includes a finite number of $\mathbf{QDF}_{\mathcal{I}}$-inference steps with $\mathcal{I} = S_{\mathcal{P}}$, relying on finitely many qc-facts $\psi_i \in S_{\mathcal{P}}$ $(1 \leq i \leq p)$. As $S_{\mathcal{P}} = \bigcup_{k \in \mathbb{N}} ST_{\mathcal{P}} \uparrow^k (\bot\!\!\!\bot)$ because of Th. 1, there must exist some $k \in \mathbb{N}$ such that all the ψ_i $(1 \leq i \leq p)$ belong to $ST_{\mathcal{P}} \uparrow^k (\bot\!\!\!\bot)$ and thus $ST_{\mathcal{P}} \uparrow^k (\bot\!\!\!\bot) \Vdash_{\mathcal{D},\mathcal{C}} \varphi$. Therefore, it is enough to prove by induction on k that

$$ST_{\mathcal{P}} \uparrow^k (\bot\!\!\!\bot) \Vdash_{\mathcal{D},\mathcal{C}} \varphi \Longrightarrow \mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} \varphi$$

Basis (k=0). Assume $ST_{\mathcal{P}} \uparrow^0 (\bot\!\!\!\bot) \Vdash_{\mathcal{D},\mathcal{C}} \varphi$ with $\mathcal{I}$-QCRWL$(\mathcal{D},\mathcal{C})$-proof tree $\mathcal{T}$. As $ST_{\mathcal{P}} \uparrow^0 (\bot\!\!\!\bot) = \bot\!\!\!\bot$, which only includes trivial qc-facts and $\mathbf{QDF}_{\mathcal{I}}$ always uses non-trivial qc-facts, $\mathcal{T}$ cannot include $\mathbf{QDF}_{\mathcal{I}}$-inference steps. Hence, $\mathcal{T}$ also serves as a QCRWL$(\mathcal{D},\mathcal{C})$-proof tree which includes no $\mathbf{QDF}_{\mathcal{P}}$-inference steps and proves $ST_{\mathcal{P}} \uparrow^0 (\bot\!\!\!\bot) \vdash_{\mathcal{D},\mathcal{C}} \varphi$.

Inductive step (k>0). Assume $ST_{\mathcal{P}} \uparrow^{k+1} (\bot\!\!\!\bot) \Vdash_{\mathcal{D},\mathcal{C}} \varphi$ with $\mathcal{I}$-QCRWL$(\mathcal{D},\mathcal{C})$-proof tree $\mathcal{T}$. Then $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} \varphi$ can be proved by an auxiliary induction on the size of $\mathcal{T}$, measured as its number of nodes. The reasoning must distinguish six cases, according to the $\mathcal{I}$-QCRWL$(\mathcal{D},\mathcal{C})$-inference rule **QRL** used to infer φ at the root of $\mathcal{T}$. Here we present only the most interesting case, when **QRL** is $\mathbf{QDF}_{\mathcal{I}}$. In this case, φ is a non-trivial qc-statement of the form $(f(\overline{e}_n) \to t)\sharp d \Leftarrow \Pi$, and $\mathcal{T}$ has the form

$$\frac{(\ (e_i \to t_i)\sharp d_i \Leftarrow \Pi\ )_{i=1\ldots n}}{\varphi : (f(\overline{e}_n) \to t)\sharp d \Leftarrow \Pi}\ \mathrm{QDF}_{\mathcal{I}}$$

with non-trivial $\psi : ((f(\overline{t}_n) \to t)\sharp d_0 \Leftarrow \Pi) \in ST_{\mathcal{P}} \uparrow^{k+1} (\bot\!\!\!\bot)$, $d \trianglelefteq d_i$ $(0 \leq i \leq n)$, and $ST_{\mathcal{P}} \uparrow^{k+1} (\bot\!\!\!\bot) \Vdash_{\mathcal{D},\mathcal{C}} (e_i \to t_i)\sharp d_i \Leftarrow \Pi$ proved by $\mathcal{I}$-QCRWL$(\mathcal{D},\mathcal{C})$-proof trees $\mathcal{T}_i$ wit sizes smaller than the size of $\mathcal{T}$ $(1 \leq i \leq n)$. Therefore, the inductive hypothesis of the nested induction guarantees

- (1) $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} (e_i \to t_i)\sharp d_i \Leftarrow \Pi$ with QCRWL$(\mathcal{D},\mathcal{C})$-proof trees $\hat{\mathcal{T}}_i$ $(1 \leq i \leq n)$

On the other hand, Lemma 3 ensures $\psi \in preST_{\mathcal{P}}(ST_{\mathcal{P}} \uparrow^k (\bot\!\!\!\bot))$. Therefore, recalling Def. 8, there must exist $f(\overline{s}_n) \xrightarrow{\alpha} r \Leftarrow \overline{\delta}_m \in \mathcal{P}$, a substitution θ and qualification values $d'_0, d'_1, \ldots, d'_m$ satisfying $s_i\theta = t_i$ $(1 \leq i \leq n)$ and

- (2) $ST_{\mathcal{P}} \uparrow^k (\bot\!\!\!\bot) \Vdash_{\mathcal{D},\mathcal{C}} \delta_j\theta\sharp d'_j \Leftarrow \Pi$ $(1 \leq j \leq m)$
- (3) $ST_{\mathcal{P}} \uparrow^k (\bot\!\!\!\bot) \Vdash_{\mathcal{D},\mathcal{C}} (r\theta \to t)\sharp d'_0 \Leftarrow \Pi$
- (4) $d_0 \trianglelefteq \alpha \circ d'_j$ $(0 \leq j \leq m)$

By the inductive hypothesis of the main induction, applied to (2) and (3), we get:

- (5) $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} \delta_j\theta\sharp d'_j \Leftarrow \Pi$ with QCRWL$(\mathcal{D},\mathcal{C})$-proof trees $\hat{\mathcal{T}'}_j$ $(1 \leq j \leq m)$

– (6) $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} (r\theta \to t)\sharp d'_0 \Leftarrow \Pi$ with QCRWL$(\mathcal{D},\mathcal{C})$-proof tree $\hat{\mathcal{T}}'$

From $d \trianglelefteq d_i$ $(0 \le i \le n)$ and (4) we also obtain:

– (7) $d \trianglelefteq d_i$ $(0 \le i \le n)$, $d \trianglelefteq \alpha \circ d'_j$ $(0 \le j \le m)$

Finally, we can prove $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} \varphi$ with a QCRWL$(\mathcal{D},\mathcal{C})$-proof tree $\hat{\mathcal{T}}$ of the form:

$$\frac{((e_i \to s_i\theta)\sharp d_i \Leftarrow \Pi)_{i=1\ldots n} \quad (r\theta \to t)\sharp d'_0 \Leftarrow \Pi \quad (\delta_j\theta\sharp d'_j \Leftarrow \Pi)_{j=1\ldots m}}{\varphi : (f(\overline{e}_n) \to t)\sharp d \Leftarrow \Pi}\ \mathrm{QDF}_{\mathcal{P}}$$

using the program rule instance $(f(\overline{s}_n) \xrightarrow{\alpha} r \Leftarrow \overline{\delta}_m)\theta \in [\mathcal{P}]_\perp$, where (5) and (6) provide proof trees for deriving the premises and (7) ensures the additional conditions required by the $\mathbf{QDF}_{\mathcal{P}}$ inference at the root of $\hat{\mathcal{T}}$. □

3.3 Goals and their Solutions

In all declarative programming paradigms, programs are generally used by placing goals and computing answers for them. In this brief subsection we define the syntax of QCFLP$(\mathcal{D},\mathcal{C})$-goals and we give a declarative characterization of goal solutions, based on the QCRWL$(\mathcal{D},\mathcal{C})$ logic. This will allow formal proofs of correctness for the goal solving methods presented in Section 4.

Definition 9 (QCFLP$(\mathcal{D},\mathcal{C})$-Goals and their Solutions). *Assume a a countable set $\mathcal{V}ar$ of so-called* qualification variables *W, disjoint from $\mathcal{V}ar$ and $\mathcal{C}$'s signature Σ, and a QCFLP$(\mathcal{D},\mathcal{C})$-program $\mathcal{P}$. Then:*

1. *A* goal *G for $\mathcal{P}$ has the form $\delta_1\sharp W_1, \ldots, \delta_m\sharp W_m \,[\!]\, W_1 \trianglerighteq \beta_1, \ldots, W_m \trianglerighteq \beta_m$, abbreviated as $(\ \delta_i\sharp W_i,\ W_i \trianglerighteq \beta_i\ )_{i=1\ldots m}$, where $\delta_j\sharp W_j$ $(1 \le j \le m)$ are atomic $\mathcal{C}$-constraints annotated with different qualification variables W_i, and $W_i \trianglerighteq \beta_i$ are so-called* threshold conditions*, with $\beta_i \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ $(1 \le i \le m)$.*
2. *A* solution *for G is any triple $\langle \sigma, \mu, \Pi \rangle$ such that σ is a substitution, μ is a $\mathcal{D}$-valuation, Π is a finite set of atomic primitive $\mathcal{C}$-constraints, and the following two conditions hold for all $1 \le i \le m$: $W_i\mu = d_i \trianglerighteq \beta_i$, and $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} (\delta_i\sigma)\sharp d_i \Leftarrow \Pi$. The set of all solutions for G is noted $Sol_{\mathcal{P}}(G)$.* □

Thanks to the *Canonicity* property of Theorem 2, solutions of $\mathcal{P}$ are valid in the least model $S_{\mathcal{P}}$ and hence in all models of $\mathcal{P}$. A goal for the library program and one solution for it have been presented in the Introduction. In this particular example, $\Pi = \emptyset$ and the QCRWL$(\mathcal{U},\mathcal{R})$ proof needed to check the solution according to Definition 9 can be formalized by following the intuitive ideas sketched in the Introduction.

4 Implementation by Program Transformation

Goal solving in instances of the CFLP$(\mathcal{C})$ scheme from [13] has been formalized by means of *constrained narrowing* procedures as e.g. [12,5], and is supported

by systems such as Curry [9] and $\mathcal{TOY}$ [3]. In this section we present a semantically correct transformation from QCFLP($\mathcal{D},\mathcal{C}$) into the first-order fragment of CFLP($\mathcal{C}$) which can be used for implementing goal solving in QCFLP($\mathcal{D},\mathcal{C}$).

By abuse of notation, the first-order fragment of the CFLP($\mathcal{C}$) scheme will be noted simply as CFLP($\mathcal{C}$) in the sequel. A formal description of CFLP($\mathcal{C}$) can be found in [13]; it is easily derived from the previous Section 3 by simply omitting everything related to qualification domains and values. Programs $\mathcal{P}$ are sets of program rules of the form $f(\bar{t}_n) \to r \Leftarrow \Delta$, with no attenuation factors attached. Program semantics relies on inference mechanisms for deriving *c-staments* from programs. In analogy to Def. 4, a c-statement φ may be a c-production $e \to t \Leftarrow \Pi$ or a c-atom $\delta \Leftarrow \Pi$. In analogy to Def. 6, *c-interpretations* are defined as sets of c-statements closed under a $\mathcal{C}$-entailment relation. Program models and semantic consequence are defined similarly as in Def. 7. Results similar to Th. 1 and Th. 2 can be obtained to characterize program semantics in terms of an interpretation transformer and a rewriting logic CRWL($\mathcal{C}$), respectively.

For the purposes of this section it is enough to focus on CRWL($\mathcal{C}$), which is a formal system consisting of the six inference rules displayed in Fig. 4. They are quite similar to the QCRWL($\mathcal{D},\mathcal{C}$)-inference rules from Fig. 3, except that attenuation factors and qualification values are absent.

TI $\dfrac{}{\varphi}$ if φ is a trivial c-statement.

RR $\dfrac{}{v \to v \Leftarrow \Pi}$ if $v \in \mathcal{V}ar \cup B_{\mathcal{C}}$.

DC $\dfrac{(\ e_i \to t_i \Leftarrow \Pi\ )_{i=1\ldots n}}{c(\bar{e}_n) \to c(\bar{t}_n) \Leftarrow \Pi}$ if $c \in DC^n$.

DF$_{\mathcal{P}}$ $\dfrac{(\ e_i \to t_i \Leftarrow \Pi\ )_{i=1\ldots n} \quad r \to t \Leftarrow \Pi \quad (\ \delta_j \Leftarrow \Pi\ )_{j=1\ldots m}}{f(\bar{e}_n) \to t \Leftarrow \Pi}$

if $f \in DF^n$ and $(f(\bar{t}_n) \xrightarrow{\alpha} r \Leftarrow \delta_1, \ldots, \delta_m) \in [\mathcal{P}]_\perp$
where $[\mathcal{P}]_\perp = \{R_l\theta \mid R_l$ is a rule in $\mathcal{P}$ and θ is a substitution$\}$.

PF $\dfrac{(\ e_i \to t_i \Leftarrow \Pi\ )_{i=1\ldots n}}{p(\bar{e}_n) \to v \Leftarrow \Pi}$ if $p \in PF^n$, $v \in \mathcal{V}ar \cup DC^0 \cup B_{\mathcal{C}}$ and $\Pi \models_{\mathcal{C}} p(\bar{t}_n) \to v$.

AC $\dfrac{(\ e_i \to t_i \Leftarrow \Pi\ )_{i=1\ldots n}}{p(\bar{e}_n) == v \Leftarrow \Pi}$ if $p \in PF^n$, $v \in \mathcal{V}ar \cup DC^0 \cup B_{\mathcal{C}}$ and $\Pi \models_{\mathcal{C}} p(\bar{t}_n) == v$.

Fig. 4. First Order Constrained Rewriting Logic

The notation $\mathcal{P} \vdash_{\mathcal{C}} \varphi$ indicates that φ can be inferred from $\mathcal{P}$ in CRWL($\mathcal{C}$). In analogy to the Canonicity Property from Th. 2, it is possible to prove that

the least model of $\mathcal{P}$ w.r.t. set inclusion can be characterized as $S_{\mathcal{P}} = \{\varphi \mid \varphi \text{ is a c-fact and } \mathcal{P} \vdash_{\mathcal{C}} \varphi\}$. Therefore, working with formal inference in the rewrite logics QCRWL$(\mathcal{D},\mathcal{C})$ and CRWL$(\mathcal{C})$ is sufficient for proving the semantic correctness of the transformations presented in the rest of this section.

The following definition is similar to Def. 9. It will be useful for proving the correctness of the goal solving procedure for QCFLP$(\mathcal{D},\mathcal{C})$-goals discussed in the final part of this section.

Definition 10 (CFLP$(\mathcal{C})$-Goals and their Solutions). *Assume a CFLP$(\mathcal{C})$-program $\mathcal{P}$. Then:*

1. *A* goal *G for $\mathcal{P}$ has the form $\delta_1, \ldots, \delta_m$ where δ_j are atomic $\mathcal{C}$-constraints.*
2. *A* solution *for G is any pair $\langle \sigma, \Pi \rangle$ such that σ is a substitution, Π is a finite set of atomic primitive $\mathcal{C}$-constraints, and $\mathcal{P} \vdash_{\mathcal{C}} \delta_j\sigma \Leftarrow \Pi$ holds for $1 \leq j \leq m$. The set of all solutions for G is noted $Sol_{\mathcal{P}}(G)$.* □

Now we are ready to describe a semantically correct transformation from QCFLP$(\mathcal{D},\mathcal{C})$ into CFLP$(\mathcal{C})$. The transformation goes from a source signature Σ into a target signature Σ' such that each $f \in DF^n$ in Σ becomes $f' \in DF^{n+1}$ in Σ', and all the other symbols in Σ remain the same in Σ'. There are four group of transformation rules displayed in Figure 5 and designed to transform expressions, qc-statements, program rules and goals, respectively. The transformation works by introducing fresh qualification variables W to represent the qualification values attached to the results of calls to defined functions, as well as qualification constraints to be imposed on the values of qualification variables. Let us comment the four groups of rules in order.

Transforming any expression e yields a triple $e^{\mathcal{T}} = (e', \Omega, \mathcal{W})$, where Ω is a set of qualification constraints and $\mathcal{W}$ is the set of qualification variables occurring in e' at outermost positions. This set is relevant because the qualification value attached to e cannot exceed the infimum in $\mathcal{D}$ of the values of the variables $W \in \mathcal{W}$, and $e^{\mathcal{T}}$ is computed by recursion on e's syntactic structure as specified by the transformation rules **TAE**, **TCE**$_1$ and **TCE**$_2$. Note that **TCE**$_2$ introduces a new qualification variable W for each call to a defined function $f \in DF^n$ and builds a set Ω' of qualification constraints ensuring that W must be interpreted as a qualification value not greater than the qualification values attached to f's arguments. **TCE**$_1$ deals with calls to constructors and primitive functions just by collecting information from the arguments, and **TAE** is self-explanatory.

Unconditional productions and atomic constraints are transformed by means of **TP** and **TA**, respectively, relying on the transformation of expressions in the obvious way. Relying on **TP** and **TA**, **TCS** transforms any qc-statement of the form $\psi \sharp d \Leftarrow \Pi$ into a c-statement whose conditional part includes, in addition to Π, the qualification constraints Ω coming from $\psi^{\mathcal{T}}$ and extra qualification constraints ensuring that d is not greater than allowed by ψ's qualification.

Program rules are transformed by **TPR**. Transforming the left-hand side $f(\bar{t}_n)$ introduces a fresh symbol $f' \in DF^{n+1}$ and a fresh qualification variable W. The transformed right-hand side r' comes from $r^{\mathcal{T}}$, and the transformed conditions are obtained from the constraints coming from $r^{\mathcal{T}}$ and ${\delta_i}^{\mathcal{T}} (1 \leq i \leq m)$

Transforming Expressions

TAE $\dfrac{}{v^{\mathcal{T}} = (v, \emptyset, \emptyset)}$ if $v \in \mathcal{V}ar \cup B_{\mathcal{C}}$.

TCE$_1$ $\dfrac{(\ e_i{}^{\mathcal{T}} = (e'_i, \Omega_i, \mathcal{W}_i)\ )_{i=1\ldots n}}{h(\overline{e}_n)^{\mathcal{T}} = (h(\overline{e'}_n), \bigcup_{i=1}^n \Omega_i, \bigcup_{i=1}^n \mathcal{W}_i)}$ if $h \in DC^n \cup PF^n$.

TCE$_2$ $\dfrac{(\ e_i{}^{\mathcal{T}} = (e'_i, \Omega_i, \mathcal{W}_i)\ )_{i=1\ldots n}}{f(\overline{e}_n)^{\mathcal{T}} = (f'(\overline{e'}_n, W), \Omega', \{W\})}$

if $f \in DF^n$ and W is a fresh variable,
where $\Omega' = (\bigcup_{i=1}^n \Omega_i) \cup \{\mathsf{qVal}(W)\} \cup \{\ulcorner W \trianglelefteqslant W' \urcorner \mid W' \in \bigcup_{i=1}^n \mathcal{W}_i\}$.

Transforming qc-Statements

TP $\dfrac{e^{\mathcal{T}} = (e', \Omega, \mathcal{W})}{(e \to t)^{\mathcal{T}} = (e' \to t,\ \Omega, \mathcal{W})}$

TA $\dfrac{(\ e_i{}^{\mathcal{T}} = (e'_i, \Omega_i, \mathcal{W}_i)\ )_{i=1\ldots n}}{(p(\overline{e}_n) == v)^{\mathcal{T}} = (\ p(\overline{e'}_n) == v, \bigcup_{i=1}^n \Omega_i, \bigcup_{i=1}^n \mathcal{W}_i\ )}$

if $p \in PF^n$, $v \in \mathcal{V}ar \cup DC^0 \cup B_{\mathcal{C}}$.

TCS $\dfrac{\psi^{\mathcal{T}} = (\psi', \Omega, \mathcal{W})}{(\psi \sharp d \Leftarrow \Pi)^{\mathcal{T}} = (\psi' \Leftarrow \Pi, \Omega \cup \{\ulcorner d \trianglelefteqslant W \urcorner \mid W \in \mathcal{W}\}))}$

if ψ is of the form $e \to t$ or $p(\overline{e}_n) == v$ and $d \in D_{\mathcal{D}}$.

Transforming Program Rules

TPR
$$\frac{r^{\mathcal{T}} = (r', \Omega_r, \mathcal{W}_r) \qquad (\ \delta_i{}^{\mathcal{T}} = (\delta'_i, \Omega_i, \mathcal{W}_i)\ )_{i=1\ldots m}}{\begin{array}{l}(f(\overline{t}_n) \xrightarrow{\alpha} r \Leftarrow \delta_1, \ldots, \delta_m)^{\mathcal{T}} = \\ \quad f'(\overline{t}_n, W) \to r' \Leftarrow \mathsf{qVal}(W),\ \Omega_r,\ (\ulcorner W \trianglelefteqslant \alpha \circ W' \urcorner)_{W' \in \mathcal{W}_r},\\ \qquad\qquad (\ \Omega_i,\ (\ulcorner W \trianglelefteqslant \alpha \circ W' \urcorner)_{W' \in \mathcal{W}_i},\ \delta'_i\ )_{i=1\ldots m}\end{array}}$$

where W is a fresh variable.

Transforming Goals

TG
$$\frac{(\ \delta_i{}^{\mathcal{T}} = (\delta'_i, \Omega'_i, \mathcal{W}'_i)\ )_{i=1\ldots m}}{\begin{array}{l}((\ \delta_i \sharp W_i, W_i \trianglerighteq \beta_i\ )_{i=1\ldots m})^{\mathcal{T}} = \\ \quad (\ \Omega'_i,\ \mathsf{qVal}(W_i),\ (\ulcorner W_i \trianglelefteqslant W' \urcorner)_{W' \in \mathcal{W}'_i},\ \ulcorner W_i \trianglerighteq \beta_i \urcorner,\ \delta'_i\ )_{i=1\ldots m}\end{array}}$$

Fig. 5. Transformation rules

by adding extra qualification constraints to be imposed on W, namely $\mathsf{qVal}(W)$ and $(\ulcorner W \trianglelefteqslant \alpha \circ W' \urcorner)_{W' \in \mathcal{W}'}$, for $\mathcal{W}' = \mathcal{W}_r$ and $\mathcal{W}' = \mathcal{W}_i$ $(1 \leq i \leq m)$. By convention, $(\ulcorner W \trianglelefteqslant \alpha \circ W' \urcorner)_{W' \in \mathcal{W}'}$ is understood as $\ulcorner W \trianglelefteqslant \alpha \urcorner$ in case that $\mathcal{W}' = \emptyset$. The idea is that W's value cannot exceed the infimum in $\mathcal{D}$ of all the values $\alpha \circ \beta$, for the different β coming from the qualifications of r and δ_i $(1 \leq i \leq m)$.

Finally, **TG** transforms a goal $(\delta_i \sharp W_i, W_i \trianglerighteq \beta_i)_{i=1...m}$ by transforming each atomic constraint δ_i and adding $\mathsf{qVal}(W_i)$, $(\ulcorner W_i \trianglelefteqslant W' \urcorner)_{W' \in \mathcal{W}'_i}$ and $\ulcorner W_i \trianglerighteq \beta_i \urcorner$ $(1 \leq i \leq m)$ to ensure that each W_i is interpreted as a qualification value not bigger than the qualification computed for δ_i and satisfying the threshold condition $W_i \trianglerighteq \beta_i$. In case that $\mathcal{W}'_i = \emptyset$, $(\ulcorner W_i \trianglelefteqslant W' \urcorner)_{W' \in \mathcal{W}'_i}$ is understood as $\ulcorner W_i \trianglelefteqslant \mathbf{t} \urcorner$.

The result of applying **TPR** to all the program rules of a program $\mathcal{P}$ will be noted as $\mathcal{P}^{\mathcal{T}}$. The following theorem proves that QCRWL$(\mathcal{D},\mathcal{C})$-derivability from $\mathcal{P}$ corresponds to CRWL$(\mathcal{C})$-derivability from $\mathcal{P}^{\mathcal{T}}$. Since program semantics in QCFLP$(\mathcal{D},\mathcal{C})$ and in CFLP$(\mathcal{C})$ is characterized by, respectively, derivability in QCRWL$(\mathcal{D},\mathcal{C})$ and in CRWL$(\mathcal{C})$, the program transformation is semantically correct. The theorem uses an auxiliary lemma we are proving first which indicates that the constraints obtained when transforming a qc-statement always admit a solution.

Lemma 5. *Let $\varphi = \psi \sharp d \Leftarrow \Pi$ be a qc-statement such that $\varphi^{\mathcal{T}} = (\psi' \Leftarrow \Pi, \Omega')$. Then exists $\rho : var(\Omega') \to D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ solution of Ω'.*

Proof. $\varphi^{\mathcal{T}}$ is obtained by the transformation rule **TCS** of Figure 5. This rule needs to obtain $\psi^{\mathcal{T}}$ which can be done using either the transformation rule **TP** or **TA** of the same figure. In the case of using **TP**, ψ must be of the form $(e \to t)$ and Ω' will be of the form $\Omega \cup \{\ulcorner d \trianglelefteqslant W \urcorner \mid W \in \mathcal{W}\}$, with $\Omega, \mathcal{W}$ such that $e^{\mathcal{T}} = (e', \Omega, \mathcal{W})$. Checking the transformation rules for expressions (again Figure 5) we see that Ω is a set of constraints where each element is either of the form $\ulcorner W \trianglelefteqslant W' \urcorner$ or $\mathsf{qVal}(W)$, with $W, W' \in \mathcal{V}ar$. Then ρ can be defined assigning $\mathbf{t}$ to every variable W occurring in either Ω' or $\mathcal{W}$. The case corresponding to the transformation rule **TA** is analogous. □

Theorem 3. *Let $\mathcal{P}$ be a QCFLP$(\mathcal{D},\mathcal{C})$-program and $\psi \sharp d \Leftarrow \Pi$ a qc-statement such that $(\psi \sharp d \Leftarrow \Pi)^{\mathcal{T}} = (\psi' \Leftarrow \Pi, \Omega')$. Then the two following statements are equivalent:*

1. *$\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} \psi \sharp d \Leftarrow \Pi$.*
2. *$\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} \psi' \rho \Leftarrow \Pi$ for some $\rho \in Sol_{\mathcal{C}}(\Omega')$ such that $vdom(\rho) = var(\Omega')$.*

Proof. We prove the equivalence separately proving each implication.

[*1. ⇒ 2.*] (*Transformation completeness*). Assume $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} \psi \sharp d \Leftarrow \Pi$ by means of a QCRWL$(\mathcal{D},\mathcal{C})$ proof tree T with k nodes. By induction on k we show the existence of a CRWL$(\mathcal{C})$ proof tree T' witnessing $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} \psi' \rho \Leftarrow \Pi$ for some $\rho \in \mathrm{Sol}_{\mathcal{C}}(\Omega')$ such that $\mathrm{vdom}(\rho) = \mathrm{var}(\Omega')$.

Basis (k=1). If T contains only one node the QCRWL$(\mathcal{D},\mathcal{C})$ inference step applied at the root must be one of the following:

- **QTI**. In this case $\psi\sharp d \Leftarrow \Pi$ is a trivial qc-statement, and we take ρ as the substitution defined in Lemma 5. By Def. 4, $\psi\sharp d \Leftarrow \Pi$ trivial implies either $\psi = e \to \bot$ or $\mathrm{Unsat}_{\mathcal{C}}(\Pi)$. In the first case $\psi' = e' \to \bot$ and therefore $\psi'\rho \Leftarrow \Pi$ is trivial. Analogously, if $\mathrm{Unsat}_{\mathcal{C}}(\Pi)$ then $\psi'\rho \Leftarrow \Pi$ is trivial as well. Hence T' consists of a single node $\psi'\rho \Leftarrow \Pi$ with a **TI** inference step at its root.
- **QRR**. In this case $\psi = t \to t$ for some $t \in \mathcal{V}ar \cup \mathcal{B}_{\mathcal{C}}$, and $(\psi\sharp d \Leftarrow \Pi)^{\mathcal{T}} = (t \to t \Leftarrow \Pi, \emptyset)$ (applying the transformation rules **TCS**, **TP** and **TAE** to obtain $t^{\mathcal{T}} = (t, \emptyset, \emptyset)$). Therefore ρ can be defined as the identity substitution and prove $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} \psi'\rho \Leftarrow \Pi$ by using a single **RR** inference step.
- **QDC**. In this case $\psi = c \to c$ and $(\psi\sharp d \Leftarrow \Pi)^{\mathcal{T}} = (c \to c \Leftarrow \Pi, \emptyset)$ (applying the transformation rules **TCS**, **TP** and $\mathbf{TCE}_1$ for $c^{\mathcal{T}} = (c, \emptyset, \emptyset)$). Therefore ρ can be defined as the identity substitution and prove $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} \psi'\rho \Leftarrow \Pi$ by using a single **DC** inference step.

Inductive step (k>1). The QCRWL($\mathcal{D}, \mathcal{C}$) inference step applied at the root must be one of the following:

- **QDC**. In this case $\psi = c(\overline{e}_n) \to c(\overline{t}_n)$ and the first inference step is of the form
$$\frac{(\ (e_i \to t_i)\sharp d_i \Leftarrow \Pi\ )_{i=1\ldots n}}{(c(\overline{e}_n) \to c(\overline{t}_n))\sharp d \Leftarrow \Pi}$$
with $d \trianglelefteq d_i$ $(1 \le i \le n)$. In order to obtain $\psi\sharp d \Leftarrow \Pi^{\mathcal{T}}$ we apply the transformation rules as follows:
  - By the transformation rule $\mathbf{TCE}_1$,
$$c(\overline{e}_n)^{\mathcal{T}} = (c(\overline{e'}_n), \bigcup_{i=1}^{n} \Omega_i, \bigcup_{i=1}^{n} \mathcal{W}_i)$$
with ${e_i}^{\mathcal{T}} = (e'_i, \Omega_i, \mathcal{W}_i)$ for $i = 1 \ldots n$.
  - By **TP** and with the result of the previous step,
$$\psi^{\mathcal{T}} = (c(\overline{e}_n) \to c(\overline{t}_n))^{\mathcal{T}} = (c(\overline{e'}_n) \to c(\overline{t}_n), \bigcup_{i=1}^{n} \Omega_i, \bigcup_{i=1}^{n} \mathcal{W}_i)\ .$$
  - And finally from $\psi^{\mathcal{T}}$ and by **TCS**,
$$(\psi\sharp d \Leftarrow \Pi)^{\mathcal{T}} = (c(\overline{e'}_n) \to c(\overline{t}_n) \Leftarrow \Pi, \Omega')\ ,$$
with
$$\Omega' = \bigcup_{i=1}^{n} \Omega_i \cup \{\ulcorner d \trianglelefteq W \urcorner \mid W \in \bigcup_{i=1}^{n} \mathcal{W}_i\}\ .$$
From the premises $(\ (e_i \to t_i)\sharp d_i \Leftarrow \Pi\ )_{i=1\ldots n}$ of the **QDC** step and by the induction hypothesis we have that $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} (e'_i \to t_i)\rho_i \Leftarrow \Pi$, $i = 1 \ldots n$ for some substitutions $\rho_i : \mathrm{var}(\Omega'_i) \to D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ solution of
$$\Omega'_i = \Omega_i \cup \{\ulcorner d_i \trianglelefteq W \urcorner \mid W \in \mathcal{W}_i\}$$

for $i = 1 \ldots n$. Since $\mathrm{var}(\Omega'_i) \cap \mathrm{var}(\Omega'_j) = \emptyset$ for every $1 \leq i, j \leq n$, $i \neq j$, and $\mathrm{var}(\Omega') = \bigcup_{i=1}^{n} \mathrm{var}(\Omega'_i)$, we can define a new substitution $\rho : \mathrm{var}(\Omega') \to D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ as $\rho = \biguplus_{i=1}^{n} \rho_i$. It is easy to check that ρ is solution of Ω':

- It is solution of every Ω'_i for $i = 1 \ldots n$, since $\rho{\restriction}\mathrm{var}(\Omega'_i) = \rho_i$. Therefore it is solution of $\bigcup_{i=1}^{n} \Omega_i$.
- It is a solution of $\{\ulcorner d \trianglelefteqslant W \urcorner \mid W \in \bigcup_{i=1}^{n} \mathcal{W}_i\}$ because as solution of Ω'_i for $i = 1 \ldots n$, ρ is solution of $\{\ulcorner d_i \trianglelefteqslant W \urcorner \mid W \in \mathcal{W}_i\}$, and by the hypothesis of **QDC** $d \trianglelefteqslant d_i$.

Therefore we prove $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} (c(\overline{e'}_n)\rho \to c(\overline{t}_n))\rho \Leftarrow \Pi$ with a proof tree T' which starts with a **DC** inference rule of the form

$$\frac{(\,(\, e'_i \to t_i)\rho \Leftarrow \Pi\,)_{i=1\ldots n}}{(c(\overline{e'}_n) \to c(\overline{t}_n))\rho \Leftarrow \Pi}\ .$$

In order to justify that $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} (e'_i \to t_i)\rho \Leftarrow \Pi$ for each $i = 1 \ldots n$, we observe that the only variables of $e'_i \to t_i$ that can be affected by ρ are those introduced in e'_i by the transformation, and that therefore $(e'_i \to t_i)\rho = (e'_i \to t_i)\rho_i$ for $i = 1 \ldots n$, and these premises correspond to the inductive hypotheses of this case.

– **QDF**$_{\mathcal{P}}$. In this case $\psi = f(\overline{e}_n) \to t$ and the inference step applied at the root is of the form

$$\frac{(\,(e_i \to t_i\theta)\sharp d_i \Leftarrow \Pi\,)_{i=1\ldots n} \quad (r\theta \to t)\sharp d'_0 \Leftarrow \Pi \quad (\,\delta_j\theta\sharp d'_j \Leftarrow \Pi\,)_{j=1\ldots m}}{(f(\overline{e}_n) \to t)\sharp d \Leftarrow \Pi}$$

for some program rule $R_l = (f(\overline{t}_n) \xrightarrow{\alpha} r \Leftarrow \overline{\delta}_m) \in \mathcal{P}$ and substitution θ such that $R_l\theta \in [\mathcal{P}]_{\perp}$, and with $d \trianglelefteqslant d_i$ $(1 \leq i \leq n)$ and $d \trianglelefteqslant \alpha \circ d'_j$ $(0 \leq j \leq m)$.

The inductive hypotheses in this case are:

1. $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} (e'_i \to t_i\theta)\rho_i \Leftarrow \Pi$ for $i = 1 \ldots n$, with ${e_i}^{\mathcal{T}} = (e'_i, \Omega_i, \mathcal{W}_i)$ and ρ_i solution of $\Omega'_i = \Omega_i \cup \{\ulcorner d_i \trianglelefteqslant W' \urcorner \mid W' \in \mathcal{W}_i\}$, for $i = 1 \ldots n$.
2. $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} (r'\theta \to t)\rho'_0 \Leftarrow \Pi$, with ${r}^{\mathcal{T}} = (r', \Omega_r, \mathcal{W}'_0)$ (it is easy to check that if ${r}^{\mathcal{T}} = (r', \Omega_r, \mathcal{W}'_0)$ then ${(r\theta)}^{\mathcal{T}} = (r'\theta, \Omega_r, \mathcal{W}'_0)$ for every substitution θ), and ρ'_0 solution of $\Omega'_r = \Omega_r \cup \{\ulcorner d'_0 \trianglelefteqslant W' \urcorner \mid W' \in \mathcal{W}'_0\}$.
3. $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} (\delta'_j\theta)\rho'_j \Leftarrow \Pi$ with ${\delta_j}^{\mathcal{T}} = (\delta'_j, \Omega_{\delta_j}, \mathcal{W}'_j)$ for $j = 1 \ldots k$ (it is easy to check that if ${\delta_j}^{\mathcal{T}} = (\delta'_j, \Omega_{\delta_j}, \mathcal{W}'_j)$ then ${(\delta_j\theta)}^{\mathcal{T}} = (\delta'_j\theta, \Omega_{\delta_j}, \mathcal{W}'_j)$ for every substitution θ and $j = 1 \ldots k$). The substitution ρ'_j is solution of $\Omega'_{\delta_j} = \Omega_{\delta_j} \cup \{\ulcorner d'_j \trianglelefteqslant W' \urcorner \mid W' \in \mathcal{W}'_j\}$ for $j = 1 \ldots m$.

In this case, $(\psi\sharp d \Leftarrow \Pi)^{\mathcal{T}}$ is obtained by means of the transformation rule **TCS**. This rule asks first for the transformation of the qualified statement $(f(\overline{e}_n) \to t)\sharp d$, which can be obtained by rule **TP**, and this one requires the transformation of $f(\overline{e}_n)$, provided by rule rule **TCE**$_2$. Let's see it:

$$\frac{(\ e_i{}^{\mathcal{T}} = (e'_i, \Omega_i, \mathcal{W}_i)\ )_{i=1\ldots n}}{f(\overline{e}_n)^{\mathcal{T}} = (\ f(\overline{e'}_n, W),\ (\bigcup_{i=1}^n \Omega_i) \cup \{\mathsf{qVal}(W)\} \cup \{\ulcorner W \trianglelefteqslant W' \urcorner \mid W' \in \bigcup_{i=1}^n \mathcal{W}_i\},\ \{W\}\ )}\ \mathbf{TCE}_2$$

$$\frac{}{(f(\overline{e}_n) \to t)^{\mathcal{T}} = (\ f(\overline{e'}_n, W) \to t,\ (\bigcup_{i=1}^n \Omega_i) \cup \{\mathsf{qVal}(W)\} \cup \{\ulcorner W \trianglelefteqslant W' \urcorner \mid W' \in \bigcup_{i=1}^n \mathcal{W}_i\},\ \{W\}\ )}\ \mathbf{TP}$$

$$\frac{}{((f(\overline{e}_n) \to t)\sharp d \Leftarrow \Pi)^{\mathcal{T}} = (\ f(\overline{e'}_n, W) \to t \Leftarrow \Pi,\ (\bigcup_{i=1}^n \Omega_i) \cup \{\mathsf{qVal}(W)\} \cup \{\ulcorner W \trianglelefteqslant W' \urcorner \mid W' \in \bigcup_{i=1}^n \mathcal{W}_i\} \cup \{\ulcorner d \trianglelefteqslant W \urcorner\}\ )}\ \mathbf{TCS}$$

Therefore

$$\Omega' = (\bigcup_{i=1}^n \Omega_i) \cup \{\mathsf{qVal}(W)\} \cup \{\ulcorner W \trianglelefteqslant W' \urcorner \mid W' \in \bigcup_{i=1}^n \mathcal{W}_i\} \cup \{\ulcorner d \trianglelefteqslant W \urcorner\}\ .$$

We define a new substitution

$$\rho = \biguplus_{i=1}^n \rho_i \uplus \rho'_0 \uplus \biguplus_{j=1}^m \rho'_j \uplus \{W \mapsto d\}\ .$$

It is straightforward to check that ρ is a solution for Ω' because ρ is solution of:

- Each Ω_i $(1 \leq i \leq n)$, because ρ_i is solution of Ω'_i which contains Ω_i (see inductive hypothesis 1) and ρ is an extension of ρ_i.
- $\{\mathsf{qVal}(W)\}$ because $\mathsf{qVal}(W)\rho = \mathsf{qVal}(d)$ which holds by definition.
- $\{\ulcorner W \trianglelefteqslant W' \urcorner \mid W' \in \bigcup_{i=1}^n \mathcal{W}_i\}$ because $W\rho = d$, ρ is solution of $\{\ulcorner d_i \trianglelefteqslant W' \urcorner \mid W' \in \mathcal{W}_i\}$ for each $i = 1 \ldots n$ (see inductive hypothesis 1), and $d \trianglelefteqslant d_i$ $(1 \leq i \leq n)$ by the hypotheses of the inference rule $\mathbf{QDP}_{\mathcal{P}}$.
- $\{\ulcorner d \trianglelefteqslant W \urcorner\}$ since $W\rho = d$ and trivially $d \trianglelefteqslant d$.

The transformed of the program rule $R_l = (f(\bar{t}_n) \xrightarrow{\alpha} r \Leftarrow \overline{\delta}_m) \in \mathcal{P}$ will be a program rule in $\mathcal{P}^{\mathcal{T}}$ of the form:

$$\begin{aligned}(R_l)^{\mathcal{T}} = (f(\bar{t}_n, W) \to r' \Leftarrow\ & \mathsf{qVal}(W), \Omega_r, (\ulcorner W \trianglelefteqslant \alpha \circ W' \urcorner)_{W' \in \mathcal{W}'_0},\\ & \Omega_{\delta_1}, (\ulcorner W \trianglelefteqslant \alpha \circ W'_1 \urcorner)_{W'_1 \in \mathcal{W}'_1}, \delta'_1\\ & \vdots\\ & \Omega_{\delta_m}, (\ulcorner W \trianglelefteqslant \alpha \circ W'_m \urcorner)_{W'_m \in \mathcal{W}'_m}, \delta'_m\end{aligned}$$

with $r^{\mathcal{T}} = (r', \Omega_r, \mathcal{W}'_0)$ and $(\ \delta_j{}^{\mathcal{T}} = (\delta'_j, \Omega_{\delta_j}, \mathcal{W}'_j)\ )_{j=1\ldots m}$.
Then we prove $(f(\overline{e'}_n, W) \to t)\rho \Leftarrow \Pi$ in CFLP($\mathcal{C}$) with a $\mathbf{DF}_{\mathcal{P}}$ root inference step using the program rule $(R_l)^{\mathcal{T}}$ and the substitution $\theta' = \theta \uplus \rho$ to instantiate the program rule. We next check that every premise of this inference can be proven in CRWL($\mathcal{C}$):

- $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} (e'_i\rho \to t_i(\theta \uplus \rho)) \Leftarrow \Pi$ for $i = 1 \ldots n$. We observe that the only variables of e'_i that can be affected by ρ are those in ρ_i. Moreover, ρ cannot affect t_i because the program transformation does not introduce new variables in terms. Therefore $(e'_i\rho \to t_i(\theta \uplus \rho)) = (e'_i \to t_i\theta)\rho_i$ and $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} (e'_i \to t_i\theta)\rho_i \Leftarrow \Pi$ for $i = 1 \ldots n$ follows from inductive hypothesis number 1.
- $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} (W\rho \to W(\theta \uplus \rho)) \Leftarrow \Pi$. By construction of ρ, $(W\rho \to W(\theta \uplus \rho)) = d \to d$ and one $\mathbf{RR}$ inference step proves this statement.
- $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} (r'(\theta \uplus \rho) \to t\rho) \Leftarrow \Pi$. In this case $t\rho = t$ because t it contains no variables introduced during the transformation, and $r'(\theta \uplus \rho) = r'(\theta\rho'_0)$ since ρ'_0 is the only part of ρ that can affect r' and the range of θ does not include any of the new variables in the domain of ρ'_0. Now, $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} (r'\theta \to t)\rho'_0 \Leftarrow \Pi$ follows from inductive hypothesis number 2.
- $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} \mathsf{qVal}(W)(\theta \uplus \rho) \Leftarrow \Pi$. W is a fresh variable and, by construction of ρ, $\mathsf{qVal}(W)(\theta \uplus \rho) = \mathsf{qVal}(d)$. $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} \mathsf{qVal}(d) \Leftarrow \Pi$ trivially holds.
- $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} \Omega_r(\theta \uplus \rho) \Leftarrow \Pi$. $\Omega_r(\theta \uplus \rho) = \Omega_r\rho = \Omega_r\rho'_0$ and, by construction, ρ'_0 is solution of Ω_r.
- $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} (\ulcorner W \trianglelefteq \alpha \circ W'\urcorner)(\theta \uplus \rho) \Leftarrow \Pi$ for each $W' \in \mathcal{W}'_0$. We have $(\ulcorner W \trianglelefteq \alpha \circ W'\urcorner)(\theta \uplus \rho) = (\ulcorner W \trianglelefteq \alpha \circ W'\urcorner)\rho = \ulcorner W\rho \trianglelefteq \alpha \circ W'\rho'_0\urcorner = \ulcorner d \trianglelefteq \alpha \circ W'\rho'_0\urcorner$. And $\ulcorner d \trianglelefteq \alpha \circ W'\rho'_0\urcorner$ holds because $d \trianglelefteq \alpha \circ d'_0$ by the hypotheses of the inference rule $\mathbf{QDP}_{\mathcal{P}}$, and $\ulcorner d'_0 \trianglelefteq W'\urcorner$ by inductive hypothesis number 2.
- $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} \Omega_{\delta_j}(\theta \uplus \rho) \Leftarrow \Pi$ for $j = 1 \ldots m$. As in the previous premises $\Omega_{\delta_j}(\theta \uplus \rho) = \Omega_{\delta_j}\rho = \Omega_{\delta_j}\rho'_j$ and ρ'_j is solution of Ω_{δ_j} as a consequence of the inductive hypothesis number 3.
- $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} (\ulcorner W \trianglelefteq \alpha \circ W'_j\urcorner)(\theta \uplus \rho) \Leftarrow \Pi$ for every $W'_j \in \mathcal{W}'_j$ and $j = 1 \ldots m$. We have $(\ulcorner W \trianglelefteq \alpha \circ W'_j\urcorner)(\theta \uplus \rho) = (\ulcorner W \trianglelefteq \alpha \circ W'_j\urcorner)\rho = \ulcorner W\rho \trianglelefteq \alpha \circ W'_j\rho\urcorner = \ulcorner d \trianglelefteq \alpha \circ W'_j\rho'_j\urcorner$. Now, from the hypotheses of the inference rule $\mathbf{QDP}_{\mathcal{P}}$ follows $d \trianglelefteq \alpha \circ d'_j$ for $j = 1 \ldots m$, and from inductive hypothesis number 3, ρ'_j is solution of $\ulcorner d'_j \trianglelefteq W'_j\urcorner$. Hence $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} \ulcorner d \trianglelefteq \alpha \circ W'_j\rho'_j\urcorner \Leftarrow \Pi$ for $j = 1 \ldots k$.
- $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} \delta'_j(\theta \uplus \rho) \Leftarrow \Pi$ for $j = 1 \ldots m$. In this case δ'_j can contain variables from both θ and ρ'_j. Hence $\delta'_j(\theta \uplus \rho) = (\delta'_j\theta)\rho'_j$. And $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} (\delta'_j\theta)\rho'_j \Leftarrow \Pi$ follows from the inductive hypothesis number 3.

– **QPF**. In this case $\psi = p(\overline{e}_n) \to v$ and the inference step applied at the root is of the form

$$\frac{(\ (e_i \to t_i)\sharp d_i \Leftarrow \Pi\ )_{i=1\ldots n}}{(p(\overline{e}_n) \to v)\sharp d \Leftarrow \Pi}$$

with $v \in \mathcal{V}ar \cup DC^0 \cup B_{\mathcal{C}}$, $\Pi \models_{\mathcal{C}} p(\overline{t}_n) \to v$ and $d \trianglelefteq d_i$ $(1 \leq i \leq n)$. In order to obtain $(\psi\sharp d \Leftarrow \Pi)^{\mathcal{T}}$ one has to:

- First, apply the transformation rule $\mathbf{TCE}_1$,

$$p(\overline{e}_n)^{\mathcal{T}} = (p(\overline{e'}_n), \bigcup_{i=1}^{n} \Omega_i, \bigcup_{i=1}^{n} \mathcal{W}_i)$$

  where ${e_i}^{\mathcal{T}} = (e'_i, \Omega_i, \mathcal{W}_i)$ for $i = 1 \ldots n$.
- Second, apply the transformation rule **TP**,

$$(p(\overline{e}_n) \to v)^{\mathcal{T}} = (p(\overline{e'}_n) \to v, \bigcup_{i=1}^{n} \Omega_i, \bigcup_{i=1}^{n} \mathcal{W}_i) \ .$$

- And finally, apply the transformation rule **TCS**,

$$(\psi \sharp d \Leftarrow \Pi)^{\mathcal{T}} = (p(\overline{e'}_n) \to v \Leftarrow \Pi, \bigcup_{i=1}^{n} \Omega_i \cup \{\ulcorner d \trianglelefteqslant W \urcorner \mid W \in \bigcup_{i=1}^{n} \mathcal{W}_i\}) \ .$$

Therefore

$$\Omega' = \bigcup_{i=1}^{n} \Omega_i \cup \{\ulcorner d \trianglelefteqslant W \urcorner \mid W \in \bigcup_{i=1}^{n} \mathcal{W}_i\} \ .$$

From the premises $(\ (e_i \to t_i)\sharp d_i \Leftarrow \Pi\ )_{i=1\ldots n}$ of the inference rule **QPF**, and by the inductive hypothesis we have $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} (e'_i \to t_i)\rho_i \Leftarrow \Pi$ $(1 \leq i \leq n)$ for some substitutions $\rho_i : \mathrm{var}(\Omega'_i) \to D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ solution of

$$\Omega'_i = \Omega_i \cup \{\ulcorner d_i \trianglelefteqslant W \urcorner \mid W \in \mathcal{W}_i\}$$

for $i = 1 \ldots n$. We define a new substitution $\rho : \mathrm{var}(\Omega') \to D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ as $\rho = \biguplus_{i=1}^{n} \rho_i$. It is easy to check that ρ is solution of Ω':

- It is solution of every Ω'_i for $i = 1 \ldots n$, since $\rho|\mathrm{var}(\Omega'_i) = \rho_i$. Therefore it is solution of $\bigcup_{i=1}^{n} \Omega_i$.
- It is a solution of $\{\ulcorner d \trianglelefteqslant W \urcorner \mid W \in \bigcup_{i=1}^{n} \mathcal{W}_i\}$ because as solution of Ω'_i for $i = 1 \ldots n$, ρ is solution of $\{\ulcorner d_i \trianglelefteqslant W \urcorner \mid W \in \mathcal{W}_i\}$, and by the hypothesis of the inference rule **QPF**, $d \trianglelefteqslant d_i$ $(1 \leq i \leq n)$.

We now prove $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} (p(\overline{e'}_n) \to v)\rho \Leftarrow \Pi$ with a proof tree T' with a **PF** root inference of the form:

$$\frac{(\ (e'_i \to t_i)\rho \Leftarrow \Pi\ )_{i=1\ldots n}}{(p(\overline{e'}_n)\rho \to v) \Leftarrow \Pi}$$

The rule can be applied because the requirements $v \in \mathcal{V}ar \cup DC^0 \cup B_{\mathcal{C}}$ and $\Pi \models_{\mathcal{C}} p(\overline{t}_n) \to v$ are ensured by the hypothesis of the inference rule **QPF**. In order to justify that $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} (e'_i \to t_i)\rho \Leftarrow \Pi$ for each $i = 1 \ldots n$, we observe that the only variables of $(e'_i \to t_i)$ that can be affected by ρ are those introduced in e'_i by the transformation, and that therefore $(e'_i \to t_i)\rho = (e'_i \to t_i)\rho_i$ for $i = 1 \ldots n$, and it is easy to check that these premises correspond to the inductive hypotheses of this case.

– **QAC**. This case is analogous to the previous proof, with the only differences being:

- The inference rule applied at the root of the proof tree is a **QAC** inference rule instead of a **QPF** inference rule.
- In order to obtain the $(\psi\sharp d \Leftarrow \Pi)^{\mathcal{T}}$, the transformation rules applied are **TA** and **TCS** instead of $\mathbf{TCE}_1$, **TP** and **TCS**.
- The proof tree T' will have an **AC** inference step at its root instead of a **PF** inference step.

[*2.* $\Rightarrow$ *1.*] *(Transformation soundness).* Assume $\rho \in \mathrm{Sol}_{\mathcal{C}}(\Omega')$ such that $\mathrm{vdom}(\rho) = \mathrm{var}(\Omega')$ and $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} \psi'\rho \Leftarrow \Pi$ by means of a CRWL($\mathcal{C}$) proof tree T with k nodes. Reasoning by induction on k we show the existence of a QCRWL($\mathcal{D},\mathcal{C}$) proof tree T' witnessing $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} \psi\sharp d \Leftarrow \Pi$.

Basis (k=1). If T contains only one node the QCRWL($\mathcal{D},\mathcal{C}$) inference step applied at the root must be any of the following:

- **TI**. In this case $\psi'\rho \Leftarrow \Pi$ is a trivial c-statement. Then $\psi'\rho$ is either of the form $e' \to \bot$ or $\mathrm{Unsat}_{\mathcal{C}}(\Pi)$. In the first case, since the transformation introduces no new variables at the right-hand side of a production, ψ' is of the form $e'' \to \bot$ with $e' = e''\rho$, and ψ is of the form $e \to \bot$, hence $\psi\sharp d \Leftarrow \Pi$ is trivial. Analogously, if $\mathrm{Unsat}_{\mathcal{C}}(\Pi)$ then $\psi\sharp d \Leftarrow \Pi$ is trivial as well. Therefore T' consists of a single node $\psi\sharp d \Leftarrow \Pi$ with d any value in $D_{\mathcal{D}} \setminus \{\mathbf{b}\}$, with a **QTI** inference step at its root.
- **RR**. In this case $\psi'\rho = v \to v$ with $v \in \mathcal{V}ar \cup B_{\mathcal{C}}$. Then $\psi' = v_1 \to v_2$ for some $v_1, v_2 \in \mathcal{V}ar \cup B_{\mathcal{C}}$ such that $\psi'\rho = v \to v$. Since ψ' cannot contain new variables introduced by the transformation (by the transformation rules), this means $\psi'\rho = \psi'$, and then $\psi' = v \to v$. Therefore $\psi = v \to v$, and T' consists of a single node containing $(v \to v)\sharp d \Leftarrow \Pi$ for any $d \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ as the conclusion of a **QRR** inference step.
- **DC**. Then $\psi'\rho = c \to c$, which means that ψ' can be either of the form $c \to c$, $X \to c$, or $X \to Y$ with X, Y variables. In every case ψ' does not include new variables introduced by the transformation, and therefore $\psi'\rho = \psi'$, which means that $\psi' = c \to c$ is the only possibility. Therefore $\psi = c \to c$, and T' consists of a single node containing $(c \to c)\sharp d \Leftarrow \Pi$ for some $d \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ as the conclusion of a **QDC** inference step.

Inductive step (k>1). The CRWL($\mathcal{C}$) inference step applied at the root must be any of the following:

- **DC**. Then $\psi'\rho = c(\overline{e''}_n) \to c(\overline{t}_n)$ where $c \in DC^n$ and $n > 0$, which implies that $\psi = c(\overline{e}_n) \to c(\overline{t}_n)$ for values e_i verifying ${e_i}^{\mathcal{T}} = (e'_i, \Omega_i, \mathcal{W}_i)$ for $i = 1 \ldots n$, and $e''_i = e'_i\rho$ for $i = 1 \ldots n$. Then

$$\psi^{\mathcal{T}} = (c(\overline{e}_n) \to c(\overline{t}_n))^{\mathcal{T}} = (c(\overline{e'}_n) \to c(\overline{t}_n), \bigcup_{i=1}^{n} \Omega_i, \bigcup_{i=1}^{n} \mathcal{W}_i)$$

  and thus $\varphi = (c(\overline{e}_n) \to c(\overline{t}_n))\sharp d \Leftarrow \Pi$ for some $d \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ such that $\varphi^{\mathcal{T}} = (\psi' \Leftarrow \Pi, \Omega')$, with

$$\Omega' = \bigcup_{i=1}^{n} \Omega_i \cup \{\ulcorner d \lhd W \urcorner \mid W \in \bigcup_{i=1}^{n} \mathcal{W}_i\}$$

The substitution $\rho : \mathrm{var}(\Omega') \to D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ must be solution of Ω', and the inference step at the root must be of the form:

$$\frac{(\ e'_i\rho \to t_i \Leftarrow \Pi\ )_{i=1\ldots n}}{c(\overline{e'}_n)\rho \to c(\overline{t}_n) \Leftarrow \Pi}$$

In the premises we have the proofs T_i of $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} e'_i\rho \Leftarrow \Pi$ for $i = 1 \ldots n$. Now, for each $1 \leq i \leq n$ we obtain a new value $d_i \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ as $d_i = \bigsqcap\{W\rho \mid W \in \mathcal{W}_i\}$. Then we will prove $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} \varphi$ applying the following **QDC** inference step at the root:

$$\frac{(\ (e_i \to t_i)\sharp d_i \Leftarrow \Pi\ )_{i=1\ldots n}}{(c(\overline{e}_n) \to c(\overline{t}_n))\sharp d \Leftarrow \Pi}$$

In order to ensure that this step must be applied we must check that $d \trianglelefteq d_i$ $(1 \leq i \leq n)$. This holds because ρ is solution of Ω', in particular of $\{\ulcorner d \trianglelefteq W \urcorner \mid W \in \mathcal{W}_i\}$ for $i = 1 \ldots n$. Therefore for each $i = 1 \ldots n$ and $W \in \mathcal{W}_i$, $d \trianglelefteq W\rho$, which means that $d \trianglelefteq d_i = \bigsqcap\{W\rho \mid W \in \mathcal{W}_i\}$. To complete the proof we must check that there are proof trees for the premises, i.e. that $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} \varphi_i$ with $\varphi_i = (e_i \to t_i)\sharp d_i \Leftarrow \Pi$, $i = 1 \ldots n$. This is a consequence of the inductive hypotheses since for each $i = 1 \ldots n$:

- ${\varphi_i}^{\mathcal{T}} = (e'_i \to t_i \Leftarrow \Pi, \Omega'_i)$, with $\Omega'_i = \Omega_i \cup \{\ulcorner d_i \trianglelefteq W \urcorner \mid W \in \mathcal{W}_i\}$.
- ρ is solution of Ω'_i, since it is solution of Ω_i and by the definition of d_i, $d_i \trianglelefteq W\rho$ for every $W \in \mathcal{W}_i$.
- We have that $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} e'_i\rho \Leftarrow \Pi$ for $i = 1 \ldots n$ (the premises of the **DC** step).

– $\mathbf{DF}_{\mathcal{P}}$. The inference step at the root of T will use an instance $({R_l}^{\mathcal{T}})\theta \in [\mathcal{P}^{\mathcal{T}}]_{\perp}$ of a program rule ${R_l}^{\mathcal{T}}$ of $\mathcal{P}^{\mathcal{T}}$. ${R_l}^{\mathcal{T}}$ will be the transformed of a program rule $R_l = (f(\overline{t}_n) \xrightarrow{\alpha} r \Leftarrow \overline{\delta}_m) \in \mathcal{P}$, and therefore will have the form:

$$\begin{aligned} {R_l}^{\mathcal{T}} = (f(\overline{t}_n, W) \to r' \Leftarrow\ & \mathsf{qVal}(W), \Omega_r, (\ulcorner W \trianglelefteq \alpha \circ W' \urcorner)_{W' \in \mathcal{W}'_0}, \\ & \Omega_{\delta_1}, (\ulcorner W \trianglelefteq \alpha \circ W'_1 \urcorner)_{W'_1 \in \mathcal{W}'_1}, \delta'_1 \\ & \vdots \\ & \Omega_{\delta_m}, (\ulcorner W \trianglelefteq \alpha \circ W'_m \urcorner)_{W'_m \in \mathcal{W}'_m}, \delta'_m \end{aligned}$$

with ${r}^{\mathcal{T}} = (r', \Omega_r, \mathcal{W}'_0)$ and $({\delta_j}^{\mathcal{T}} = (\delta'_j, \Omega'_j, \mathcal{W}'_j))_{j=1\ldots m}$.
In this case, $\psi'\rho$ must be of the form $(f(\overline{e'}_{n+1}) \to t)\rho$. By the theorem premises, there exists a qc-statement $\psi\sharp d \Leftarrow \Pi$ such that $(\psi\sharp d \Leftarrow \Pi)^{\mathcal{T}} = (\psi' \Leftarrow \Pi, \Omega')$ for some Ω'. Examining the transformation program rules we observe that the only possibility for ψ is to be of the form $f(\overline{e}_n) \to t$ and that the **TCS** transformation rules should have been applied followed by **TP** and $\mathbf{TCE}_2$. This means in particular that $d \neq \mathbf{b}$ and that ${e_i}^{\mathcal{T}} = (e'_i, \Omega_i, \mathcal{W}_i)$ for $i = 1 \ldots n$ and that $e'_{n+1} = V$ with V fresh variable. Hence

$$\begin{aligned} {\psi}^{\mathcal{T}} = (f(\overline{e'}_n, V) \to t, (\ & \textstyle\bigcup_{i=1}^{n} \Omega_i) \cup \{\mathsf{qVal}(V)\} \cup \\ & \{\ulcorner V \trianglelefteq W' \urcorner \mid W' \in \textstyle\bigcup_{i=1}^{n} \mathcal{W}_i\},\ \{V\}) \end{aligned}$$

and $\varphi = (f(\overline{e}_n) \to t)\sharp d \Leftarrow \Pi$ for some $d \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$. By hypotheses, ρ is solution of

$$\Omega' = (\bigcup_{i=1}^{n} \Omega_i) \cup \{\mathsf{qVal}(V)\} \cup \{\ulcorner V \trianglelefteqslant W' \urcorner \mid W' \in \bigcup_{i=1}^{n} \mathcal{W}_i\} \cup \{\ulcorner d \trianglelefteqslant V \urcorner\}$$

which means, in particular, that $V\rho \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$, since it must hold both $\mathsf{qVal}(V)$ and $\ulcorner d \trianglelefteqslant V \urcorner$.
Therefore the root of T will be $f(\overline{e'}_n, V)\rho \to t \Leftarrow \Pi$, with premises proof trees proving:

1. $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} (\ e'_i\rho \to t_i\theta \Leftarrow \Pi\ )_{i=1\ldots n}$.
2. $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} (\ V\rho \to W\theta \Leftarrow \Pi\ )$. Since $V\rho \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ then either $W\theta = V\rho$ or $W\theta = \mathbf{b}$. By premise 4 below, $W\theta \neq \mathbf{b}$, therefore $W\theta = V\rho$.
3. $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} r'\theta \to t \Leftarrow \Pi$.
4. $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} \mathsf{qVal}(W\theta) \Leftarrow \Pi$.
5. $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} \Omega_r\theta \Leftarrow \Pi$.
6. $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} (\ulcorner W \trianglelefteqslant \alpha \circ W' \urcorner)_{W' \in \mathcal{W}'_0}\theta \Leftarrow \Pi$.
7. $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} \Omega_{\delta_j}\theta \Leftarrow \Pi$ for $j = 1 \ldots m$.
8. $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} (\ulcorner W \trianglelefteqslant \alpha \circ W'_j \urcorner)_{W'_j \in \mathcal{W}'_j}\theta \Leftarrow \Pi$ for $j = 1 \ldots m$.
9. $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} \delta'_j\theta \Leftarrow \Pi$ for $j = 1 \ldots m$.

Then we can prove $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} \varphi$ by applying a $\mathbf{QDF}_{\mathcal{P}}$ inference step of the form:

$$\frac{(\ (e_i \to t_i\theta)\sharp d_i \Leftarrow \Pi\ )_{i=1\ldots n} \quad (r\theta \to t)\sharp d'_0 \Leftarrow \Pi \quad (\ \delta_j\theta\sharp d'_j \Leftarrow \Pi\ )_{j=1\ldots m}}{(f(\overline{e}_n) \to t)\sharp d \Leftarrow \Pi}$$

where

- $d_i = \bigsqcap\{W\rho \mid W \in \mathcal{W}_i\}$ for $i = 1 \ldots n$.
- $d'_0 = \bigsqcap\{W\theta \mid W \in \mathcal{W}'_0\}$.
- $d'_j = \bigsqcap\{W\theta \mid W \in \mathcal{W}'_j\}$ for $j = 1 \ldots m$.

For proving $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} \varphi$ we need to check that

- $d \trianglelefteqslant d_i$ $(1 \leq i \leq n)$. Since ρ is solution of Ω', $d \trianglelefteqslant W\rho$, and $W\rho \trianglelefteqslant W'\rho$ for every $W' \in \mathcal{W}_i$ and every $1 \leq i \leq n$. Therefore $d \trianglelefteqslant \bigsqcap\{\rho(W) \mid W \in \mathcal{W}_i\} = d_i$ for $i = 1 \ldots n$.
- $d \trianglelefteqslant \alpha \circ d'_0$. Since ρ is solution of Ω', $d \trianglelefteqslant V\rho = W\theta$. From premise 6, $W\theta \trianglelefteqslant \alpha \circ W'\theta$ for every $W' \in \mathcal{W}'_0$. Therefore $d \trianglelefteqslant \bigsqcap\{W\theta \mid W \in \mathcal{W}'_0\} = d'_0$.
- $d \trianglelefteqslant \alpha \circ d'_j$ $(1 \leq j \leq m)$. Analogous to the previous point but using premise 8.

Finally, in order to justify the premises of the $\mathbf{QDF}_{\mathcal{P}}$ we must prove:

- $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} (e_i \to t_i\theta)\sharp d_i \Leftarrow \Pi$, which is a consequence of applying the inductive hypotheses to the premises 1, $(\ e'_i\rho \to t_i\theta \Leftarrow \Pi\ )_{i=1\ldots n}$, following the same reasoning we applied for the premises of the **DC** inference.
- $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} (r\theta \to t)\sharp d'_0 \Leftarrow \Pi$. Analogously, is a consequence of the inductive hypothesis and of premise 3.

- $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} (\ \delta_j\theta \sharp d'_j \Leftarrow \Pi\ )_{j=1\ldots m}$. Again a consequence of the inductive hypothesis, this time applied to the premise 9.

– **PF**. Analogous to the proof for the **DC** inference step.
– **AC**. analogous to the proof for the **DC** inference step. □

Using Theorem 3 we can prove that the transformation of goals specified in Fig. 5 preserves solutions in the sense of the following result.

Theorem 4. *Let G be a goal for a given QCFLP($\mathcal{D},\mathcal{C}$)-program $\mathcal{P}$. Then, the two following statements are equivalent:*

1. $\langle \sigma, \mu, \Pi \rangle \in Sol_{\mathcal{P}}(G)$.
2. $\langle \sigma \uplus \mu \uplus \rho, \Pi \rangle \in Sol_{\mathcal{P}^{\mathcal{T}}}(G^{\mathcal{T}})$ *for some* $\rho \in Val_{\mathcal{D}}$ *such that* $vdom(\rho)$ *is the set of new variables W introduced by the transformation of G.*

Proof. Let $G = (\ \delta_i \sharp W_i, W_i \trianglerighteq \beta_i\ )_{i=1\ldots m}$, σ and μ be given. For $i = 1 \ldots m$, consider ${\delta_i}^{\mathcal{T}} = (\delta'_i, \Omega_i, \mathcal{W}_i)$ and $\Omega'_i = \Omega_i \cup \{\ulcorner W_i \trianglelefteq W \urcorner \mid W \in \mathcal{W}_i\}$. According to Fig. 5, $G^{\mathcal{T}} = (\Omega'_i,\ \mathsf{qVal}(W_i), \ulcorner W_i \trianglerighteq \beta_i \urcorner,\ \delta'_i)_{i=1\ldots m}$. Then, because of Def. 9(2) and the analogous notion of solution for CFLP($\mathcal{C}$) goals explained in Sect. 3, the two statements of the theorem can be reformulated as follows:

(*a*) $W_i\mu \trianglerighteq \beta_i$ and $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} \delta_i\sigma \sharp W_i\mu \Leftarrow \Pi$ hold for $i = 1 \ldots m$.
(*b*) There exists $\rho \in \mathrm{Val}_{\mathcal{D}}$ with $\mathrm{vdom}(\rho) = \bigcup_{i=1}^{m} \mathrm{var}(\Omega_i)$ such that $\rho \in \mathrm{Sol}_{\mathcal{C}}(\Omega'_i\mu)$, $W_i\mu \trianglerighteq \beta_i$ and $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} (\delta'_i\sigma)\rho \Leftarrow \Pi$ hold for $i = 1 \ldots m$.

[$(a) \Rightarrow (b)$] Assume (a). Note that $\delta_i\sigma \sharp W_i\mu \Leftarrow {\Pi}^{\mathcal{T}}$ is $\delta'_i\sigma \Leftarrow \Pi, \Omega'_i\mu$. Applying Theorem 3 (with $\psi = \delta_i\sigma$, $d = W_i\mu$ and Π) we obtain $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} (\delta'_i\sigma)\rho_i \Leftarrow \Pi$ for some $\rho_i \in \mathrm{Sol}_{\mathcal{C}}(\Omega'_i\mu)$ with $\mathrm{vdom}(\rho_i) = \mathrm{var}(\Omega'_i\mu) = \mathrm{var}(\Omega_i)$. Then (b) holds for $\rho = \biguplus_{i=1}^{m} \rho_i$.
[$(b) \Rightarrow (a)$] Assume (b). Let $\rho_i = \rho{\restriction}\mathrm{var}(\Omega_i)$, $i = 1 \ldots m$. Note that (b) ensures $\mathcal{P}^{\mathcal{T}} \vdash_{\mathcal{C}} (\delta'_i\sigma)\rho_i \Leftarrow \Pi$ and $\rho \in \mathrm{Sol}_{\mathcal{C}}(\Omega'_i\mu)$. Then Theorem 3 can be applied (again with $\psi = \delta_i\sigma$, $d = W_i\mu$ and Π) to obtain $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} \delta_i\sigma \sharp W_i\mu \Leftarrow \Pi$. Therefore, (a) holds. □

As an example of goal solving via the transformation, we consider again the *library program* $\mathcal{P}$ and the goal G discussed in the Introduction. Both belong to the instance QCFLP($\mathcal{U},\mathcal{R}$) of our scheme. Their translation into CFLP($\mathcal{R}$) can be executed in the $\mathcal{TOY}$ system [3] after loading the Real Domain Constraints library (`cflpr`). The source and translated code are publicly available at `gpd.sip.ucm.es/cromdia/qlp`. Solving the transformed goal in $\mathcal{TOY}$ computes the answer announced in the Introduction as follows:

```
Toy(R)> qVal([W]), W>=0.65, search("German","Essay",intermediate,W) == R
      { R -> 4 }
      { W=<0.7, W>=0.65 }
sol.1, more solutions (y/n/d/a) [y]? no
```

The best qualification value for `W` provided by the answer constraints is `0.7`.

5 Conclusions

The work in this report is based on the scheme CFLP($\mathcal{C}$) for functional logic programming with constraints presented in [13]. Our main results are: a new programming scheme QCFLP($\mathcal{D},\mathcal{C}$) extending the first-order fragment of CFLP($\mathcal{C}$) with qualified computation capabilities; a rewriting logic QCRWL($\mathcal{D},\mathcal{C}$) characterizing QCFLP($\mathcal{D},\mathcal{C}$)-program semantics; and a transformation of QCFLP($\mathcal{D},\mathcal{C}$) into CFLP($\mathcal{C}$) preserving program semantics and goal solutions, that can be used as a correct implementation technique. Existing CFLP($\mathcal{C}$) systems such as $\mathcal{TOY}$ [3] and Curry [9] that use definitional trees as an efficient implementation tool can easily adopt the implementation, since the structure of definitional trees is quite obviously preserved by the transformation.

As argued in the Introduction, our scheme is more expressive than the main related approaches we are aware of. By means of an example dealing with a simplified library, we have shown that instances of QCFLP($\mathcal{D},\mathcal{C}$) can serve as a declarative language for flexible information retrieval problems, where qualified (rather than exact) answers to user's queries can be helpful.

As future work we plan to extend QCFLP($\mathcal{D},\mathcal{C}$) and the program transformation in order to provide explicit support for similarity-based reasoning, as well as the higher-order programming features available in CFLP($\mathcal{C}$). We also plan to automate the program transformation, which should be embedded as part of an enhanced version of the $\mathcal{TOY}$ system. Finally, we plan further research on flexible information retrieval applications, using different instances of our scheme.

References

1. S. Antoy, R. Echahed, and M. Hanus. A needed narrowing strategy. *Journal of the ACM*, 47(4):776–822, 2000.
2. K. 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.
3. 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 2.3.1, 2007. R. Caballero and J. Sánchez (Eds.), Available at `http://toy.sourceforge.net`.
4. R. Caballero, M. Rodríguez-Artalejo, and C. A. Romero-Díaz. Similarity-based reasoning in qualified logic programming. In *PPDP '08: Proceedings of the 10th international ACM SIGPLAN conference on Principles and Practice of Declarative Programming*, pages 185–194, New York, NY, USA, 2008. ACM.
5. R. del Vado-Vírseda. Declarative constraint programming with definitional trees. In B. 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.
6. M. Gabbrielli, G. M. Dore, and G. Levi. Observable semantics for constraint logic programs. *Journal of Logic and Computation*, 5(2):133–171, 1995.

7. M. Gabbrielli and G. Levi. Modeling answer constraints in constraint logic programs. In *Proceedings of the 8th International Conference on Logic Programming (ICLP'91)*, pages 238–252. The MIT Press, 1991.
8. 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.
9. 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`.
10. 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.
11. J. W. Lloyd. *Foundations of Logic Programming, Second Edition.* Springer, 1987.
12. F. J. López-Fraguas, M. Rodríguez-Artalejo, and R. del Vado-Virseda. A lazy narrowing calculus for declarative constraint programming. In *Proceedings of the 6th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP'04)*, pages 43–54. ACM Press, 2004.
13. 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.
14. G. Moreno and V. Pascual. Formal properties of needed narrowing with similarity relations. *Electronic Notes in Theoretical Computer Science*, 188:21–35, 2007.
15. 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.
16. S. Riezler. *Probabilistic Constraint Logic Programming.* PhD thesis, Neuphilologischen Fakultät del Universität Tübingen, 1998.
17. M. Rodríguez-Artalejo. Functional and constraint logic programming. In C. M. 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.
18. M. Rodríguez-Artalejo and C. A. Romero-Díaz. A generic scheme for qualified logic programming. Technical Report SIC-1-08, Universidad Complutense, Departamento de Sistemas Informáticos y Computación, Madrid, Spain, 2008.
19. M. Rodríguez-Artalejo and C. A. Romero-Díaz. Quantitative logic programming revisited. In J. Garrigue and M. Hermenegildo, editors, *Functional and Logic Programming (FLOPS'08)*, volume 4989 of *LNCS*, pages 272–288. Springer Verlag, 2008.
20. M. I. Sessa. Approximate reasoning by similarity-based SLD resolution. *Theoretical Computer Science*, 275(1-2):389–426, 2002.
21. V. S. Subrahmanian. Uncertainty in logic programming: Some recollections. *Association for Logic Programming Newsletter*, 20(2), 2007.
22. A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5(2):285–309, 1955.

Fixpoint & Proof-theoretic Semantics for CLP with Qualification and Proximity ∗

Technical Report SIC-1-10

MARIO RODRÍGUEZ-ARTALEJO and CARLOS A. ROMERO-DÍAZ

Departamento de Sistemas Informáticos y Computación, Universidad Complutense
Facultad de Informática, 28040 Madrid, Spain
(*e-mail:* mario@sip.ucm.es, cromdia@fdi.ucm.es)

Abstract

Uncertainty in Logic Programming has been investigated during the last decades, dealing with various extensions of the classical LP paradigm and different applications. Existing proposals rely on different approaches, such as clause annotations based on uncertain truth values, qualification values as a generalization of uncertain truth values, and unification based on proximity relations. On the other hand, the CLP scheme has established itself as a powerful extension of LP that supports efficient computation over specialized domains while keeping a clean declarative semantics. In this report we propose a new scheme SQ-CLP designed as an extension of CLP that supports qualification values and proximity relations. We show that several previous proposals can be viewed as particular cases of the new scheme, obtained by partial instantiation. We present a declarative semantics for SQCLP that is based on observables, providing fixpoint and proof-theoretical characterizations of least program models as well as an implementation-independent notion of goal solutions.

KEYWORDS: Constraint Logic Programming, Qualification Domains and Values, Proximity Relations.

1 Introduction

Many extensions of logic programming (shortly LP) to deal with uncertainty have been proposed in the last decades. A line of research not related to this report is based on probabilistic extensions of LP such as (Ng and Subrahmanian 1992). Other proposals in the field replace classical two-valued logic by some kind of many-valued logic whose truth values can be attached to computed answers and are usually interpreted as certainty degrees. The next paragraphs summarize some relevant approaches of this kind.

There are extensions of LP using annotations in program clauses to compute a certainty degree for the head atom from the certainty degrees previously computed

∗ This work has been partially supported by the Spanish projects STAMP (TIN2008-06622-C03-01), PROMETIDOS–CM (S2009TIC-1465) and GPD–UCM (UCM–BSCH–GR58/08-910502).

for the body atoms. This line of research includes the seminal proposal of Quantitative Logic Programming by (van Emden 1986) and inspired later works such as the Generalized Annotated logic Programs (shortly GAP) by (Kifer and Subrahmanian 1992) and the QLP scheme for Qualified LP (Rodríguez-Artalejo and Romero-Díaz 2008b). While (van Emden 1986) and other early approaches used real numbers of the interval $[0, 1]$ as certainty degrees, QLP and GAP take elements from a parametrically given lattice to be used in annotations and attached to computed answers. In the case of QLP, the lattice is called a *qualification domain* and its elements (called *qualification values*) are not always understood as certainty degrees. As argued in (Rodríguez-Artalejo and Romero-Díaz 2008b), GAP is a more general framework, but QLP's semantics have some advantages for its intended scope.

There are also extended LP languages based on fuzzy logic (Zadeh 1965; Hájek 1998), which can be classified into two major lines. The first line includes Fuzzy LP languages such as (Vojtáš 2001; Vaucheret et al. 2002; Guadarrama et al. 2004) and the Multi-Adjoint LP (shortly MALP) framework by (Medina et al. 2001a; Medina et al. 2001b). All these approaches extend classical LP by using clause annotations and a fuzzy interpretation of the connectives and aggregation operators occurring in program clauses and goals. There is a relationship between Fuzzy LP and GAP that has been investigated in (Krajči et al. 2004). Intended applications of Fuzzy LP languages include expert knowledge representation.

The second line includes Similarity-based LP (shortly SLP) in the sense of (Arcelli and Formato 1999; Sessa 2002; Loia et al. 2004) and related proposals, which keep the classical syntax of LP clauses but use a *similarity relation* over a set of symbols S to allow "flexible" unification of syntactically different symbols with a certain approximation degree. Similarity relations over a given set S have been defined in (Zadeh 1971; Sessa 2002) and related literature as fuzzy relations represented by mappings $\mathcal{S} : S \times S \to [0, 1]$ which satisfy reflexivity, symmetry and transitivity axioms analogous to those required for classical equivalence relations. A more general notion called *proximity relation* was introduced in (Dubois and Prade 1980) by omitting the transitivity axiom. As noted by (Shenoi and Melton 1999) and other authors, the transitivity property required for similarity relations may conflict with user's intentions in some cases. The Bousi∼Prolog language (Julián-Iranzo et al. 2009; Julián-Iranzo and Rubio-Manzano 2009b; Julián-Iranzo and Rubio-Manzano 2009a) has been designed with the aim of generalizing SLP to work with proximity relations. A different generalization of SLP is the SQLP scheme (Caballero et al. 2008), designed as an extension of the QLP scheme. In addition to clause annotations in QLP style, SQLP uses a given similarity relation $\mathcal{S} : S \times S \to D$ (where D is the carrier set of a parametrically given qualification domain) in order to support flexible unification. In the sequel we use the acronym SLP as including proximity-based LP languages also. Intended applications of SLP include flexible query answering. An analogy of proximity relations in a different context (namely partial constraint satisfaction) can be found in (Freuder and Wallace 1992), where several metrics are proposed to measure the proximity between the solution sets of two different constraint satisfaction problems.

Several of the above mentioned LP extensions (including GAP, QLP, the Fuzzy

LP language in (Guadarrama et al. 2004) and SQLP) have used constraint solving as an implementation technique. However, we only know two approaches which have been conceived as extensions of the classical CLP scheme (Jaffar and Lassez 1987). Firstly, (Riezler 1996; Riezler 1998) extended the formulation of CLP by (Höhfeld and Smolka 1988) with quantitative LP in the sense of (van Emden 1986); this work was motivated by problems from the field of natural language processing. Secondly, (Bistarelli et al. 2001) proposed a semiring-based approach to CLP, where constraints are solved in a soft way with levels of consistency represented by values of a semiring. This approach was motivated by constraint satisfaction problems and implemented with `clp(FD,S)` in (Georget and Codognet 1998) for a particular class of semirings which enable to use local consistency algorithms. The relationship between (Riezler 1996; Riezler 1998; Bistarelli et al. 2001) and the results of this report will be further discussed in Section 4.

Finally, there are a few preliminary attempts to combine some of the above mentioned approaches with the Functional Logic Programming (shortly FLP) paradigm found in languages such as Curry (Hanus) and $\mathcal{TOY}$ (Arenas et al. 2007). Similarity-based unification for FLP languages has been investigated by (Moreno and Pascual 2007), while (Caballero et al. 2009) have proposed a generic scheme QCFLP designed as a common extension of the two schemes CLP and QLP with first-order FLP features.

In this report we propose a new extension of CLP that supports qualification values and proximity relations. More precisely, we define a generic scheme SQCLP whose instances SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$) are parameterized by a proximity relation $\mathcal{S}$, a qualification domain $\mathcal{D}$ and a constraint domain $\mathcal{C}$. We will show that several previous proposals can be viewed as particular cases of SQCLP, obtained by partial instantiation. Moreover, we will present a declarative semantics for SQCLP that is inspired in the observable CLP semantics by (Gabbrielli and Levi 1991; Gabbrielli et al. 1995) and provides fixpoint and proof-theoretical characterizations of least program models as well as an implementation-independent notion of goal solution that can be used to specify the expected behavior of goal solving systems.

The reader is assumed to be familiar with the semantic foundations of LP (Lloyd 1987; Apt 1990) and CLP (Jaffar and Lassez 1987; Jaffar et al. 1998). The rest of the report is structured as follows: Section 2 introduces constraint domains, qualification domains and proximity relations. Section 3 presents the SQCLP scheme and the main results on its declarative semantics. Finally, Section 4 concludes by giving an overview of related approaches (many of which can be viewed as particular cases of SQCLP) and pointing to some lines open for future work.

2 Constraints, Qualification & Proximity

2.1 Constraint Domains

The Constraint Logic Programming paradigm (CLP) was introduced in (Jaffar and Lassez 1987) with the aim of generalizing the Herbrand Universe which underlies classical Logic Programming (LP) to other domains tailored to specific applica-

tion areas. In this seminal paper, CLP was introduced as a generic scheme with instances CLP($\mathcal{C}$) parameterized by *constraint domains* $\mathcal{C}$, each of which supplies several items: a *constraint language* providing a class of domain specific formulae, called *constraints* and serving as logical conditions in CLP($\mathcal{C}$) programs and computations; a *constraint structure* serving as interpretation of the constraint language; a *constraint theory* serving as a basis for proof-theoretical deduction with constraints; and a *constraint solver* for checking constraint satisfiability. Certain assumptions were made to ensure the proper relationship between the constraint language, structure, theory and solver, so that the classical results on the operational and declarative semantics of LP (Lloyd 1987; Apt 1990) could be extended to all the CLP($\mathcal{C}$) languages. A revised and updated presentation of the main results from (Jaffar and Lassez 1987) can be found in (Jaffar et al. 1998), while a survey of CLP as a programming paradigm is given in (Jaffar and Maher 1994).

The notion of constraint domain is a key ingredient of the CLP scheme. In addition to the classical formulation in (Jaffar and Lassez 1987; Jaffar et al. 1998), other formalizations have been used for different purposes. Some significative examples are: the CLP scheme proposed in (Höhfeld and Smolka 1988), motivated by applications to computational linguistics and allowing more than one constraint structure to come along with a given constraint language; the proof-theoretical notion of constraint system given in (Saraswat 1992), intended for application to concurrent constraint languages; and the constraint systems proposed in (Lucio et al. 2008) as the basis of a functorial semantics for CLP with negation, where a single constraint structure is replaced by a class of elementary equivalent structures.

In this paper we will use a simple notion of constraint domain, motivated by three main considerations: firstly, to focus on declarative semantics, rather than proof-theoretic or operational issues; secondly, to provide a purely relational framework; and thirdly, to clarify the interplay between domain-specific programming resources such as basic values and primitive predicates, and general-purpose programming resources such as data constructors and defined predicates.

2.1.1 Preliminary notions

Before presenting constraint domains in a formal way, let us introduce some mainly syntactic notions that will be used all along the paper.

Definition 2.1 (*Signatures*)
We assume a *universal programming signature* $\Gamma = \langle DC, DP \rangle$ where $DC = \bigcup_{n\in\mathbb{N}} DC^n$ and $DP = \bigcup_{n\in\mathbb{N}} DP^n$ are infinite and mutually disjoint sets of free function symbols (called *data constructors* in the sequel) and *defined predicate* symbols, respectively, ranked by arities. We will use *domain specific signatures* $\Sigma = \langle DC, DP, PP \rangle$ extending Γ with a disjoint set $PP = \bigcup_{n\in\mathbb{N}} PP^n$ of *primitive predicate* symbols, also ranked by arities. The idea is that primitive predicates come along with constraint domains, while defined predicates are specified in user programs. Each PP^n maybe any countable set of n-ary predicate symbols. In practice, PP is expected to be a finite set. □

In the sequel, we assume that any signature Σ includes two nullary constructors $\mathsf{true}, \mathsf{false} \in DC^0$ to represent the boolean values, a binary constructor $\mathsf{pair} \in DC^2$ to represent ordered pairs, as well as constructors to represent lists and other common data structures. Given a signature Σ, a set B of *basic values* u and a countably infinite set $\mathcal{V}ar$ of variables X, *terms* and *atoms* are built as defined below, where $\overline{o}_n$ abbreviates the n-tuple of syntactic objects $o_1, \ldots, o_n$ and $\mathrm{var}(o)$ denotes the set of all variables occurring in the syntactic object o.

Definition 2.2 (*Terms and atoms*)

- *Constructor Terms* $t \in \mathrm{Term}(\Sigma, B, \mathcal{V}ar)$ have the syntax $t ::= X | u | c(\overline{t}_n)$, where $c \in DC^n$. They will be called just terms in the sequel. In concrete examples, we will use Prolog syntax for terms built with list constructors, and we will write (t_1, t_2) rather than $\mathsf{pair}(t_1, t_2)$ for terms representing ordered pairs.
- The set of all the variables occurring in t is noted as $\mathrm{var}(t)$. A term t is called *ground* iff $\mathrm{var}(t) = \emptyset$. $\mathrm{Term}(\Sigma, B)$ stands for the set of all ground terms.
- *Atoms* $A \in \mathrm{At}(\Sigma, B, \mathcal{V}ar)$ can be *defined atoms* $r(\overline{t}_n)$, where $r \in DP^n$ and $t_i \in \mathrm{Term}(\Sigma, B, \mathcal{V}ar)$ $(1 \leq i \leq n)$; *primitive atoms* $p(\overline{t}_n)$, where $p \in PP^n$ and $t_i \in \mathrm{Term}(\Sigma, B, \mathcal{V}ar)$ $(1 \leq i \leq n)$; and *equations* $t_1 == t_2$, where $t_1, t_2 \in \mathrm{Term}(\Sigma, B, \mathcal{V}ar)$ and '==' is the *equality symbol*, which does not belong to the signature Σ. Primitive atoms are noted as κ and the set of all primitive atoms is noted $\mathrm{PAt}(\Sigma, B, \mathcal{V}ar)$. Equations and primitive atoms are collectivelly called $\mathcal{C}$*-based atoms*.
- The set of all the variables occurring in A is noted as $\mathrm{var}(A)$. An atom A is called *ground* iff $\mathrm{var}(A) = \emptyset$. The set of all ground atoms (resp. ground primitive atoms) is noted as $\mathrm{GAt}(\Sigma, B)$ (resp. $\mathrm{GPAt}(\Sigma, B)$). □

Note that the equality symbol '==' used as part of the syntax of equational atoms is not the same as the symbol '=' generally used for mathematical equality. In particular, metalevel equations $o = o'$ can be used to assert the identity of two syntactical objects o and o'.

Following well-known ideas, the syntactical structure of terms and atoms can be represented by means of trees with nodes labeled by signature symbols, basic values and variables. In the sequel we will use the notation $\|t\|$ to denote the *syntactical size* of t measured as the number of nodes in the tree representation of t. The *positions* of nodes in this tree can be noted as finite sequences p of natural numbers. In particular, the empty sequence ε represents the root position. The next definition presents essential notions concerning positions in terms. Positions in atoms can be treated similarly.

Definition 2.3 (*Positions*)

1. The set $\mathrm{pos}(t)$ of positions of the term t is defined by recursion on the structure of t:
   - $\mathrm{pos}(X) = \{\varepsilon\}$ for each variable $X \in \mathcal{V}ar$.
   - $\mathrm{pos}(u) = \{\varepsilon\}$ for each basic value $u \in B$.
   - $\mathrm{pos}(c(\overline{t}_n)) = \{\varepsilon\} \cup \bigcup_{i=1}^{n} \{iq \mid q \in \mathrm{pos}(t_i)\}$ for each $c \in DC^n$.

2. Given $p \in \mathrm{pos}(t)$, the symbol $t \circ p$ of t at position p is defined recursively:
   - $X \circ \varepsilon = X$ for each variable $X \in \mathcal{V}ar$.
   - $u \circ \varepsilon = u$ for each basic value $u \in B$.
   - $c(t_1, \ldots, t_n) \circ \varepsilon = c$ if $c \in DC^n$.
   - $c(t_1, \ldots, t_n) \circ iq = t_i \circ q$ if $c \in DC^n$, $1 \leq i \leq n$ and $q \in \mathrm{vpos}(t_i)$.
3. Given $p \in \mathrm{pos}(t)$, the subterm $t|_p$ of t at position p is defined as follows:
   - $t|_\varepsilon = t$ for any t.
   - $c(t_1, \ldots, t_n)|_{iq} = t_i|_q$ if $c \in DC^n$, $1 \leq i \leq n$ and $q \in \mathrm{pos}(t_i)$.
4. $p \in \mathrm{pos}(t)$ is called a *variable position* of t iff $t|p$ is a variable, and a *rigid position* of t otherwise. We define $\mathrm{vpos}(t) = \{p \in \mathrm{pos}(t) \mid p \text{ is a variable position}\}$ and $\mathrm{rpos}(t) = \{p \in \mathrm{pos}(t) \mid p \text{ is a rigid position}\}$.
5. Given $p \in \mathrm{vpos}(t)$ and another term s, the result of replacing s for the subterm of t at position p is noted as $t[s]_p$. See e.g. (Baader and Nipkow 1998) for a recursive definition. □

As usual, *substitutions* are defined as mappings $\sigma : \mathcal{V}ar \to \mathrm{Term}(\Sigma, B, \mathcal{V}ar)$ assigning terms to variables. The set of all substitutions is noted as $\mathrm{Subst}(\Sigma, B, \mathcal{V}ar)$. Substitutions are extended to act over terms and other syntactic objects o in the natural way. By convention, the result of replacing each variable X occurring in o by $\sigma(X)$ is noted as $o\sigma$. Other common notions concerning substitutions are defined as follows:

Definition 2.4 (*Notions concerning Substitutions*)

- The *composition* $\sigma\sigma'$ of two substitutions is such that $o(\sigma\sigma')$ equals $(o\sigma)\sigma'$.
- For a given $\sigma \in \mathrm{Subst}(\Sigma, B, \mathcal{V}ar)$, the *domain* $\mathrm{dom}(\sigma)$ is defined as $\{X \in \mathcal{V}ar \mid X\sigma \neq X\}$, and the *variable range* $\mathrm{vran}(\sigma)$ is defined as $\bigcup_{X \in \mathrm{dom}(\sigma)} \mathrm{var}(X\sigma)$.
- A substitution σ is called *ground* iff $X\sigma$ is a ground term for all $X \in \mathrm{dom}(\sigma)$. The set of all ground substitutions is noted $\mathrm{GSubst}(\Sigma, B)$.
- A substitution σ is called *finite* iff $\mathrm{dom}(\sigma)$ is a finite set, say $\{X_1, \ldots, X_k\}$. In this case, σ can be represented as the *set of bindings* $\{X_1 \mapsto t_1, \ldots, X_k \mapsto t_k\}$, where $t_i = X_i\sigma$ for all $1 \leq i \leq k$.
- Assume two substitutions σ, σ', a set of variables $\mathcal{X}$ and a variable Y. The notation $\sigma =_{\mathcal{X}} \sigma'$ means that $X\sigma = X\sigma'$ holds for all variables $X \in \mathcal{X}$. We also write $\sigma =_{\setminus \mathcal{X}} \sigma'$ and $\sigma =_{\setminus Y} \sigma'$ to abbreviate $\sigma =_{\mathcal{V}ar \setminus \mathcal{X}} \sigma'$ and $\sigma =_{\mathcal{V}ar \setminus \{Y\}} \sigma'$, respectively. □

2.1.2 Constraint domains, constraints and their solutions

We are now prepared to present constraint domains as mathematical structures providing a set of basic values along with an terms and an interpretation of primitive predicates[1]. The formal definition is as follows:

[1] As we will see in Section 3, the interpretation of defined predicate symbols is program dependent.

Definition 2.5 (*Constraint Domains*)
A *Constraint Domain* of signature Σ is any relational structure of the form $\mathcal{C} = \langle C, \{p^{\mathcal{C}} \mid p \in PP\}\rangle$ such that:

1. The carrier set C is $\mathrm{Term}(\Sigma, B)$ for a certain set B of *basic values*. When convenient, we note B and C as $B_{\mathcal{C}}$ and $C_{\mathcal{C}}$, respectively.
2. $p^{\mathcal{C}} : C^n \to \{0,1\}$, written simply as $p^{\mathcal{C}} \in \{0,1\}$ in the case $n = 0$, is called the *interpretation* of p in $\mathcal{C}$. A ground primitive atom $p(\bar{t}_n)$ is *true* in $\mathcal{C}$ iff $p^{\mathcal{C}}(\bar{t}_n) = 1$; otherwise $p(\bar{t}_n)$ is *false* in $\mathcal{C}$. □

For the examples in this paper we will use a constraint domain $\mathcal{R}$ which allows to work with arithmetic constraints over the real numbers, as formalized in Definition 2.6 below.

Definition 2.6 (*The Real Constraint Domain* $\mathcal{R}$)
The constraint domain $\mathcal{R}$ is defined to include:

- The set of basic values $B_{\mathcal{R}} = \mathbb{R}$. Note that $C_{\mathcal{R}}$ includes ground terms built from real values and data constructors, in addition to real numbers.
- Primitive predicates for encoding the usual arithmetic operations over $\mathbb{R}$. For instance, the addition operation $+$ over $\mathbb{R}$ is encoded by a ternary primitive predicate op_+ such that, for any $t_1, t_2 \in C_{\mathcal{R}}$, $op_+(t_1, t_2, t)$ is true in $\mathcal{R}$ iff $t_1, t_2, t \in \mathbb{R}$ and $t_1 + t_2 = t$. In particular, $op_+(t_1, t_2, t)$ is false in $\mathcal{R}$ if either t_1 or t_2 includes data constructors. The primitive predicates encoding other arithmetic operations such as $\times$ and $-$ are defined analogously.
- Primitive predicates for encoding the usual inequality relations over $\mathbb{R}$. For instance, the ordering $\leq$ over $\mathbb{R}$ is encoded by a binary primitive predicate $cp_\leq$ such that, for any $t_1, t_2 \in C_{\mathcal{R}}$, $cp_\leq(t_1, t_2)$ is true in $\mathcal{R}$ iff $t_1, t_2, t \in \mathbb{R}$ and $t_1 \leq t_2$. In particular, $cp_\leq(t_1, t_2)$ is false in $\mathcal{R}$ if either t_1 or t_2 includes data constructors. The primitive predicates encoding the other inequality relations, namely $>$, $\geq$ and $>$, are defined analogously. □

The domain $\mathcal{R}$ is well known as the basis of the CLP($\mathcal{R}$) language and system (Jaffar et al. 1992). Some presentations of $\mathcal{R}$ known in the literature represent the arithmetical operations by using primitive functions instead of primitive predicates. In this paper we have chosen to work in a purely relational framework in order to simplify some technicalities without loss of real expressivity.

Other useful instances of constraint domains are known in the Constraint Programming literature; see e.g. (Jaffar and Maher 1994; López-Fraguas et al. 2007). In particular, the *Herbrand* domain $\mathcal{H}$ is intended to work just with equality constraints, while $\mathcal{FD}$ allows to work with constraints involving *finite domain variables*. The set of basic values of $\mathcal{FD}$ is $\mathbb{Z}$. There are also known techniques for combining several given constraint domains into a more expressive one; see e.g. the *coordination domains* defined in (Estévez-Martín et al. 2009).

The following definition introduces constraints over a given domain:

Definition 2.7 (*Constraints and Their Solutions*)
Given a constraint domain $\mathcal{C}$ of signature Σ:

1. *Atomic constraints* over $\mathcal{C}$ are of two kinds: primitive atoms $p(\bar{t}_n)$ and equations $t_1 == t_2$.
2. *Compound constraints* are built from atomic constraints using logical conjunction $\wedge$, existential quantification $\exists$, and sometimes other logical operations. Constraints of the form $\exists X_1 \ldots \exists X_n (B_1 \wedge \ldots \wedge B_m)$ –where B_j $(1 \leq j \leq m)$ are atomic– are called *existential*. The set of all constraints over $\mathcal{C}$ is noted $\mathrm{Con}_{\mathcal{C}}$.
3. Substitutions $\sigma : \mathcal{V}ar \to \mathrm{Term}(\Sigma, B, \mathcal{V}ar)$ where $\mathrm{Term}(\Sigma, B, \mathcal{V}ar)$ is built using the set $B_{\mathcal{C}}$ of basic values are called *$\mathcal{C}$-substitutions*. Ground substitutions $\eta \in \mathrm{GSubst}(\Sigma, B)$ are called *variable valuations*. The set of all possible variable valuations is noted $\mathrm{Val}_{\mathcal{C}}$.
4. *The solution set* $\mathrm{Sol}_{\mathcal{C}}(\pi)$ of a constraint $\pi \in \mathrm{Con}_{\mathcal{C}}$ is defined by recursion on π's syntactic structure as follows:
   - If π is a primitive atom $p(\bar{t}_n)$, then $\mathrm{Sol}_{\mathcal{C}}(\pi)$ is the set of all $\eta \in \mathrm{Val}_{\mathcal{C}}$ such that $p(\bar{t}_n)\eta$ is ground and true in $\mathcal{C}$.
   - If π is an equation $t_1 == t_2$, then $\mathrm{Sol}_{\mathcal{C}}(\pi)$ is the set of all $\eta \in \mathrm{Val}_{\mathcal{C}}$ such that $t_1\eta$ and $t_2\eta$ are ground and syntactically identical terms.
   - If π is $\pi_1 \wedge \pi_2$ then $\mathrm{Sol}_{\mathcal{C}}(\pi) = \mathrm{Sol}_{\mathcal{C}}(\pi_1) \cap \mathrm{Sol}_{\mathcal{C}}(\pi_2)$.
   - If π is $\exists X \pi'$ then $\mathrm{Sol}_{\mathcal{C}}(\pi)$ is the set of all $\eta \in \mathrm{Val}_{\mathcal{C}}$ such that $\eta' \in \mathrm{Sol}_{\mathcal{C}}(\pi')$ holds for some $\eta' \in \mathrm{Val}_{\mathcal{C}}$ verifying $\eta =_{\setminus X} \eta'$.

   π is called *satisfiable* over $\mathcal{C}$ iff $\mathrm{Sol}_{\mathcal{C}}(\pi) \neq \emptyset$, and π is called *unsatisfiable* over $\mathcal{C}$ iff $\mathrm{Sol}_{\mathcal{C}}(\pi) = \emptyset$.
5. *The solution set* $\mathrm{Sol}_{\mathcal{C}}(\Pi)$ of a set Π of constraints is defined as $\bigcap_{\pi \in \Pi} \mathrm{Sol}_{\mathcal{C}}(\pi)$. In this way, finite sets of constraints are interpreted as the conjunction of their members. Π is called *satisfiable* over $\mathcal{C}$ iff $\mathrm{Sol}_{\mathcal{C}}(\Pi) \neq \emptyset$, and Π is called *unsatisfiable* over $\mathcal{C}$ iff $\mathrm{Sol}_{\mathcal{C}}(\Pi) = \emptyset$.
6. A constraint π *is entailed* by a set of constraints Π (in symbols, $\Pi \models_{\mathcal{C}} \pi$) iff $\mathrm{Sol}_{\mathcal{C}}(\Pi) \subseteq \mathrm{Sol}_{\mathcal{C}}(\pi)$. $\square$

The following example illustrates the previous definition:

Example 2.1 (*Constraint solutions and constraint entailment over $\mathcal{R}$*)
Consider the set of constraints $\Pi = \{cp_{\geq}(A, 3.0),\ op_{+}(A, A, X), op_{\times}(2.0, A, Y)\} \subseteq \mathrm{Con}_{\mathcal{R}}$. Then:

1. For any valuation $\eta \in \mathrm{Val}_{\mathcal{R}}$: $\eta \in \mathrm{Sol}_{\mathcal{R}}(\Pi)$ holds iff $\eta(A)$, $\eta(X)$ and $\eta(Y)$ are real numbers $a, x, y \in \mathbb{R}$ such that $a \geq 3.0$, $a + a = x$ and $2.0 \times a = y$.
2. Due to the previous item, the following $\mathcal{R}$-entailments are valid:
   (a) $\Pi \models_{\mathcal{R}} cp_{>}(X, 5.5)$, because $\mathrm{Sol}_{\mathcal{R}}(\Pi) \subseteq \mathrm{Sol}_{\mathcal{R}}(cp_{>}(X, 5.5))$.
   (b) $\Pi \models_{\mathcal{R}} X == Y$, because $\mathrm{Sol}_{\mathcal{R}}(\Pi) \subseteq \mathrm{Sol}_{\mathcal{R}}(\mathrm{X == Y})$.
   (c) $\Pi \models_{\mathcal{R}} c(X) == c(Y)$, because $\mathrm{Sol}_{\mathcal{R}}(\Pi) \subseteq \mathrm{Sol}_{\mathcal{R}}(\mathrm{c(X) == c(Y)})$. Here we assume $c \in DC^1$.

(d) $\Pi \models_{\mathcal{R}} [X,Y] == [Y,X]$, because $\mathrm{Sol}_{\mathcal{R}}(\Pi) \subseteq \mathrm{Sol}_{\mathcal{R}}([\mathrm{X},\mathrm{Y}] == [\mathrm{Y},\mathrm{X}])$. Here, the terms $[X,Y]$ and $[Y,X]$ are built from variables and list constructors. □

The next technical result will be useful later on:

Lemma 2.1 (*Substitution Lemma*)
Assume a set of constraints $\Pi \subseteq \mathrm{Con}_{\mathcal{C}}$ and a $\mathcal{C}$-substitution σ. Then:

1. For any valuation $\eta \in \mathrm{Val}_{\mathcal{C}}$: $\eta \in \mathrm{Sol}_{\mathcal{C}}(\Pi\sigma) \iff \sigma\eta \in \mathrm{Sol}_{\mathcal{C}}(\Pi)$.
2. For any constraint $\pi \in \mathrm{Con}_{\mathcal{C}}$: $\Pi \models_{\mathcal{C}} \pi \Longrightarrow \Pi\sigma \models_{\mathcal{C}} \pi\sigma$.

Proof
Let us give a separate reasoning for each item.

1. The following statement holds for any constraint $\pi \in \mathrm{Con}_{\mathcal{C}}$:

$$(\star)\ \ \eta \in \mathrm{Sol}_{\mathcal{C}}(\pi\sigma) \iff \sigma\eta \in \mathrm{Sol}_{\mathcal{C}}(\pi)$$

In fact, $(\star)$ can can be easily proved reasoning by induction on the syntactic structure of π. Now, using $(\star)$ we can reason as follows:

$$\eta \in \mathrm{Sol}_{\mathcal{C}}(\Pi\sigma) \iff \eta \in \mathrm{Sol}_{\mathcal{C}}(\pi\sigma) \text{ for all } \pi \in \Pi \iff_{(\star)} \sigma\eta \in \mathrm{Sol}_{\mathcal{C}}(\pi) \text{ for all } \pi \in \Pi \iff \sigma\eta \in \mathrm{Sol}_{\mathcal{C}}(\Pi)$$

2. Assume $\Pi \models_{\mathcal{C}} \pi$. For the sake of proving $\Pi\sigma \models_{\mathcal{C}} \pi\sigma$, also assume an arbitrary $\eta \in \mathrm{Sol}_{\mathcal{C}}(\Pi\sigma)$. Then we get $\sigma\eta \in \mathrm{Sol}_{\mathcal{C}}(\Pi)$ because of item 1 and $\sigma\eta \in \mathrm{Sol}_{\mathcal{C}}(\pi)$ due to the assumption $\Pi \models_{\mathcal{C}} \pi$, which implies $\eta \in \mathrm{Sol}_{\mathcal{C}}(\pi\sigma)$ again because of item 1. Since η is arbitrary, we have proved $\mathrm{Sol}_{\mathcal{C}}(\Pi\sigma) \subseteq \mathrm{Sol}_{\mathcal{C}}(\pi\sigma)$, i.e. $\Pi\sigma \models_{\mathcal{C}} \pi\sigma$. □

2.1.3 Term equivalence w.r.t. a given constraint set

Given two terms t, s we will use the notation $t \approx_{\Pi} s$ (read as t and s are Π-*equivalent*) as an abbreviation of $\Pi \models_{\mathcal{C}} t == s$, assuming that the constraint domain $\mathcal{C}$ and the constraint set $\Pi \subseteq \mathrm{Con}_{\mathcal{C}}$ are known. For the sake of simplicity, $\mathcal{C}$ is not made explicit in the $\approx_{\Pi}$ notation. In this subsection we present some properties related to $\approx_{\Pi}$ which will be needed later. First, we prove that $\approx_{\Pi}$ is an equivalence relation with a natural characterization.

Lemma 2.2 (Π-*Equivalence Lemma*)
1. $\approx_{\Pi}$ is an equivalence relation over $\mathrm{Term}(\Sigma, B, \mathcal{V}ar)$.
2. For any given terms t and s the following two statements are equivalent:
   (a) $t \approx_{\Pi} s$.
   (b) For any common position $p \in \mathrm{pos}(t) \cap \mathrm{pos}(s)$ some of the cases below holds:
      i $t \circ p$ or $s \circ p$ is a variable, and moreover $t|_p \approx_{\Pi} s|_p$.
      ii $t \circ p = s \circ p = u$ for some $u \in B_{\mathcal{C}}$.
      iii $t \circ p = s \circ p = c$ for some $n \in \mathbb{N}$ and some $c \in DC^n$.
3. $\approx_{\Pi}$ boils down to the syntactic equality relation $=$ when Π is the empty set.

Proof
We give a separate reasoning for each item.

1. Checking that $\approx_\Pi$ satisfies the axioms of an equivalence relation (i.e. *reflexivity*, *symmetry* and *transitivity*) is quite obvious.
2. Due to Definition 2.7, $t \approx_\Pi s$ holds iff $t\eta$ and $s\eta$ are identical ground terms for each $\eta \in \mathrm{Sol}_\mathcal{C}(\Pi)$. This statement can be proved equivalent to condition 2.(b) reasoning by induction on $\|t\| + \|s\|$.
3. Note that $t \approx_\emptyset s$ holds iff $t\eta$ and $s\eta$ are identical ground terms for each $\eta \in \mathrm{Sol}_\mathcal{C}(\emptyset) = \mathrm{Val}_\mathcal{C}$. This can happen iff t and s are syntactically identical. □

Since the set $\mathcal{V}ar$ of all variables is countably infinite, we can assume an arbitrarily fixed bijective mapping $\mathrm{ord} : \mathcal{V}ar \to \mathbb{N}$. By convention, $\mathrm{ord}(X)$ is called the ordinal number of X. The notions defined below rely on this convention.

Definition 2.8 (Π-*Canonical Variables and Terms*)

1. A variable X is called Π-*canonical* iff there is no other variable X' such that $X \approx_\Pi X'$ and $\mathrm{ord}(X') < \mathrm{ord}(X)$.
2. For each variable X its Π-*canonical form* $\mathrm{cf}_\Pi(X)$ is defined as the member of the set $\{X' \in \mathcal{V}ar \mid X \approx_\Pi X'\}$ with the least ordinal number.
3. A term t is called Π-*canonical* iff all the variables occurring in t are Π-canonical.
4. For each term t its Π-*canonical form* $\mathrm{cf}_\Pi(t)$ is defined as the result of replacing $\mathrm{cf}_\Pi(X)$ for each variable X occurring in t. □

The following lemma states some obvious properties of terms in canonical form:

Lemma 2.3 (Π-*Canonicity Lemma*)
For each term t, $\mathrm{cf}_\Pi(t)$ is Π-canonical and such that $t \approx_\Pi \mathrm{cf}_\Pi(t)$. Moreover, t and $\mathrm{cf}_\Pi(t)$ have the same positions and structure, except that each variable X occurring at some position $p \in \mathrm{vpos}(t)$ is replaced by an occurrence of $\mathrm{cf}_\Pi(X)$ at the same position p in $\mathrm{cf}_\Pi(t)$.

Proof
Straightforward consequence of the construction of $\mathrm{cf}_\Pi(t)$ from t and the Π-Equivalence Lemma 2.2. □

Given two terms t and s, the term built from t by replacing within t each variable X occurring at some position $p \in \mathrm{vpos}(t) \cap \mathrm{pos}(s)$ by the subterm $s|_p$ is called the *extension of* t *w.r.t. to* s and noted as $t \ll s$ (or equivalently, $s \gg t$). A more precise definition of this notion and some related properties are given below.

Definition 2.9 (*Term extension*)
Given any two terms t and s, the *extension of* t *w.r.t.* s is defined by recursion on the syntactical structure of t:

- $X \ll s = s$ for each variable $X \in \mathcal{V}ar$.
- $u \ll s = u$ for each basic value $u \in B$.

- $c(t_1,\ldots,t_n) \ll s = c(t_1 \ll s_1,\ldots,t_n \ll s_n)$ if $c \in DC^n$ and there is some $c' \in DC^n$ such that $s = c'(s_1,\ldots,s_n)$.
- $c(t_1,\ldots,t_n) \ll s = c(t_1,\ldots,t_n)$ if $c \in DC^n$ and there is no $c' \in DC^n$ such that $s = c'(s_1,\ldots,s_n)$. □

Lemma 2.4 (*Extension Lemma*)
The term extension operation $\ll$ enjoys the two following properties:

1. *Symmetrical Extension Property:*
   Let t', t'' be Π-canonical terms such that $t' \approx_\Pi t''$. Under this assumption $(t' \ll t'') = (t'' \ll t')$.
2. Π-*Equivalence Extension Property:*
   Let the terms t, s be such that for any $p \in \mathrm{pos}(t)$ with $t|_p = X \in \mathcal{V}ar$ one has $p \in \mathrm{pos}(s)$ and $X \approx_\Pi s|_p$. Under this assumption $t \approx_\Pi (t \ll s)$.

Proof of Symmetrical Extension Property
Recall that the hypothesis $t' \approx_\Pi t''$ means that $\Pi \models_{\mathcal{C}} t' == t''$. We reason by complete induction on $\|t'\| + \|t''\|$. There are five possible cases:

1. $t' == t''$ is $c'(\overline{t'}_n) == c''(\overline{t''}_n)$ for some $n \in \mathbb{N}$, $c', c'' \in DC^n$. In this case, the Π-Equivalence Lemma 2.2 ensures that $c' = c'' = c \in DC^n$ and $t'_i \approx_\Pi t''_i$ holds for all $1 \leq i \leq n$. Clearly, the terms t'_i, t''_i are Π-canonical. Therefore, by induction hypothesis we can assume $(t'_i \ll t''_i) = (t''_i \ll t'_i)$ for all $1 \leq i \leq n$. Then, by definition of $\ll$ we get $t' \ll t'' = c(t'_1 \ll t''_1,\ldots,t'_n \ll t''_n) = c(t''_1 \ll t'_1,\ldots,t''_n \ll t'_n) = t'' \ll t'$.
2. $t' == t''$ is $u' == u''$ for some $u', u'' \in B$. In this case, $u' \approx_\Pi u''$ implies that $u' = u'' = u \in B$, and by definition of $\ll$ we get $t' \ll t'' = t'' \ll t' = u \ll u = u$.
3. $t' == t''$ is $X == Y$ for some $X, Y \in \mathcal{V}ar$. In this case, $X \approx_\Pi Y$ and X, Y Π-canonical implies that X, Y must be identical variables. By definition of $\ll$ we get $t' \ll t'' = t'' \ll t' = X \ll X = X$.
4. $t' == t''$ is $X == t''$ with $X \in \mathcal{V}ar$, $t'' \notin \mathcal{V}ar$. In this case, by definition of $\ll$ we get $t' \ll t'' = X \ll t'' = t''$ and $t'' \ll t' = t'' \ll X = t''$.
5. $t' == t''$ is $t' == Y$ with $Y \in \mathcal{V}ar$, $t' \notin \mathcal{V}ar$. In this case, by definition of $\ll$ we get $t' \ll t'' = t' \ll Y = t' = t''$ and $t'' \ll t' = Y \ll t' = t'$. □

Proof of Π-*Equivalence Extension Property*
Recall that the thesis $t \approx_\Pi (t \ll s)$ means that $\Pi \models_{\mathcal{C}} t == (t \ll s)$. We reason by complete induction on $\|t\|$. There are four possible cases:

1. t is a variable $X \in \mathcal{V}ar$. In this case, $X \ll s = s$ by definition of $\ll$, and $X \approx_\Pi s$ holds by hypothesis.
2. t is a basic value $u \in B$. In this case, $u \ll s = u$ by definition of $\ll$, and $u \approx_\Pi u$ holds trivially.
3. t is $c(\overline{t}_n)$ for some $c \in DC^n$ and there is no $c' \in DC^n$ such that s has the form $c'(\overline{s}_n)$. In this case, $c(\overline{t}_n) \ll s = c(\overline{t}_n)$ by definition of $\ll$, and $c(\overline{t}_n) \approx_\Pi c(\overline{t}_n)$ holds trivially.

4. t is $c(\overline{t}_n)$ for some $c \in DC^n$ and s is $c'(\overline{s}_n)$ for some $c' \in DC^n$. In this case $c(\overline{t}_n) \ll c'(\overline{s}_n) = c(t_1 \ll s_1, \ldots, t_n \ll s_n)$ by definition of $\ll$. Moreover, the assumptions of the Π-Equivalent Extension Property hold for the smaller terms t_i, s_i $(1 \leq i \leq n)$. By induction hypothesis we can assume $t_i \approx_\Pi (t_i \ll s_i)$ for all $1 \leq i \leq n$. Therefore, $c(\overline{t}_n) \approx_\Pi c(t_1 \ll s_1, \ldots, t_n \ll s_n)$ due to the Π-Equivalence Lemma 2.2. □

2.2 Qualification Domains

The intended role of *Qualification Domains* in an extended logic programming scheme SQCLP have been already explained in the Introduction. They were originally introduced in (Rodríguez-Artalejo and Romero-Díaz 2008b) and their axiomatic definition was extended with axioms for an additional operation $\oslash$ in (Rodríguez-Artalejo and Romero-Díaz 2009) in order to enable a particular implementation technique for program clauses with threshold conditions in their bodies. The definition given below is again closer to the original one: $\oslash$ is omitted and the axioms of the operator $\circ$ are slightly refined.

Definition 2.10 (*Qualification Domains*)
A *Qualification Domain* is any structure $\mathcal{D} = \langle D, \preccurlyeq, \mathbf{b}, \mathbf{t}, \circ \rangle$ verifying the following requirements:

1. D, noted as $D_\mathcal{D}$ when convenient, is a set of elements called *qualification values*.
2. $\langle D, \preccurlyeq, \mathbf{b}, \mathbf{t} \rangle$ is a lattice with extreme points $\mathbf{b}$ (called *infimum* or *bottom* element) and $\mathbf{t}$ (called *maximum* or *top* element) w.r.t. the partial ordering $\preccurlyeq$, called *qualification ordering*. For given elements $d, e \in D$, we write $d \sqcap e$ for the *greatest lower bound* (*glb*) of d and e, and $d \sqcup e$ for the *least upper bound* (*lub*) of d and e. We also write $d \lhd e$ as abbreviation for $d \preccurlyeq e \wedge d \neq e$.
3. $\circ : D \times D \to D$, called *attenuation operation*, verifies the following axioms:
   (a) $\circ$ is associative, commutative and monotonic w.r.t. $\preccurlyeq$.
   (b) $\forall d \in D : d \circ \mathbf{t} = d$.
   (c) $\forall d \in D : d \circ \mathbf{b} = \mathbf{b}$.
   (d) $\forall d, e \in D : d \circ e \preccurlyeq e$.
   (e) $\forall d, e_1, e_2 \in D : d \circ (e_1 \sqcap e_2) = (d \circ e_1) \sqcap (d \circ e_2)$. □

Actually, some of the properties of $\circ$ postulated as axioms in the previous definition are redundant.[2] More precisely:

Proposition 2.1 (*Redundant postulates of Qualification Domains*)
The properties (3)(c) and (3)(d) are redundant and can be derived from the other axioms in Definition 2.10.

[2] The authors are thankful to G. Gerla for pointing out this fact.

Proof
Note that $\circ$ is commutative and monotonic w.r.t. $\preccurlyeq$ because of axiom (3)(a). Since $\mathbf{t}$ is the top element of the lattice, $d \preccurlyeq \mathbf{t}$ holds for any $d \in D$. By monotonicity of $\circ$, $d \circ e \preccurlyeq \mathbf{t} \circ e$ also holds for any $e \in D$. By commutativity of $\circ$ and axiom (3)(b), $d \circ e \preccurlyeq \mathbf{t} \circ e$ is the same as $d \circ e \preccurlyeq e$. Therefore (3)(d) is a consequence of the other axioms postulated for $\circ$. In particular, taking $e = \mathbf{b}$ we get $d \circ \mathbf{b} \preccurlyeq \mathbf{b}$, which implies $d \circ \mathbf{b} = \mathbf{b}$ because $\mathbf{b}$ is the bottom element of the lattice. Hence, (3)(c) also follows form the other axioms. □

In the rest of the report, $\mathcal{D}$ will generally denote an arbitrary qualification domain. For any finite $S = \{e_1, e_2, \ldots, e_n\} \subseteq D$, the *greatest lower bound* (also called *infimum* of S and noted as $\bigsqcap S$) exists and can be computed as $e_1 \sqcap e_2 \sqcap \cdots \sqcap e_n$ (which reduces to $\top$ in the case $n = 0$). The dual claim concerning *least upper bounds* is also true. As an easy consequence of the axioms, one gets the identity $d \circ \bigsqcap S = \bigsqcap\{d \circ e \mid e \in S\}$.

Many useful qualification domains are such that $\forall d, e \in D \setminus \{\mathbf{b}\} : d \circ e \neq \mathbf{b}$. In the sequel, any qualification domain $\mathcal{D}$ that verifies this property will be called *stable*. Below we present some basic qualification domains which are clearly stable, along with brief explanations of their role for building extended CLP languages as instances of the SQCLP scheme proposed in this report. Checking that these domains satisfy the axioms given in Def. 2.10 is left as an easy exercise. In fact, the axioms have been chosen as a natural generalization of some basic properties satisfied by the ordering $\leq$ and the operation $\times$ over the real interval $[0, 1]$.

2.2.1 The Domain $\mathcal{B}$ of Classical Boolean Values

This domain is $\mathcal{B} =_{\text{def}} \langle\{0, 1\}, \leq, 0, 1, \wedge\rangle$, where 0 and 1 stand for the two classical truth values *false* and *true*, $\leq$ is the usual numerical ordering over $\{0, 1\}$, and $\wedge$ stands for the classical conjunction operation over $\{0, 1\}$.

2.2.2 The Domain $\mathcal{U}$ of Uncertainty Values and its variant $\mathcal{U}'$

This domain is $\mathcal{U} =_{\text{def}} \langle \mathrm{U}, \leq, 0, 1, \times\rangle$, where $\mathrm{U} = [0, 1] = \{d \in \mathbb{R} \mid 0 \leq d \leq 1\}$, $\leq$ is the usual numerical ordering, and $\times$ is the multiplication operation. The top element $\mathbf{t}$ is 1 and the greatest lower bound $\bigsqcap S$ of a finite $S \subseteq \mathrm{U}$ is the minimum value $\min(S)$, which is 1 if $S = \emptyset$. Elements of $\mathcal{U}$ are intended to represent certainty degrees as used in (van Emden 1986).

A slightly different domain $\mathcal{U}'$ can be defined as $\langle \mathrm{U}, \leq, 0, 1, \min\rangle$ where the only difference with respect to $\mathcal{U}$ is that in the case of $\mathcal{U}'$, $\circ = \min$.

2.2.3 The Domain $\mathcal{W}$ of Weight Values and related variants

This domain is $\mathcal{W} =_{\text{def}} \langle \mathrm{P}, \geq, \infty, 0, +\rangle$, where $\mathrm{P} = [0, \infty] = \{d \in \mathbb{R} \cup \{\infty\} \mid d \geq 0\}$, $\geq$ is the reverse of the usual numerical ordering (with $\infty \geq d$ for any $d \in \mathrm{P}$), and $+$ is the addition operation (with $\infty + d = d + \infty = \infty$ for any $d \in \mathrm{P}$). The top

element $\mathbf{t}$ is 0 and the greatest lower bound $\bigsqcap S$ of a finite $S \subseteq \mathrm{P}$ is the maximum value $\max(S)$, which is 0 if $S = \emptyset$. Elements of $\mathcal{W}$ are intended to represent proof costs, measured as the weighted depth of proof trees.

In analogy to the definition of $\mathcal{U}'$ as a variant of $\mathcal{U}$, we can define a qualification domain $\mathcal{W}'$ as $\langle \mathrm{P}, \geq, \infty, 0, \max \rangle$ with $\circ = \max$. Also, as a discrete variant of $\mathcal{W}$, we define the qualification domain $\mathcal{W}_d =_{\mathrm{def}} \langle \mathrm{P}, \geq, \infty, 0, + \rangle$ with the only difference w.r.t. $\mathcal{W}$ being that $\mathrm{P} = \mathbb{N} \cup \{\infty\}$. Elements of $\mathcal{W}_d$ are also intended to represent proof costs (represented by natural numbers in this case). Finally, a variant $\mathcal{W}'_d$ of $\mathcal{W}_d$ can be defined by replacing the attenuation operation in $\mathcal{W}_d$ by max.

2.2.4 Two product constructions

To close this section, we present two product constructions that can be used to build compound qualification domains. The mathematical definition is as follows:

Definition 2.11 (*Products of Qualification Domains*)
Let two qualification domains $\mathcal{D}_i = \langle D_i, \preccurlyeq_i, \mathbf{b}_i, \mathbf{t}_i, \circ_i \rangle$ $(i \in \{1, 2\})$ be given.

1. The *cartesian product* $\mathcal{D}_1 \times \mathcal{D}_2$ is defined as $\mathcal{D} =_{\mathrm{def}} \langle D, \preccurlyeq, \mathbf{b}, \mathbf{t}, \circ \rangle$ where $D =_{\mathrm{def}} D_1 \times D_2$, the partial ordering $\preccurlyeq$ is defined as $(d_1, d_2) \preccurlyeq (e_1, e_2) \Longleftrightarrow_{\mathrm{def}} d_1 \preccurlyeq_1 e_1$ and $d_2 \preccurlyeq_2 e_2$, $\mathbf{b} =_{\mathrm{def}} (\mathbf{b}_1, \mathbf{b}_2)$, $\mathbf{t} =_{\mathrm{def}} (\mathbf{t}_1, \mathbf{t}_2)$ and the attenuation operator $\circ$ is defined as $(d_1, d_2) \circ (e_1, e_2) =_{\mathrm{def}} (d_1 \circ_1 e_1, d_2 \circ_2 e_2)$.
2. Given two elements $d_1 \in D_1$ and $d_2 \in D_2$, the *strict pair* $(\!|\, d_1, d_2 \,|\!)$ is defined by case distinction as follows: if $d_1 \neq \mathbf{b}_1$ and $d_2 \neq \mathbf{b}_2$, then $(\!|\, d_1, d_2 \,|\!) = (d_1, d_2)$; if $d_1 = \mathbf{b}_1$ or $d_2 = \mathbf{b}_2$, then $(\!|\, d_1, d_2 \,|\!) = (\mathbf{b}_1, \mathbf{b}_2)$. In both cases, $(\!|\, d_1, d_2 \,|\!) \in D_1 \times D_2$.
3. The *strict cartesian product* $\mathcal{D}_1 \otimes \mathcal{D}_2$ is defined as $\mathcal{D} =_{\mathrm{def}} \langle D, \preccurlyeq, \mathbf{b}, \mathbf{t}, \circ \rangle$ where $D = D_1 \otimes D_2 =_{\mathrm{def}} \{(\!|\, d_1, d_2 \,|\!) \mid d_1 \in D_1, d_2 \in D_2\}$ (or equivalently, $D = ((D_1 \setminus \{\mathbf{b}_1\}) \times (D_2 \setminus \{\mathbf{b}_2\})) \cup \{(\mathbf{b}_1, \mathbf{b}_2)\}$), the partial ordering $\preccurlyeq$ is defined as $(d_1, d_2) \preccurlyeq (e_1, e_2) \Longleftrightarrow_{\mathrm{def}} d_1 \preccurlyeq_1 e_1$ and $d_2 \preccurlyeq_2 e_2$, $\mathbf{b} =_{\mathrm{def}} (\!|\, \mathbf{b}_1, \mathbf{b}_2 \,|\!) = (\mathbf{b}_1, \mathbf{b}_2)$, $\mathbf{t} =_{\mathrm{def}} (\!|\, \mathbf{t}_1, \mathbf{t}_2 \,|\!)$, and the attenuation operator $\circ$ is defined as $(d_1, d_2) \circ (e_1, e_2) =_{\mathrm{def}} (d_1 \circ_1 e_1, d_2 \circ_2 e_2)$. Note the special case when D_1 or D_2 is a singleton set. Then, D is the singleton set $\{(\mathbf{b}_1, \mathbf{b}_2)\}$, $(\!|\, \mathbf{t}_1, \mathbf{t}_2 \,|\!) = (\mathbf{b}_1, \mathbf{b}_2)$, and $(\mathbf{t}_1, \mathbf{t}_2) \in D$ happens to be false if one of the two sets D_1, D_2 is not a singleton. □

Intuitively, each value (d_1, d_2) belonging to a product domain $\mathcal{D}_1 \times \mathcal{D}_2$ or $\mathcal{D}_1 \otimes \mathcal{D}_2$ imposes the qualification d_1 *and also* the qualification d_2. In particular, values (c, d) belonging to the product domains $\mathcal{U} \times \mathcal{W}$ and $\mathcal{U} \otimes \mathcal{W}$ impose two qualifications, namely: a certainty value greater or equal than c and a proof tree with weighted depth less or equal than d. This intuition indeed corresponds to the declarative semantics formally defined in Section 3.

The next theorem shows that the class of the qualification domains is closed under ordinary cartesian products, while the subclass of stable qualification domains is closed under strict cartesian products. We are particularly interested in stable

qualification domains built from basic domains by reiterated strict products, because they can be encoded into into constraint domains in the sense explained in Subsection 2.2.5 below.

Theorem 2.1
Assume two given qualification domains $\mathcal{D}_1$ and $\mathcal{D}_2$. Then the ordinary cartesian product $\mathcal{D}_1 \times \mathcal{D}_2$ is always a qualification domain. Moreover, if $\mathcal{D}_1$ and $\mathcal{D}_2$ are stable, then the strict cartesian product $\mathcal{D}_1 \otimes \mathcal{D}_2$ is a stable qualification domain.

Proof
Here we reason only for the case of the strict cartesian product since the reasonings needed for the ordinary cartesian product are very similar and even simpler. Assume that $\mathcal{D}_1$ and $\mathcal{D}_2$ are stable qualification domains, and let $\mathcal{D} = \mathcal{D}_1 \otimes \mathcal{D}_2$ be constructed as in Definition 2.11. In order to show that $\mathcal{D}$ is a stable qualification domain, we prove the four items below. The assumption that $\mathcal{D}_1$ and $\mathcal{D}_2$ satisfy all the axioms from Definition 2.10 is used in all the reasonings, often implicitly.

1. The attenuation operator $\circ$ of $\mathcal{D}$ is well defined. Assume $(d_1, d_2), (e_1, e_2) \in D$. According to Definition 2.11, $(d_1, d_2) \circ (e_1, e_2)$ is defined as $(d_1 \circ_1 e_1, d_2 \circ_2 e_2)$. Since $D = D_1 \otimes D_2$ is a strict subset of $D_1 \times D_2$, we must prove that $(d_1 \circ_1 e_1, d_2 \circ_2 e_2) \in D$. We reason by distinction of cases:
   1.1. $(d_1, d_2) = (\mathbf{b}_1, \mathbf{b}_2)$ or $(e_1, e_2) = (\mathbf{b}_1, \mathbf{b}_2)$. In this case, $(d_1 \circ_1 e_1, d_2 \circ_2 e_2) = (\mathbf{b}_1, \mathbf{b}_2) \in D$.
   1.2. $(d_1, d_2) \neq (\mathbf{b}_1, \mathbf{b}_2)$ and $(e_1, e_2) \neq (\mathbf{b}_1, \mathbf{b}_2)$. In this case, $d_1, e_1 \in D_1 \setminus \{\mathbf{b}_1\}$ and $d_2, e_2 \in D_2 \setminus \{\mathbf{b}_2\}$. The assumption that $\mathcal{D}_1$ and $\mathcal{D}_2$ are stable ensures $d_1 \circ_1 e_1 \neq \mathbf{b}_1$ and $d_2 \circ_2 e_2 \neq \mathbf{b}_2$, and therefore $(d_1 \circ_1 e_1, d_2 \circ_2 e_2) \in D$.

2. $\langle D, \preccurlyeq, \mathbf{b}, \mathbf{t} \rangle$ is a lattice with extreme points $\mathbf{b} = (\!|\, \mathbf{b}_1, \mathbf{b}_2 \,|\!) = (\mathbf{b}_1, \mathbf{b}_2)$ and $\mathbf{t} =_{\text{def}} (\!|\, \mathbf{t}_1, \mathbf{t}_2 \,|\!)$ w.r.t. the partial ordering $\preccurlyeq$. By definition, $(d_1, d_2) \preccurlyeq (e_1, e_2) \iff d_1 \preccurlyeq_1 e_1 \wedge d_2 \preccurlyeq_2 e_2$. The fact that $\preccurlyeq$ is a partial ordering with minimum (bottom) element $\mathbf{b}$ is an obvious connsequence. To prove that $\mathbf{t}$ is the maximum (top) element, we reason by case distinction. If D_1 is a singleton set, then $D_1 = \{\mathbf{b}_1\}$, $\mathbf{t}_1 = \mathbf{b}_1$, $D = \{(\mathbf{b}_1, \mathbf{b}_2)\}$, and $(\!|\, \mathbf{t}_1, \mathbf{t}_2 \,|\!) = (\mathbf{b}_1, \mathbf{b}_2)$ is obviously the top element. The case that D_2 is a singleton set is argued similarly. Finally, if neither D_1 nor D_2 are singleton, we have $\mathbf{t}_1 \neq \mathbf{b}_1$, $\mathbf{t}_2 \neq \mathbf{b}_2$, and $\mathbf{t} = (\!|\, \mathbf{t}_1, \mathbf{t}_2 \,|\!) = (\mathbf{t}_1, \mathbf{t}_2)$ is clearly the top element.
   To show that $\langle D, \preccurlyeq, \mathbf{b}, \mathbf{t} \rangle$ is a lattice, we assume two arbitrary elements (d_1, d_2), $(e_1, e_2) \in D$, and we prove:
   2.1. There is a *lub* $(d_1, d_2) \sqcup (e_1, e_2) \in D$. The *lubs* $d_1 \sqcup_1 e_1 \in D_1$ and $d_2 \sqcup_2 e_2 \in D_2$ are known to exist. We claim that $(d_1, d_2) \sqcup (e_1, e_2) = (d_1 \sqcup_1 e_1, d_2 \sqcup_2 e_2)$. Due to the component-wise definition of $\preccurlyeq$, it suffices to show that $(d_1 \sqcup_1 e_1, d_2 \sqcup_2 e_2) \in D$. We prove this by case distinction:
      2.1.1. If $(d_1, d_2) = (\mathbf{b}_1, \mathbf{b}_2)$ then $(d_1 \sqcup_1 e_1, d_2 \sqcup_2 e_2) = (e_1, e_2) \in D$.
      2.1.2. If $(e_1, e_2) = (\mathbf{b}_1, \mathbf{b}_2)$ then $(d_1 \sqcup_1 e_1, d_2 \sqcup_2 e_2) = (d_1, d_2) \in D$.
      2.1.3. If $(d_1, d_2) \neq (\mathbf{b}_1, \mathbf{b}_2)$ and $(e_1, e_2) \neq (\mathbf{b}_1, \mathbf{b}_2)$ then the construction of D ensures that $d_1, e_1 \in D_1 \setminus \{\mathbf{b}_1\}$ and $d_2, e_2 \in D_2 \setminus \{\mathbf{b}_2\}$. This implies $d_1 \sqcup_1 e_1 \neq \mathbf{b}_1$ and $d_2 \sqcup_2 e_2 \neq \mathbf{b}_2$, which guarantees $(d_1 \sqcup_1 e_1, d_2 \sqcup_2 e_2) \in D$.

2.2. There is a *glb* $(d_1, d_2) \sqcap (e_1, e_2) \in D$. The *glbs* $d_1 \sqcap_1 e_1 \in D_1$ and $d_2 \sqcap_2 e_2 \in D_2$ are known to exist. We claim that $(d_1, d_2) \sqcap (e_1, e_2) = (\!| d_1 \sqcap_1 e_1, d_2 \sqcap_2 e_2 |\!)$. We prove the claim by case distinction:

2.2.1. If $d_1 \sqcap_1 e_1 \neq \mathbf{b}_1$ and $d_2 \sqcap_2 e_2 \neq \mathbf{b}_2$, then $(\!| d_1 \sqcap_1 e_1, d_2 \sqcap_2 e_2 |\!)$ is the same as $(d_1 \sqcap_1 e_1, d_2 \sqcap e_2) \in D$, and this pair is the *glb* of (d_1, d_2) and (e_1, e_2) due to the component-wise definition of $\preccurlyeq$.

2.2.2. If $d_1 \sqcap_1 e_1 = \mathbf{b}_1$ or $d_2 \sqcap_2 e_2 = \mathbf{b}_2$, then $(\!| d_1 \sqcap_1 e_1, d_2 \sqcap_2 e_2 |\!) = (\mathbf{b}_1, \mathbf{b}_2)$ is obviously a common lower bound of (d_1, d_2) and (e_1, e_2). In order to conclude that $(\mathbf{b}_1, \mathbf{b}_2)$ is the *glb* of (d_1, d_2) and (e_1, e_2), we show that $(\mathbf{b}_1, \mathbf{b}_2)$ is the only common lower bound of (d_1, d_2) and (e_1, e_2) by the following reasoning: assume an arbitrary $(x, y) \in D$ such that $(x, y) \preccurlyeq (d_1, d_2)$ and $(x, y) \preccurlyeq (e_1, e_2)$. Then $x \preccurlyeq d_1 \sqcap_1 e_1$ and $y \preccurlyeq d_2 \sqcap_2 e_2$. Since $d_1 \sqcap_1 e_1 = \mathbf{b}_1$ or $d_2 \sqcap_2 e_2 = \mathbf{b}_2$, it follows that $x = \mathbf{b}_1$ or $y = \mathbf{b}_2$. By construction of D, it must be the case that $x = \mathbf{b}_1$ *and* $y = \mathbf{b}_2$, because otherwise (x, y) would not belong to D. Therefore $(x, y) = (\mathbf{b}_1, \mathbf{b}_2)$, as desired.

3. $\circ$ satisfies axioms required for attenuation operators in Definition 2.10. By definition of $\circ$ we know

$$(\star)\ (d_1, d_2) \circ (e_1, e_2) = (d_1 \circ_1 e_1, d_2 \circ_2 e_2)$$

which always belongs to D as already proved in item (1) above. All the axioms listed under item (3) of Definition 2.10 except (3)(e) follow easily from the equation $(\star)$ and the corresponding axioms for $\circ_1$ and $\circ_2$. In order to verify axiom (3)(e) for $\circ$, we assume three pairs $(d_1, d_2), (e_1, e_2), (e'_1, e'_2) \in D$. We must prove the equation

$$(\dagger)\ (d_1, d_2) \circ ((e_1, e_2) \sqcap (e'_1, e'_2)) = (d_1, d_2) \circ (e_1, e_2) \sqcap (d_1, d_2) \circ (e'_1, e'_2)\ .$$

We reason by case distinction:

3.1. If $d_1 = \mathbf{b}_1$ and $d_2 = \mathbf{b}_2$ then both sides of $(\dagger)$ are equal to $(\mathbf{b}_1, \mathbf{b}_2)$, as shown by the following calculations:

$$(d_1, d_2) \circ ((e_1, e_2) \sqcap (e'_1, e'_2)) = (\mathbf{b}_1, \mathbf{b}_2) \circ ((e_1, e_2) \sqcap (e'_1, e'_2)) = (\mathbf{b}_1, \mathbf{b}_2)$$

$$(d_1, d_2) \circ (e_1, e_2) \sqcap (d_1, d_2) \circ (e'_1, e'_2) = (\mathbf{b}_1, \mathbf{b}_2) \circ (e_1, e_2) \sqcap (\mathbf{b}_1, \mathbf{b}_2) \circ (e'_1, e'_2) = (\mathbf{b}_1, \mathbf{b}_2) \sqcap (\mathbf{b}_1, \mathbf{b}_2) = (\mathbf{b}_1, \mathbf{b}_2)$$

3.2. If the previous case does not apply, the construction of D ensures that $d_1 \neq \mathbf{b}_1$ and $d_2 \neq \mathbf{b}_2$. We distiguish two subcases:

3.2.1. If $e_1 \sqcap_1 e'_1 = \mathbf{b}_1$ or $e_2 \sqcap_2 e'_2 = \mathbf{b}_2$ we get also $d_1 \circ_1 (e_1 \sqcap_1 e'_1) = \mathbf{b}_1$ or $d_2 \circ_2 (e_2 \sqcap_2 e'_2) = \mathbf{b}_2$, and we can assume the following:

$$(\clubsuit)\quad (\!| e_1 \sqcap_1 e'_1, e_2 \sqcap_2 e'_2 |\!) = (\mathbf{b}_1, \mathbf{b}_2)$$
$$(\spadesuit)\quad (\!| d_1 \circ_1 (e_1 \sqcap_1 e'_1), d_2 \circ_2 (e_2 \sqcap_2 e'_2) |\!) = (\mathbf{b}_1, \mathbf{b}_2)$$

We can now prove that both sides of (†) are equal to $(\mathbf{b}_1, \mathbf{b}_2)$ as follows:

$$(d_1, d_2) \circ ((e_1, e_2) \sqcap (e'_1, e'_2)) = \\ (d_1, d_2) \circ (\!| e_1 \sqcap_1 e'_1, e_2 \sqcap_2 e'_2 |\!) =_{\clubsuit} (d_1, d_2) \circ (\mathbf{b}_1, \mathbf{b}_2) = (\mathbf{b}_1, \mathbf{b}_2)$$

$$(d_1, d_2) \circ (e_1, e_2) \sqcap (d_1, d_2) \circ (e'_1, e'_2) = \\ (d_1 \circ_1 e_1, d_2 \circ_2 e_2) \sqcap (d_1 \circ_1 e'_1, d_2 \circ_2 e'_2) = \\ (\!| d_1 \circ_1 e_1 \sqcap_1 d_1 \circ_1 e'_1, d_2 \circ_2 e_2 \sqcap_2 d_2 \circ_2 e'_2 |\!) = \\ (\!| d_1 \circ_1 (e_1 \sqcap_1 e'_1), d_2 \circ_2 (e_2 \sqcap_2 e'_2) |\!) =_{\spadesuit} (\mathbf{b}_1, \mathbf{b}_2)$$

3.2.2. If $e_1 \sqcap_1 e'_1 \neq \mathbf{b}_1$ and $e_2 \sqcap_2 e'_2 \neq \mathbf{b}_2$ then the stability assumption made for $\mathcal{D}_1$ and $\mathcal{D}_2$ ensures $d_1 \circ_1 (e_1 \sqcap_1 e'_1) \neq \mathbf{b}_1$ and $d_2 \circ_2 (e_2 \sqcap_2 e'_2) \neq \mathbf{b}_2$, and we can assume the following:

($\diamondsuit$) $\quad (\!| e_1 \sqcap_1 e'_1, e_2 \sqcap_2 e'_2 |\!) = (e_1 \sqcap_1 e'_1, e_2 \sqcap_2 e'_2)$
($\heartsuit$) $\quad (\!| d_1 \circ_1 (e_1 \sqcap_1 e'_1), d_2 \circ_2 (e_2 \sqcap_2 e'_2) |\!) = (d_1 \circ_1 (e_1 \sqcap_1 e'_1), d_2 \circ_2 (e_2 \sqcap_2 e'_2))$

Then, (†) is proved by the following calculations:

$$(d_1, d_2) \circ ((e_1, e_2) \sqcap (e'_1, e'_2)) = \\ (d_1, d_2) \circ (\!| e_1 \sqcap_1 e'_1, e_2 \sqcap_2 e'_2 |\!) =_{\diamondsuit} (d_1, d_2) \circ (e_1 \sqcap_1 e'_1, e_2 \sqcap_2 e'_2) = \\ (d_1 \circ_1 (e_1 \sqcap_1 e'_1), d_2 \circ_2 (e_2 \sqcap_2 e'_2))$$

$$(d_1, d_2) \circ (e_1, e_2) \sqcap (d_1, d_2) \circ (e'_1, e'_2) = \\ (d_1 \circ_1 e_1, d_2 \circ_2 e_2) \sqcap (d_1 \circ_1 e'_1, d_2 \circ_2 e'_2) = \\ (\!| d_1 \circ_1 e_1 \sqcap_1 d_1 \circ_1 e'_1, d_2 \circ_2 e_2 \sqcap_2 d_2 \circ_2 e'_2 |\!) = \\ (\!| d_1 \circ_1 (e_1 \sqcap_1 e'_1), d_2 \circ_2 (e_2 \sqcap_2 e'_2) |\!) =_{\heartsuit} \\ (d_1 \circ_1 (e_1 \sqcap_1 e'_1), d_2 \circ_2 (e_2 \sqcap_2 e'_2))$$

4. $\mathcal{D}_1 \otimes \mathcal{D}_2$ is stable. To prove this let us assume $(d_1, d_2), (e_1, e_2) \in D_1 \otimes D_2 \setminus \{(\mathbf{b}_1, \mathbf{b}_2)\}$. Then $d_1, e_1 \in D_1 \setminus \{\mathbf{b}_1\}$ and $d_2, e_2 \in D_2 \setminus \{\mathbf{b}_2\}$. Since $\mathcal{D}_1$ and $\mathcal{D}_2$ are stable qualification domains, we can infer that $d_1 \circ_1 e_1 \neq \mathbf{b}_1$ and $d_2 \circ_2 e_2 \neq \mathbf{b}_2$, which implies $(d_1, d_2) \circ (e_1, e_2) = (d_1 \circ_1 e_1, d_2 \circ_2 e_2) \neq (\mathbf{b}_1, \mathbf{b}_2)$. □

2.2.5 Encoding Qualification Domains into Constraint Domains

In this subsection we investigate a technical relationship between qualification domains and constraint domains which will play a key role in the rest of the report.

Definition 2.12 (*Expressing $\mathcal{D}$ in $\mathcal{C}$*)
A qualification domain $\mathcal{D}$ with carrier set $D_{\mathcal{D}}$ is expressible in a constraint domain $\mathcal{C}$ with carrier set $C_{\mathcal{C}}$ if there is an injective mapping $\imath : D_{\mathcal{D}} \setminus \{\mathbf{b}\} \to C_{\mathcal{C}}$ embedding $D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ into $C_{\mathcal{C}}$, and the two following requirements are satisfied:

1. There is a $\mathcal{C}$-constraint $\mathsf{qVal}(X)$ such that $\mathrm{Sol}_{\mathcal{C}}(\mathsf{qVal}(X))$ is the set of all $\eta \in \mathrm{Val}_{\mathcal{C}}$ such that $\eta(X)$ belongs to the range of $\imath$.
2. There is a $\mathcal{C}$-constraint $\mathsf{qBound}(X, Y, Z)$ encoding "$x \trianglelefteqslant y \circ z$" in the following sense: any $\eta \in \mathrm{Val}_{\mathcal{C}}$ satisfying $\eta(X) = \imath(x)$, $\eta(Y) = \imath(y)$ and $\eta(Z) = \imath(z)$ for some $x, y, z \in D \setminus \{\mathbf{b}\}$ verifies $\eta \in \mathrm{Sol}_{\mathcal{C}}(\mathsf{qBound}(X, Y, Z)) \Longleftrightarrow x \trianglelefteqslant y \circ z$.

Moreover, if $\mathsf{qVal}(X)$ and $\mathsf{qBound}(X, Y, Z)$ can be chosen as existential constraints, we say that $\mathcal{D}$ is *existentially expressable* in $\mathcal{C}$. □

In the sequel, $\mathcal{C}$-constraints built as instances of $\mathsf{qVal}(X)$ and $\mathsf{qBound}(X, Y, Z)$ are called *qualification constraints*, and Ω is used as notation for sets of qualification constraints. The following result ensures that several qualification domains built with the techniques presented in Subsection 2.2 are existentially expressible in $\mathcal{H}$, $\mathcal{R}$ or $\mathcal{FD}$, according to the case.

Proposition 2.2 (*Expressing Qualification Domains in Constraint Domains*)

1. The domain $\mathcal{B}$ is existentially expressible in any given constraint domain $\mathcal{C}$.
2. The domains $\mathcal{U}, \mathcal{U}'$, $\mathcal{W}$ and $\mathcal{W}'$ are existentially expressible in $\mathcal{R}$ (or any other constraint domain that supports the expressivity of $\mathcal{R}$).
3. The domains $\mathcal{W}_d$ and $\mathcal{W}'_d$ are existentially expressible in $\mathcal{FD}$ (or any other constraint domain that supports the expressivity of $\mathcal{FD}$).
4. Assume that the two qualification domains $\mathcal{D}_1$ and $\mathcal{D}_2$ are stable and (existentially) expressible in a constraint domain $\mathcal{C}$. Then, $\mathcal{D}_1 \otimes \mathcal{D}_2$ is also (existentially) expressible in $\mathcal{C}$.

Proof

1. Straightforward, due to the fact that $D_{\mathcal{B}} \setminus \{\mathbf{b}\}$ is the singleton set $\{\mathbf{t}\} = \{true\}$.
2. We prove that $\mathcal{U}$ can be existentially expressed in $\mathcal{R}$ as follows: $D_{\mathcal{U}} \setminus \{\mathbf{b}\} = D_{\mathcal{U}} \setminus \{0\} = (0, 1] \subseteq \mathbb{R} \subseteq C_{\mathcal{R}}$; therefore $\imath$ can be taken as the identity embedding mapping from $(0, 1]$ into $\mathbb{R}$. Moreover, $\mathsf{qVal}(X)$ can be built as the existential $\mathcal{R}$-constraint $cp_<(0, X) \wedge cp_\leq(X, 1)$ and $\mathsf{qBound}(X, Y, Z)$ can be built as the existential $\mathcal{R}$-constraint $\exists X_1(op_\times(Y, Z, X_1) \wedge cp_\leq(X, X_1))$. By very similar reasonings it is easy to check that $\mathcal{U}'$, $\mathcal{W}$ and $\mathcal{W}'$ can also be existentially expressed in $\mathcal{R}$. Note that in the cases of $\mathcal{W}$ and $\mathcal{W}'$ there is no reasonable way to define $\imath(\infty)$. This is the reason why the domain of $\imath$ is required to be $D \setminus \{\mathbf{b}\}$ in Definition 2.12.
3. Note that $D_{\mathcal{W}_d} \setminus \{\mathbf{b}\} = D_{\mathcal{W}_d} \setminus \{\infty\} = \mathbb{N}$. Moreover, $\preccurlyeq$ is $\geq$ and $\circ$ is $+$ in $\mathcal{W}_d$. Therefore, $\mathcal{W}_d$ can be expressed in $\mathcal{FD}$ by taking $\imath$ as the identity embedding mapping, building $\mathsf{qVal}(X)$ as an existential $\mathcal{FD}$ constraint that requires the value of X to be an integer $x \geq 0$, and building $\mathsf{qBound}(X, Y, Z)$ as an existential $\mathcal{FD}$ constraint that requires the values of X, Y and Z to be integers x, y and z such that $x \geq y + z$. A similar reasoning proves that $\mathcal{W}'_d$ is existentially expressible in $\mathcal{FD}$ also.
4. For $j = 1, 2$ assume the existence of injective embedding mappings $\imath_j$ and $\mathcal{C}$-constraints $\mathsf{qVal}_j(X)$, $\mathsf{qBound}_j(X, Y, Z)$ that can be used to (existentially) express $\mathcal{D}_j$ in $\mathcal{C}$. Due to Theorem 2.1 we know that $\mathcal{D}_1 \otimes \mathcal{D}_2$ is a stable qualification domain. Moreover, because of the construction of $\mathcal{D} = \mathcal{D}_1 \otimes \mathcal{D}_2$ given in Definition 2.11, we know that $D \setminus \{\mathbf{b}\} = (D_1 \setminus \{\mathbf{b}_1\}) \times (D_2 \setminus \{\mathbf{b}_2\})$. We also know that $\preccurlyeq$ is defined component-wise from $\preccurlyeq_1$, and $\preccurlyeq_2$, and analogously for $\circ$. Therefore, $\mathcal{D}$ can be (existentially) expressed in $\mathcal{C}$ by taking:

   - $\imath$ defined by $\imath(d_1, d_2) =_{\text{def}} (\imath_1(d_1), \imath_2(d_2))$.
   - $\mathsf{qVal}(X)$ built as the prenex form of the constraint

   $$\exists X_1 \exists X_2 (X == (X_1, X_2) \wedge \mathsf{qVal}_1(X_1) \wedge \mathsf{qVal}_2(X_2))$$

   which is existential if $\mathsf{qVal}_1(X_1)$ and $\mathsf{qVal}_2(X_2)$ are both existential.

- $\mathsf{qBound}(X, Y, Z)$ built as the prenex form of the constraint

$$\exists X_1 \exists X_2 \exists Y_1 \exists Y_2 \exists Z_1 \exists Z_2 (X == (X_1, X_2) \wedge Y == (Y_1, Y_2) \wedge Z == (Z_1, Z_2) \wedge \mathsf{qBound}_1(X_1, Y_1, Z_1) \wedge \mathsf{qBound}_2(X_2, Y_2, Z_2))$$

  which is existential if $\mathsf{qBound}_1(X_1, Y_1, Z_1)$ and $\mathsf{qBound}_2(X_2, Y_2, Z_2)$ are both existential.

Note that this reasoning does not work for the non-strict cartesian product $\mathcal{D} = \mathcal{D}_1 \times \mathcal{D}_2$, because in this case $D \setminus \{\mathbf{b}\} = (D_1 \times D_2) \setminus \{(\mathbf{b}_1, \mathbf{b}_2)\}$ includes some pairs (d_1, d_2) such that either $d_1 = \mathbf{b}_1$ or $d_2 = \mathbf{b}_2$ (but not both), and the given mappings $\imath_1$, $\imath_2$ cannot be used to embed such pairs into $C_{\mathcal{C}}$. □

2.3 Similarity and Proximity Relations

Similarity relations over a given set S have been defined in (Zadeh 1971; Sessa 2002) and related literature as mappings $\mathcal{S} : S \times S \to [0, 1]$ that satisfy reflexivity, symmetry and transitivity axioms analogous to those required for classical equivalence relations. A more general notion called *proximity relation* has been defined in (Dubois and Prade 1980) by omitting the transitivity axiom. Each value $\mathcal{S}(x, y)$ computed by a similarity (resp. proximity) relation $\mathcal{S}$ is called the *similarity degree* (resp. *proximity degree*) between x and y. In our previous paper (Caballero et al. 2008), we proposed to generalize similarity relations by allowing elements of an arbitrary qualification domain $\mathcal{D}$ to serve as proximity degrees. The definition below further generalizes this approach by considering proximity relations.

Definition 2.13 (*Proximity and similarity relations*)
Let a qualification domain $\mathcal{D}$ with carrier set D and a set S be given.

1. A $\mathcal{D}$-*valued relation* over S is any mapping $\mathcal{S} : S \times S \to D$.
2. A $\mathcal{D}$-valued relation $\mathcal{S}$ over S is called
   (a) *Reflexive* iff $\forall x \in S : \mathcal{S}(x, x) = \mathbf{t}$.
   (b) *Symmetrical* iff $\forall x, y \in S : \mathcal{S}(x, y) = \mathcal{S}(y, x)$.
   (c) *Transitive* iff $\forall x, y, z \in S : \mathcal{S}(x, z) \trianglerighteq \mathcal{S}(x, y) \sqcap \mathcal{S}(y, z)$.
3. $\mathcal{S}$ is called a $\mathcal{D}$-*valued proximity relation* iff $\mathcal{S}$ is reflexive and symmetrical.
4. If $\mathcal{S}$ is also transitive, then it is called a $\mathcal{D}$-*valued similarity relation.*
5. $\mathcal{S}$ is called *finitary* iff there are only finitely many choices of elements $x, y \in S$ such that $x \neq y$ and $\mathcal{S}(x, y) \neq \mathbf{b}$. From a practical viewpoint, this is a very natural requirement. □

Obviously, $\mathcal{D}$-valued similarity relations are a particular case of $\mathcal{D}$-valued proximity relations. Moreover, when $\mathcal{D}$ is chosen as the qualification domain $\mathcal{U}$, the previous definition provides proximity and similarity relations in the sense of (Zadeh 1971; Dubois and Prade 1980). In this case, a proximity degree $\mathcal{S}(x, y) = d \in [0, 1]$ can be naturally interpreted as a *certainty degree* for the assertion that x and y are interchangeable. On the other hand, if $\mathcal{S}$ is $\mathcal{W}$-valued, then $\mathcal{S}(x, y) = d \in [0, \infty]$ can be interpreted as a *cost* to be paid for y to play the role of x. More generally, the

proximity degrees computed by a $\mathcal{D}$-valued proximity relation may have different interpretations according to the intended role of $\mathcal{D}$-elements as qualification values.

In contrast to previous works such as (Sessa 2002; Caballero et al. 2008), in the rest of this report we will work with $\mathcal{D}$-valued proximity rather than similarity relations. Formally, this leads to more general results. Moreover, as already noted by (Shenoi and Melton 1999) and other authors, the transitivity property required for similarity relations may be counterintuitive in some cases. For instance, assume nullary constructors `colt`, `cold` and `gold` intended to represent words composed of four letters. Then, measuring the proximity between such words might reasonably lead to a $\mathcal{U}$-valued proximity relation $\mathcal{S}$ such that $\mathcal{S}(\texttt{colt},\texttt{cold}) = 0.9$, $\mathcal{S}(\texttt{cold},\texttt{gold}) = 0.9$ and $\mathcal{S}(\texttt{colt},\texttt{gold}) = 0.4$. On the other hand, insisting on $\mathcal{S}$ to be transitive would enforce the unreasonable condition $\mathcal{S}(\texttt{colt},\texttt{gold}) \geq 0.9$. Therefore, a similarity relation would be not appropriate in this case.

The special mapping $\mathcal{S}_{\mathrm{id}} : S \times S \to D$ defined as $\mathcal{S}_{\mathrm{id}}(x,x) = \mathbf{t}$ for all $x \in S$ and $\mathcal{S}_{\mathrm{id}}(x,y) = \mathbf{b}$ for all $x, y \in S$, $x \neq y$ is trivially a $\mathcal{D}$-valued similarity (and therefore, also proximity) relation called the *identity*.

2.3.1 Admissible triples and proximity relations

From now on, we will focus on proximity relations that are related to constraint domains in the following sense:

Definition 2.14 (*Admissible triples*)
$\langle \mathcal{S}, \mathcal{D}, \mathcal{C} \rangle$ is called an *admissible triple* iff the following requirements are fulfilled:

1. $\mathcal{C}$ is a constraint domain with signature $\Sigma = \langle DC, DP, PP \rangle$ and set of basic values $B_{\mathcal{C}}$, and $\mathcal{D}$ is a qualification domain expressible in $\mathcal{C}$ in the sense of Definition 2.12.
2. $\mathcal{S}$ is a $\mathcal{D}$-valued proximity relation over $S = \mathcal{V}ar \uplus B_{\mathcal{C}} \uplus DC \uplus DP \uplus PP$.
3. $\mathcal{S}$ restricted to $\mathcal{V}ar$ behaves as the identity, i.e. $\mathcal{S}(X,X) = \mathbf{t}$ for all $X \in \mathcal{V}ar$ and $\mathcal{S}(X,Y) = \mathbf{b}$ for all $X, Y \in \mathcal{V}ar$ such that $X \neq Y$.
4. For any $x, y \in S$, $\mathcal{S}(x,y) \neq \mathbf{b}$ can happen only if some of the following cases holds:
   (a) $x = y$ are identical.
   (b) $x, y \in B_{\mathcal{C}}$ are basic values.
   (c) $x, y \in DC$ are data constructor symbols with the same arity.
   (d) $x, y \in DP$ are defined predicate symbols with the same arity.

   In particular, $\mathcal{S}(p,p') \neq \mathbf{b}$ cannot happen if $p, p' \in PP$ are syntactically different primitive predicate symbols. □

In the rest of the report, our notions and results are valid for any choice of an admissible triple $\langle \mathcal{S}, \mathcal{D}, \mathcal{C} \rangle$. Proposition 2.2 provides useful information for building admissible triples. For any given admissible triple, $\mathcal{S}$ can be naturally extended to act over terms and atoms over $\mathcal{C}$. The extension, also noted $\mathcal{S}$, works as specified in the recursive definition below. An analogous definition for the case of $\mathcal{U}$-valued similarity relations can be found in (Sessa 2002).

Definition 2.15 ($\mathcal{S}$ *acting over terms and atoms*)
For any given admissible triple, $\mathcal{S}$ is extended to work over $\mathcal{C}$-terms and $\mathcal{C}$-atoms as follows:

1. For any $t \in \text{Term}(\Sigma, B, \mathcal{V}ar)$:
   $\mathcal{S}(t, t) = \mathbf{t}$.
2. For $X \in \mathcal{V}ar$ and for any term t different from X:
   $\mathcal{S}(X, t) = \mathcal{S}(t, X) = \mathbf{b}$.
3. For $c, c' \in DC$ with different arities n, m:
   $\mathcal{S}(c(\overline{t}_n), c'(\overline{t'}_m)) = \mathbf{b}$.
4. For $c, c' \in DC$ with the same arity n:
   $\mathcal{S}(c(\overline{t}_n), c'(\overline{t'}_n)) = \mathcal{S}(c, c') \sqcap \mathcal{S}(t_1, t'_1) \sqcap \ldots \sqcap \mathcal{S}(t_n, t'_n)$.
5. For $r, r' \in DP \cup PP$ with different arities n, m:
   $\mathcal{S}(r(\overline{t}_n), r'(\overline{t'}_m)) = \mathbf{b}$.
6. For $r, r' \in DP \cup PP$ with the same arity n:
   $\mathcal{S}(r(\overline{t}_n), r'(\overline{t'}_n) = \mathcal{S}(r, r') \sqcap \mathcal{S}(t_1, t'_1) \sqcap \ldots \sqcap \mathcal{S}(t_n, t'_n)$. □

Given two terms t, t' and some fixed qualification value $\lambda \in D \setminus \{\mathbf{b}\}$ we will use the notation $t \approx_\lambda t'$ (read as t and t' are $\mathcal{S}$*-close at level* λ) as an abbreviation of $\lambda \trianglelefteqslant \mathcal{S}(t, t')$. For the sake of simplicity, $\mathcal{S}$ is not made explicit in the $\approx_\lambda$ notation. The following lemma provides a natural characterization of $\approx_\lambda$. A similar result was given in (Sessa 2002) for the case of case of $\mathcal{U}$-valued similarity relations.

Lemma 2.5 (*Proximity Lemma*)

1. $\approx_\lambda$ is a reflexive and symmetric equivalence relation over terms, which is also transitive (and hence an equivalence relation) in the case that $\mathcal{S}$ is a similarity relation.
2. For any given terms t and t' the following two statements are equivalent:
   (a) $t \approx_\lambda t'$.
   (b) $\text{pos}(t) = \text{pos}(t')$, and for each $p \in \text{pos}(t) \cap \text{pos}(t')$ some of the cases below holds:
      i $t \circ p = t' \circ p = X$ for some $X \in \mathcal{V}ar$.
      ii $t \circ p = s \circ p = u$ for some $u \in B_{\mathcal{C}}$.
      iii $t \circ p = c$ and $t' \circ p = c'$ for some $n \in \mathbb{N}$ and some $c, c' \in DC^n$ such that $\lambda \trianglelefteqslant \mathcal{S}(c, c')$.
3. Any given terms t and t' such that $t \approx_\lambda t'$ are *quasi-identical* in the following sense: $\text{pos}(t) = \text{pos}(t')$, and for each $p \in \text{pos}(t) = \text{pos}(t')$ either $t \circ p = t' \circ p$ or else $t \circ p$ and $t' \circ p$ are two data constructors of the same arity.
4. $\approx_\lambda$ boils down to the syntactic equality relation '=' when $\mathcal{S}$ is the identity proximity relation $\mathcal{S}_{\text{id}}$.

Proof
We give a separate reasoning for each item.

1. Note that the reflexivity and symmetry of $\approx_\lambda$ are a trivial consequence of the reflexivity and symmetry of $\mathcal{S}$, as formulated in Definition 2.13. In the case that $\mathcal{S}$ is a similarity relation, transitivity of $\approx_\lambda$ follows from transitivity of $\mathcal{S}$ and the obvious fact that $\lambda \sqcap \lambda = \lambda$.
2. The claimed equivalence between conditions 2(a) and 2(b) can be proved reasoning by induction on $\|t\| + \|t'\|$.
3. This item is an obvious consequence of the previous one.
4. Assume $\mathcal{S} = \mathcal{S}_{\mathrm{id}}$. Then, as a trivial consequence of Definition 2.15, the value of $\mathcal{S}(t, t')$ is $\mathbf{t}$ if $t = t'$ and $\mathbf{b}$ otherwise. Since $\lambda \neq \mathbf{b}$, it follows that $t \approx_\lambda t'$ iff $t = t'$, as desired. □

The following result shows that $\approx_\lambda$ is compatible with the term extension operation in a natural way:

Lemma 2.6 (*Proximity Preservation Lemma*)
Assume terms t, t' and $\lambda \in D \setminus \{\mathbf{b}\}$ such that $t \approx_\lambda t'$. Then $(t \ll s) \approx_\lambda (t' \ll s)$ holds also for any term s.

Proof
Due to the assumption, $t \approx_\lambda t'$ are quasi-identical and satisfy condition 2(b) as stated in the Proximity Lemma 2.5. From this fact and Definition 2.9 it is quite clear that the same condition 2(b) holds also for $t \ll s$, $t' \ll s$ and λ. Therefore, the Proximity Lemma allows to conclude $t \approx_\lambda t'$ as desired. □

2.3.2 Term proximity w.r.t. a given constraint set

Reasoning with equations between $\mathcal{C}$-terms will require to infer information both from $\mathcal{S}$ and for some fixed constraint set $\Pi \subseteq \mathrm{Con}_{\mathcal{C}}$. This leads to a generalization of $\approx_\lambda$ formally defined as follows:

Definition 2.16 (*Term proximity w.r.t. a given constraint set*)
Let $\langle \mathcal{S}, \mathcal{D}, \mathcal{C} \rangle$ be any admissible triple. Assume $\lambda \in D \setminus \{\mathbf{b}\}$ and $\Pi \subseteq \mathrm{Con}_{\mathcal{C}}$. We will say that t and s are *$\mathcal{S}$-close at level λ w.r.t. Π* (in symbols, $t \approx_{\lambda,\Pi} s$) iff there are two terms $\hat{t}$, $\hat{s}$ such that $t \approx_\Pi \hat{t}$, $s \approx_\Pi \hat{s}$ and $\hat{t} \approx_\lambda \hat{s}$. For the sake of simplicity neither $\mathcal{S}$ nor $\mathcal{C}$ are made explicit in the notation. □

As illustration, let us present an example using the constraint domain $\mathcal{R}$ and the qualification domain $\mathcal{U}$:

Example 2.2 (*Term proximity w.r.t. $\mathcal{R}$ constraints*)
Consider $\Pi = \{op_+(A, A, X), op_\times(2.0, A, Y), Z == c(X, Y)\} \subseteq \mathrm{Con}_{\mathcal{R}}$. Note that this choice of Π ensures $X \approx_\Pi Y$. Assume $c, c', c'' \in DC^2$ and an $\mathcal{U}$-valued proximity relation $\mathcal{S}$ such that $\mathcal{S}(c', c) = \mathcal{S}(c, c'') = 0.8$ and $\mathcal{S}(c', c'') = 0.6$. Then:

1. $c(Y, X) \approx_\Pi Z$ holds, but $c'(Y, X) \approx_\Pi Z$ is false.
2. $c'(Y, X) \approx_{0.7,\Pi} Z$ holds, because $c'(Y, X) \approx_\Pi c'(X, X)$, $Z \approx_\Pi c(X, X)$ and $c'(X, X) \approx_{0.7} c(X, X)$.
3. $Z \approx_{0.7,\Pi} c''(X, Y)$ is also true, for similar reasons.

4. $c'(Y,X) \approx_{0.7,\Pi} c''(X,Y)$ is false, because there is no possible choice of terms $\hat{t}$ and $\hat{s}$ such that $c'(Y,X) \approx_\Pi \hat{t}$, $c''(X,Y) \approx_\Pi \hat{s}$ and $\hat{t} \approx_{0.7} \hat{s}$. □

The next result states some basic properties of relations $\approx_{\lambda,\Pi}$.

Lemma 2.7 (Π-*Proximity Lemma*)

1. $\approx_{\lambda,\Pi}$ is invariant w.r.t. $\approx_\Pi$ in the following sense: $t \approx_{\lambda,\Pi} s$ implies $t' \approx_{\lambda,\Pi} s'$ for all terms t', s' such that $t' \approx_\Pi t$ and $s' \approx_\Pi s$.
2. $\approx_{\lambda,\Pi}$ is a reflexive and symmetric relation over terms, which is also transitive (and hence an equivalence relation) in the case that $\mathcal{S}$ is a similarity relation.
3. For any given terms t and t' the following two statements are equivalent:
   (a) $t \approx_{\lambda,\Pi} t'$.
   (b) For any common position $p \in \mathrm{pos}(t) \cap \mathrm{pos}(t')$ some of the cases below holds:
      i $t \circ p$ or $t' \circ p$ is a variable, and moreover $t|_p \approx_{\lambda,\Pi} t'|_p$.
      ii $t \circ p = s \circ p = u$ for some $u \in B_\mathcal{C}$.
      iii $t \circ p = c$ and $t' \circ p = c'$ for some $n \in \mathbb{N}$ and some $c, c' \in DC^n$ such that $\lambda \trianglelefteq \mathcal{S}(c, c')$.
4. $\approx_{\lambda,\Pi}$ boils down to $\approx_\lambda$ when Π is the empty set, and $\approx_{\lambda,\Pi}$ boils down to $\approx_\Pi$ when $\mathcal{S}$ is the identity proximity relation $\mathcal{S}_{\mathrm{id}}$.

Proof

We give a separate reasoning for each item. Definition 2.16 and Lemmata 2.2 and 2.5 are implicitely used at some points.

1. By definition, $t \approx_{\lambda,\Pi} s$ means the existence of terms $\hat{t}, \hat{s}$ such that $t \approx_\Pi \hat{t}$, $s \approx_\Pi \hat{s}$ and $\hat{t} \approx_\lambda \hat{s}$. In case that $t' \approx_\Pi t$ and $s' \approx_\Pi s$, the same terms $\hat{t}, \hat{s}$ verify $t' \approx_\Pi \hat{t}$, $s' \approx_\Pi \hat{s}$ (since $\approx_\Pi$ is an equivalence relation) and $\hat{t} \approx_\lambda \hat{s}$. Therefore $t' \approx_{\lambda,\Pi} s'$.
2. Let us consider the three properties in turn:
   Reflexivity: $t \approx_{\lambda,\Pi} t$ holds because $\hat{t} = t$ trivially verifies $t \approx_\Pi \hat{t}$ and $\hat{t} \approx_\lambda \hat{t}$.
   Symmetry: Assume $t \approx_{\lambda,\Pi} s$. Then there are terms $\hat{t}, \hat{s}$ such that $t \approx_\Pi \hat{t}$, $s \approx_\Pi \hat{s}$ and $\hat{t} \approx_\lambda \hat{s}$. Due to the symmetry of $\approx_\lambda$ we get $\hat{s} \approx_\lambda \hat{t}$ and hence $s \approx_{\lambda,\Pi} t$.
   Transitivity: Example 2.2 above shows that $\approx_{\lambda,\Pi}$ is not transitive in general. Here we prove transitivity of $\approx_{\lambda,\Pi}$ under the assumption that $\mathcal{S}$ is a similarity relation fulfilling the transitive property stated in Definition 2.13.
   Assume terms t_1, t_2 and t_3 such that $t_1 \approx_{\lambda,\Pi} t_2$ and $t_2 \approx_{\lambda,\Pi} t_3$. Then there are terms t_1', t_2', t_2'' and t_3'' such that

$$(a)\ t_1 \approx_\Pi t_1',\ t_2 \approx_\Pi t_2',\ t_1' \approx_\lambda t_2' \quad \text{and} \quad (b)\ t_2 \approx_\Pi t_2'',\ t_3 \approx_\Pi t_3'',\ t_2'' \approx_\lambda t_3'' \ .$$

   Without loss of generality, t_1', t_2', t_2'' and t_3'' can be assumed to be Π-canonical terms. If they were not, it would suffice to to replace each of them by its Π-canonical form, built as explained in Definition 2.8. This replacement would preserve properties (a) and (b) thanks to the Canonicity Lemma 2.3.
   We claim that there are three terms $\hat{t_1}$, $\hat{t_2}$, and $\hat{t_3}$ such that

$$(c)\ t_1 \approx_\Pi \hat{t_1},\ t_2 \approx_\Pi \hat{t_2},\ t_3 \approx_\Pi \hat{t_3} \quad \text{and} \quad (d)\ \hat{t_1} \approx_\lambda \hat{t_2},\ \hat{t_2} \approx_\lambda \hat{t_3} \ .$$

Conditions (c) and (d) imply $t_1 \approx_{\lambda,\Pi} t_3$ due to Definition 2.16 and the transivity property of $\approx_\lambda$, which is ensured by the transitivity of $\mathcal{S}$ and the Proximity Lemma 2.5. In the rest of this item we prove the claim by assuming (a) and (b) and showing how to build $\hat{t_1}$, $\hat{t_2}$, and $\hat{t_3}$ fulfilling (c) and (d).
Note that the assumption $t'_1 \approx_\lambda t'_2$ implies that t'_1 and t'_2 are quasi-identical terms, due Proximity Lemma 2.5(3). Analogously, terms t''_2 and t''_3 must be also quasi-identical due to the assumption $t''_2 \approx_\lambda t''_3$, and the target condition (d) requires that $\hat{t_1}$, $\hat{t_2}$, $\hat{t_3}$ are constructed as quasi-identical terms. Since our assumptions do not guarantee quasi-identity of terms t'_2 and t''_2, we resort to the term extension construction from Definition 2.9 for building the terms $\hat{t_i}$. More precisely, we build:

$$\hat{t_1} =_{\text{def}} (t'_1 \ll t''_2); \quad \hat{t_2} =_{\text{def}} (t'_2 \ll t''_2) = (t''_2 \ll t'_2); \quad \text{and } \hat{t_3} =_{\text{def}} (t''_3 \ll t'_2)$$

where the identity $(t'_2 \ll t''_2) = (t''_2 \ll t'_2)$ is a consequence of the Symmetrical Extension Property from Lemma 2.4, which can be applied because t'_2 and t''_2 are Π-canonical and assumptions (a), (b) imply $t'_2 \sim_\Pi t''_2$. We argue that conditions (c) and (d) are satisfied as follows:
— Condition (c), $t_1 \approx_\Pi \hat{t_1}$: By assumptions (a), (b) we know $t_1 \approx_\Pi t'_1$ and $t'_2 \approx_\Pi t''_2$. It suffices to prove $t'_1 \approx_\Pi \hat{t_1}$. For each $p \in \text{pos}(t'_1)$ with $t'_1|_p = X \in \mathcal{V}ar$ we have $t'_2|_p = X$ because t'_1 and t'_2 are quasi-identical. Moreover, $t'_2 \approx_\Pi t''_2$ implies that $p \in \text{pos}(t''_2)$ and $X \sim_\Pi t''_2|_p$, due to the Π-Equivalence Lemma 2.2. In these conditions, $t'_1 \approx_\Pi \hat{t_1}$ follows from $\hat{t_1} = (t'_1 \ll t''_2)$ and the Equivalent Extension Property from Lemma 2.4.
— Condition (c), $t_3 \approx_\Pi \hat{t_3}$: The proof for this is analogous to the previous one. Since $t_3 \approx_\Pi t''_3$ and $\hat{t_3} = (t''_3 \ll t'_2)$ it suffices to prove $t''_3 \approx_\Pi (t''_3 \ll t'_2)$, which can be done with the help of the Equivalent Extension Property.
— Condition (c), $t_2 \approx_\Pi \hat{t_2}$: By assumptions (a), (b) we know $t_2 \approx_\Pi t'_2$ and $t'_2 \approx_\Pi t''_2$. It suffices to prove $t'_2 \approx_\Pi \hat{t_2}$. For each $p \in \text{pos}(t'_2)$ with $t'_2|_p = X \in \mathcal{V}ar$ we have $p \in \text{pos}(t''_2)$ and $X \sim_\Pi t''_2|_p$, due to $t'_2 \approx_\Pi t''_2$ and the Π-Equivalence Lemma 2.2. In these conditions, $t'_2 \approx_\Pi \hat{t_2}$ follows from $\hat{t_2} = (t'_2 \ll t''_2)$ and the Equivalent Extension Property.
— Condition (d), $\hat{t_1} \approx_\lambda \hat{t_2}$: By assumption (a) we have $t'_1 \approx_\lambda t'_2$. By the Proximity Preservation Lemma 2.6 this implies $(t'_1 \ll t''_2) \approx_\lambda (t'_2 \ll t''_2)$. By construction of the terms $\hat{t_i}$, this is the same as $\hat{t_1} \approx_\lambda \hat{t_2}$.
— Condition (d), $\hat{t_2} \approx_\lambda \hat{t_3}$: The proof for this is analogous to the previous one. Assumption (b) provides $t''_2 \approx_\lambda t''_3$. Then, the Proximity Preservation Lemma guarantees $(t''_2 \ll t'_2) \approx_\lambda (t''_3 \ll t'_2)$, which is the same as $\hat{t_2} \approx_\lambda \hat{t_3}$ by construction of the terms $\hat{t_i}$ (this time viewing $\hat{t_2}$ as $(t''_2 \ll t'_2)$ rather than $(t'_2 \ll t''_2)$ as in the previous argumentation).

3. The claimed equivalence between conditions 3(a) and 3(b) can proved reasoning by induction on $\|t\| + \|t'\|$.
4. According to Definition 2.16, $t \approx_{\lambda,\Pi} s$ is true iff $(\star)$ holds, where:

$$(\star) \text{ there are terms } \hat{t}, \hat{s} \text{ such that } t \approx_\Pi \hat{t},\ s \approx_\Pi \hat{s} \text{ and } \hat{t} \approx_\lambda \hat{s}.$$

Let us argue for the two cases $\Pi = \emptyset$ and $\mathcal{S} = \mathcal{S}_{\text{id}}$ separately:

- Assume that $\Pi = \emptyset$. Then, due to Π-Equivalence Lemma 2.2(3), $(\star)$ can be

rewritten as

$$\text{there are terms } \hat{t}, \hat{s} \text{ such that } t = \hat{t},\ s = \hat{s} \text{ and } \hat{t} \approx_\lambda \hat{s}$$

which is equivalent to $t \approx_\lambda s$.

- Assume now that $\mathcal{S} = \mathcal{S}_{\mathrm{id}}$. Then, due to Proximity Lemma 2.5(4), $(\star)$ can be rewritten as

$$\text{there are terms } \hat{t}, \hat{s} \text{ such that } t \approx_\Pi \hat{t},\ s \approx_\Pi \hat{s} \text{ and } \hat{t} = \hat{s}$$

which is equivalent to $t \approx_\Pi s$. □

The following technical lemma will be needed later on. Although it is closely related to Lemma 2.1(2), it needs a separate proof because statements of the form $t \approx_{\lambda,\Pi} s$ are not $\mathcal{C}$-constraints.

Lemma 2.8 (*Substitution Lemma for* $\approx_{\lambda,\Pi}$)
Let $\langle \mathcal{S}, \mathcal{D}, \mathcal{C} \rangle$ be any admissible triple. Assume $\lambda \in D \setminus \{\mathbf{b}\}$, $\Pi \subseteq \mathrm{Con}_{\mathcal{C}}$, and two terms t, s such that $t \approx_{\lambda,\Pi} s$. Then $t\sigma \approx_{\lambda,\Pi\sigma} s\sigma$ holds for every $\mathcal{C}$-substitution σ.

Proof
Because of the assumptions and Definition 2.16, there are terms $\hat{t}, \hat{s}$ such that $t \approx_\Pi \hat{t}$ (i.e. $\Pi \models_{\mathcal{C}} t == \hat{t}$), $s \approx_\Pi \hat{s}$ (i.e. $\Pi \models_{\mathcal{C}} s == \hat{s}$) and $\hat{t} \approx_\lambda \hat{s}$. Consider now any substitution σ. Due to Lemma 2.1(2), we get $\Pi\sigma \models_{\mathcal{C}} t\sigma == \hat{t}\sigma$ (i.e. $t\sigma \approx_{\Pi\sigma} \hat{t}\sigma$) and $\Pi\sigma \models_{\mathcal{C}} s\sigma == \hat{s}\sigma$ (i.e. $s\sigma \approx_{\Pi\sigma} \hat{s}\sigma$). Moreover, $\hat{t}\sigma \approx_\lambda \hat{s}\sigma$ is an easy consequence of $\hat{t} \approx_\lambda \hat{s}$ and Proximity Lemma 2.5(2). Then, Definition 2.16 allows to conclude $t\sigma \approx_{\lambda,\Pi\sigma} s\sigma$ simply by taking $\hat{t}\sigma$ as $\hat{t\sigma}$ and $\hat{s}\sigma$ as $\hat{s\sigma}$. □

3 The SQCLP Programming Scheme

In this section we develop the SQCLP scheme with instances SQCLP$(\mathcal{S}, \mathcal{D}, \mathcal{C})$ announced in the introduction. The parameters $\mathcal{S}$, $\mathcal{D}$ and $\mathcal{C}$ stand for an admissible proximity relation, a qualification domain and a constraint domain with a certain signature Σ, respectively. By convention, we consider only those instances of the scheme whose parameters are chosen to constitute an *admissible triple* in the sense of Definition 2.14. We focus on declarative semantics, using an interpretation transformer and a logical inference system to provide alternative characterizations of least program models. We also discuss declarative semantics of goals and related approaches.

A brief remark regarding notation is in place here. For the sake of notational consistency with previous works (either by us or other authors) where similarity rather than proximity relations were used, we keep the symbol $\mathcal{S}$ for proximity relations and the uppercase letter S in the names of programming schemes. Our results, however, do not rely on the transitivity property from Definition 2.13.

3.1 Programs and their Declarative Semantics

A SQCLP$(\mathcal{S}, \mathcal{D}, \mathcal{C})$-program is a set $\mathcal{P}$ of *qualified program rules* (also called *qualified clauses*) of the form $C : A \overset{\alpha}{\leftarrow} B_1 \sharp w_1, \ldots, B_m \sharp w_m$, where A is a defined atom, $\alpha \in$

$D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ is called the *attenuation factor* of the clause and each $B_j \sharp w_j$ $(1 \leq j \leq m)$ is an atom B_j annotated with a so-called *threshold value* $w_j \in (D_{\mathcal{D}} \setminus \{\mathbf{b}\}) \uplus \{?\}$. The intended meaning of C is as follows: if for all $1 \leq j \leq m$ one has $B_j \sharp e_j$ (meaning that B_j holds with qualification value e_j) for some $e_j \trianglerighteq^? w_j$, then $A \sharp d$ (meaning that A holds with qualification value d) can be inferred for any $d \in D \setminus \{\mathbf{b}\}$ such that $d \trianglelefteq \alpha \circ \prod_{j=1}^{m} e_j$. By convention, $e_j \trianglerighteq^? w_j$ means $e_j \trianglerighteq w_j$ if $w_j \neq ?$ and is identically true otherwise. In practice threshold values equal to '?' and attenuation values equal to $\mathbf{t}$ can be omitted.

As motivating example, consider a SQCLP($\mathcal{S}, \mathcal{U} \otimes \mathcal{W}, \mathcal{R}$)-program $\mathcal{P}$ including the clauses and equations for $\mathcal{S}$ displayed in Figure 1. From Subsection 2.2 recall that qualification values in $\mathcal{U} \otimes \mathcal{W}$ are pairs (d, e) (where d represents a certainty degree and e represents a proof cost), as well as the behavior of $\trianglelefteq$ and $\circ$ in $\mathcal{U} \otimes \mathcal{W}$. Consider the problem of proving `goodWork(king_liar)`$\sharp(d, e)$ from $\mathcal{P}$. This can be achieved for $d = 0.75 \times \min\{d_1, d_2\}$, $e = 3 + \max\{e_1, e_2\}$ by using R_1 instantiated by $\{\texttt{X} \mapsto \texttt{king\_liar}, \texttt{Y} \mapsto \texttt{shakespeare}\}$, and going on to prove `famousAuthor(shakespeare)`$\sharp(d_1, e_1)$ for some $d_1 \geq 0.5$, $e_1 \leq 100$ and `wrote(shakespeare,king_liar)`$\sharp(d_2, e_2)$ for some d_2, e_2. Thanks to R_2, R_3 and $\mathcal{S}$, these proofs succeed with $(d_1, e_1) = (0.9, 1)$ and $(d_2, e_2) = (0.8, 2)$. Therefore, the desired proof succeeds with certainty degree $d = 0.75 \times \min\{0.9, 0.8\} = 0.6$, and proof cost $e = 3 + \max\{1, 2\} = 5$.

R_1 : `goodWork(X) <-(0.75,3)- famousAuthor(Y)#(0.5,100), wrote(Y,X)#?`
R_2 : `famousAuthor(shakespeare) <-(0.9,1)-`
R_3 : `wrote(shakespeare,king_lear) <-(1,1)-`

`S(king_lear,king_liar) = (0.8,2)`

Fig. 1. SQCLP($\mathcal{S}, \mathcal{U} \otimes \mathcal{W}, \mathcal{R}$) Program Fragment

---

It is useful to define some special types of program clauses and programs, as follows:

- A clause is called *attenuation-free* iff $\alpha = \mathbf{t}$. The name is justified because $\mathbf{t}$ is an identity element for the attenuation operator $\circ$, as explained in Subsection 2.2. By convention, attenuation-free clauses may be written with the simplified syntax $A \leftarrow B_1 \sharp w_1, \ldots, B_m \sharp w_m$.
- A clause is called *threshold-free* iff $w_j = ?$ for all $j = 1 \ldots m$. The name is justified because the threshold value $w_j = ?$ occurring as annotation of a body atom B_j does not impose any particular requirement to the qualification value of B_j. Threshold-free clauses may be written with the simplified syntax $A \stackrel{\alpha}{\leftarrow} B_1, \ldots, B_m$.
- A clause is called *qualification-free* iff it is both attenuation-free and threshold-free. These clauses may be written with the simplified syntax $A \leftarrow B_1, \ldots, B_m$. They behave just like those used in the classical CLP scheme.
- A clause is called *constraint-free* iff all its body atoms are defined.

- A program is called attenuation-free iff all its clauses are of this type. Threshold-free, qualification-free and constraint-free programs are defined similarly.

The more technical SQCLP($\mathcal{S},\mathcal{U},\mathcal{R}$)-program $\mathcal{P}$ presented below will serve as a *running example* to illustrate various points in the rest of the report.

Example 3.1 (*Running example*)
Assume unary constructors $c, c' \in DC^1$, binary predicate symbols $p, p', q \in DP^2$ and a ternary predicate symbol $r \in DP^3$. Consider the admissible triple $\langle \mathcal{S},\mathcal{U},\mathcal{R}\rangle$, where $\mathcal{S}$ is an $\mathcal{U}$-valued proximity relation such that $\mathcal{S}(c,c') = 0.9$ and $\mathcal{S}(p,p') = 0.8$. Let $\mathcal{P}$ be the SQCLP($\mathcal{S},\mathcal{U},\mathcal{R}$)-program consisting of the qualified clauses R_1, R_2 and R_3 listed below:

$R_1 : q(X, c(X)) \xleftarrow{1.0}$
$R_2 : p(c(X), Y) \xleftarrow{0.9} q(X,Y)\sharp 0.8$
$R_3 : r(c(X), Y, Z) \xleftarrow{0.9} q(X,Y)\sharp 0.8, cp_{\geq}(X, 0.0)\sharp ?$ □

As we will see in the Conclusions, the classical CLP scheme for Constraint Logic Programming originally introduced in (Jaffar and Lassez 1987) can be seen as a particular case of the SQCLP scheme. In the rest of this subsection we present a declarative semantics for SQCLP($\mathcal{S},\mathcal{D},\mathcal{C}$)-programs inspired by (Gabbrielli and Levi 1991; Gabbrielli et al. 1995). These papers provided three different program semantics $\mathcal{S}_i$ ($i = 1, 2, 3$) characterizing *valid ground solutions for goals*, *valid open solutions for goals* and *computed answers for goals* in CLP, respectively. In fact, the $\mathcal{S}_i$ semantics in (Gabbrielli and Levi 1991; Gabbrielli et al. 1995) were conceived as the CLP counterpart of previously known semantics for logic programming, namely the least ground Herbrand model semantics (Apt 1990; Lloyd 1987), the open Herbrand model semantics, also known as $\mathcal{C}$-semantics (Clark 1979; Falaschi et al. 1993), and the $\mathcal{S}$-semantics (Falaschi et al. 1989; Bossi et al. 1994); see (Apt and Gabbrielli 1994) for a very concise and readable overview.

In this report we restrict ourselves to develop a $\mathcal{S}_2$-like semantics which can be used to characterize valid open solutions for SQCLP goals as we will see in Subsection 3.2. As a basis for our semantics we use so-called *qc-atoms* of the form $A\sharp d \Leftarrow \Pi$, intended to assert that the atom A is entailed by the constraint set Π with qualification degree d. We also use a special entailment relation $\succcurlyeq_{\mathcal{D},\mathcal{C}}$ intended to capture some implications between qc-atoms whose validity depends neither on the proximity relation $\mathcal{S}$ nor on the semantics of defined predicates. A formal definition of these notions is as follows:

Definition 3.1 (*qc-atoms, observables and* ($\mathcal{D},\mathcal{C}$)*-entailment*)

1. *Qualified constrained atoms* (or simply *qc-atoms*) are statements of the form $A\sharp d \Leftarrow \Pi$, where $A \in \mathrm{At}(\Sigma, B, \mathcal{V}ar)$ is an atom, $d \in D$ is a qualification value, and $\Pi \subseteq \mathrm{Con}_{\mathcal{C}}$ is a finite set of constraints.
2. A qc-atom $A\sharp d \Leftarrow \Pi$ is called *defined*, *primitive* or *equational* according to the syntactic form of A.
3. A qc-atom $A\sharp d \Leftarrow \Pi$ is called *observable* iff $d \in D \setminus \{\mathbf{b}\}$ and Π is satisfiable.

4. Given two qc-atoms $\varphi : A\sharp d \Leftarrow \Pi$ and $\varphi' : A'\sharp d' \Leftarrow \Pi'$, we say that φ $(\mathcal{D},\mathcal{C})$-*entails* φ' (in symbols, $\varphi \succcurlyeq_{\mathcal{D},\mathcal{C}} \varphi'$) iff there is some $\mathcal{C}$-substitution θ satisfying $A' = A\theta$, $d' \preccurlyeq d$ and $\Pi' \models_{\mathcal{C}} \Pi\theta$. □

We will focus our attention on observable qc-atoms because they can be interpreted as observations of valid open solutions for atomic goals in SQCLP$(\mathcal{S},\mathcal{D},\mathcal{C})$ as we will see in Subsection 3.2. The example below illustrates the main technical ideas from Definition 3.1.

Example 3.2 (*Observable qc-atoms and* $(\mathcal{D},\mathcal{C})$*-entailment*)
Consider the admissible triple underlying Example 3.1 and the sets of $\mathcal{R}$-constraints:

$$\Pi = \{cp_{>}(X, 1.0),\ op_{+}(A, A, X),\ op_{\times}(2.0, A, Y)\}$$
$$\Pi' = \{cp_{\geq}(A, 3.0),\ op_{\times}(2.0, A, X),\ op_{+}(A, A, Y)\}$$

Then, the following are observable qc-atoms:

$$\varphi_1 = q(X, c'(Y))\sharp 0.9 \Leftarrow \Pi \qquad \varphi_3 = r(c'(Y), c(X), Z)\sharp 0.8 \Leftarrow \Pi$$
$$\varphi_2 = p'(c'(Y), c(X))\sharp 0.8 \Leftarrow \Pi \qquad \varphi'_3 = r(c'(Y), c(X), c(Z'))\sharp 0.7 \Leftarrow \Pi'$$

and the $(\mathcal{U},\mathcal{R})$-entailment $\varphi_3 \succcurlyeq_{\mathcal{U},\mathcal{R}} \varphi'_3$ is valid thanks to $\theta = \{Z \mapsto c(Z')\}$, which satisfies $r(c'(Y), c(X), c(Z')) = r(c'(Y), c(X), Z)\theta$, $0.7 \leq 0.8$ and $\Pi' \models_{\mathcal{R}} \Pi\theta$. □

The intended meaning of $\succcurlyeq_{\mathcal{D},\mathcal{C}}$ as an entailment relation not depending on the meanings of defined predicates motivates the first item in the next definition.

Definition 3.2 (*Interpretations*)
Let $\langle \mathcal{S},\mathcal{D},\mathcal{C}\rangle$ be any given admissible triple. Then:

1. A *qualified constrained interpretation* (or *qc-interpretation*) is a set $\mathcal{I}$ of observable defined qc-atoms closed under $(\mathcal{D},\mathcal{C})$-entailment. In other words, a set $\mathcal{I}$ of qc-atoms which satisfies the following two conditions:
   (a) Each $\varphi \in \mathcal{I}$ is an observable defined qc-atom.
   (b) If $\varphi \in \mathcal{I}$ and φ' is another defined observable qc-atom such that $\varphi \succcurlyeq_{\mathcal{D},\mathcal{C}} \varphi'$, then also $\varphi' \in \mathcal{I}$.
2. Assume any given qc-interpretation $\mathcal{I}$. For any observable qc-atom φ, we say that φ is valid in $\mathcal{I}$ modulo $\mathcal{S}$ (in symbols, $\mathcal{I} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$) iff some of the three cases below holds:
   (a) φ is defined and $\varphi \in \mathcal{I}$.
   (b) $\varphi : (t == s)\sharp d \Leftarrow \Pi$ is equational and $t \approx_{d,\Pi} s$.
   (c) $\varphi : \kappa\sharp d \Leftarrow \Pi$ is primitive and $\Pi \models_{\mathcal{C}} \kappa$. □

Note that a given interpretation $\mathcal{I}$ can include several observables $A\sharp d_i \Leftarrow \Pi$ for the same (possibly not ground) atom A but is not required to include on "optimal" observable $A\sharp d \Leftarrow \Pi$ with d computed as the *lub* of all d_i. By contrast, the other related works discussed in the Introduction view program interpretations as mappings $\mathcal{I}$ from the ground Herbrand base into some set of lattice elements (the real interval $[0, 1]$ in many cases). In such interpretations, each ground atom A has attached one single lattice element $d = \mathcal{I}(A)$ intended as "the optimal qualification"

for A. Our view of interpretations is closer to the expected operational behavior of goal solving systems and can be used to characterize the validity of solutions computed by such systems, as we will see in Subsection 3.2.

Note also that the notation $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ is defined only for the case that φ is observable. In the sequel, we will implicitly assume that φ is observable in any context where the notation $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ is used. The next technical result shows that validity in any given interpretation is closed under entailment.

Proposition 3.1 (*Entailment Property for Interpretations*)
Assume that $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ and $\varphi \succcurlyeq_{\mathcal{D},\mathcal{C}} \varphi'$. Then $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi'$.

Proof
Due to the hypothesis $\varphi \succcurlyeq_{\mathcal{D},\mathcal{C}} \varphi'$ we can assume $\varphi = (A \sharp d \Leftarrow \Pi)$, $\varphi' = (A' \sharp d' \Leftarrow \Pi')$ and some $\mathcal{C}$-substitution θ such that $A' = A\theta$, $d' \trianglelefteq d$ and $\Pi' \models_{\mathcal{C}} \Pi\theta$. We now distinguish cases according to the syntactic form of φ:

1. φ is defined. In this case, φ' is also defined. Moreover, $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ is equivalent to $\varphi \in \mathcal{I}$ because of Definition 3.2, which implies $\varphi' \in \mathcal{I}$ because qc-interpretations are closed under $\succcurlyeq_{\mathcal{D},\mathcal{C}}$, which is equivalent to $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi'$ because of Definition 3.2.
2. φ is equational. In this case A and A' have the form $t == s$ and $t\theta == s\theta$, respectively. Moreover, $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ is equivalent to $t \approx_{d,\Pi} s$ because of Definition 3.2, which implies $t\theta \approx_{d,\Pi\theta} s\theta$ because of Lemma 2.8, which trivially implies $t\theta \approx_{d',\Pi'} s\theta$ because of $\Pi' \models_{\mathcal{C}} \Pi\theta$ and $d' \trianglelefteq d$, which is equivalent to $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi'$ because of Definition 3.2.
3. φ is primitive. In this case A and A' have the form κ and $\kappa\theta$, respectively. Moreover, $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ is equivalent to $\Pi \models_{\mathcal{C}} \kappa$ because of Definition 3.2, which implies $\Pi\theta \models_{\mathcal{C}} \kappa\theta$ because of Lemma 2.1, which implies $\Pi' \models_{\mathcal{C}} \kappa\theta$ because of $\Pi' \models_{\mathcal{C}} \Pi\theta$, which is equivalent to $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi'$ because of Definition 3.2. □

The definition below explains when a given interpretation is regarded as a model of a given program, as well as the related notion of semantic consequence.

Definition 3.3 (*Models and semantic consequence*)
Let a SQCLP($\mathcal{S},\mathcal{D},\mathcal{C}$)-program $\mathcal{P}$ and an observable qc-atom $\varphi : p'(\overline{t'}_n) \sharp d \Leftarrow \Pi$ be given. φ is an *immediate consequence* of a qc-interpretation $\mathcal{I}$ via a program rule $(R_l : p(\bar{t}_n) \stackrel{\alpha}{\Leftarrow} B_1 \sharp w_1, \ldots, B_m \sharp w_m) \in \mathcal{P}$ iff there exist a $\mathcal{C}$-substitution θ and a choice of qualification values $d_0, d_1, \ldots, d_n, e_1, \ldots, e_m \in D \setminus \{\mathbf{b}\}$ such that:

(a) $\mathcal{S}(p', p) = d_0$
(b) $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} (t'_i == t_i\theta) \sharp d_i \Leftarrow \Pi$ (i.e. $t'_i \approx_{d_i,\Pi} t_i\theta$) for $i = 1 \ldots n$
(c) $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} B_j\theta \sharp e_j \Leftarrow \Pi$ with $e_j \trianglerighteq^{?} w_j$ for $j = 1 \ldots m$
(d) $d \trianglelefteq \bigsqcap_{i=0}^{n} d_i \sqcap \alpha \circ \bigsqcap_{j=1}^{m} e_j$ [i.e., $d \trianglelefteq d_i$ $(0 \leq i \leq n)$ and $d \trianglelefteq \alpha \circ e_j$ $(1 \leq j \leq m)$]

Note that the qualification value d attached to φ is limited by two kinds of upper bounds: d_i $(0 \leq i \leq n)$, i.e. the $\mathcal{S}$-proximity between $p'(\overline{t'}_n)$ and the head of $R_l\theta$; and $\alpha \circ e_j$ $(1 \leq j \leq m)$, i.e. the qualification values of the atoms in the body of $R_l\theta$ attenuated w.r.t. R_l's attenuation factor α. Moreover, the inequalities

$e_j \trianglerighteq^? w_j$ $(1 \leq j \leq m)$ are required in order to impose the threshold conditions within R_l's body. As already explained at the beginning of this subsection, $e_j \trianglerighteq^? w_j$ means that either $w_j = ?$ or else $w_j \in D \setminus \{\mathbf{b}\}$ and $e_j \trianglerighteq w_j$. Now we can define:

1. $\mathcal{I}$ is a *model* of a program rule $R_l \in \mathcal{P}$ (in symbols, $\mathcal{I} \models_{\mathcal{S},\mathcal{D},\mathcal{C}} R_l$) iff every defined observable qc-atom φ which is an immediate consequence of $\mathcal{I}$ via R_l verifies $\varphi \in \mathcal{I}$; and $\mathcal{I}$ is a *model* of $\mathcal{P}$ (in symbols, $\mathcal{I} \models_{\mathcal{S},\mathcal{D},\mathcal{C}} \mathcal{P}$) iff $\mathcal{I}$ is a model of every program rule $R_l \in \mathcal{P}$.
2. φ is a *semantic consequence* of $\mathcal{P}$ (in symbols, $\mathcal{P} \models_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$) iff $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ for every qc-interpretation $\mathcal{I}$ such that $\mathcal{I} \models_{\mathcal{S},\mathcal{D},\mathcal{C}} \mathcal{P}$. □

The next example may serve as a concrete illustration:

Example 3.3 (*Models and semantic consequence*)
Recall the SQCLP($\mathcal{S},\mathcal{U},\mathcal{R}$)-program $\mathcal{P}$ from Example 3.1. Let us show that the three qc-atoms φ_1, φ_2 and φ_3 from Example 3.2 are semantic consequences of $\mathcal{P}$:

1. Assume an arbitrary model $\mathcal{I} \models_{\mathcal{S},\mathcal{U},\mathcal{R}} \mathcal{P}$. Note that the atom underlying φ_1 and the head atom of R_1 are $q(X, c'(Y))$ and $q(X, c(X))$, respectively. Since $\mathcal{S}(c, c') = 0.9$ and $\Pi \models_{\mathcal{C}} X == Y$, φ_1 can be obtained as an immediate consequence of $\mathcal{I}$ via R_1 using $\theta = \varepsilon$. Therefore $\varphi_1 \in \mathcal{I}$ and we can conclude that $\mathcal{P} \models_{\mathcal{S},\mathcal{U},\mathcal{R}} \varphi_1$.
2. Assume an arbitrary model $\mathcal{I} \models_{\mathcal{S},\mathcal{U},\mathcal{R}} \mathcal{P}$. Consider the substitution $\theta = \{Y \mapsto c'(Y)\}$. Note that the atom underlying φ_2 and the head atom of $R_2\theta$ are $p'(c'(Y), c(X))$ and $p(c(X), c'(Y))$, respectively. Moreover, $\varphi_1 \in \mathcal{I}$ (due to the previous item) and the atom $q(X, c'(Y))$ underlying φ_1 is the same as the atom in the body of $R_2\theta$. These facts together with $\mathcal{S}(p, p') = 0.8$, $\mathcal{S}(c, c') = 0.9$ and $\Pi \models_{\mathcal{C}} X == Y$ allow to obtain φ_2 as an immediate consequence of $\mathcal{I}$ via R_2. Therefore $\varphi_2 \in \mathcal{I}$ and we can conclude that $\mathcal{P} \models_{\mathcal{S},\mathcal{U},\mathcal{R}} \varphi_2$.
3. Assume an arbitrary model $\mathcal{I} \models_{\mathcal{S},\mathcal{U},\mathcal{R}} \mathcal{P}$. Consider again the substitution $\theta = \{Y \mapsto c'(Y)\}$. Note that the atom underlying φ_3 and the head atom of $R_3\theta$ are $r(c'(Y), c(X), Z)$ and $r(c(X), c'(Y), Z)$, respectively. Moreover, the two annotated atoms $B_j\theta\sharp w_j$ $(1 \leq j \leq 2)$ occurring in the body of $R_3\theta$ are such that $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} B_j\theta\sharp e_j \Leftarrow \Pi$ for suitable values $e_j \geq^? w_j$, namely $e_1 = 0.9$ and $e_2 = 1.0$. Note that $e_1 = 0.9$ works because $B_1\theta$ is the atom $q(X, c'(Y))$ underlying φ_1 and $\varphi_1 \in \mathcal{I}$, as proved in the first item of this example. On the other hand, $e_2 = 1.0$ works because $B_2\theta$ is the primitive atom $cp_{\geq}(X, 0.0)$ which is trivially entailed by Π. All these facts, together with $\mathcal{S}(c, c') = 0.9$, $0.8 \leq 0.9 \times 0.9$ and $\Pi \models_{\mathcal{C}} X == Y$ allow to obtain φ_3 as an immediate consequence of $\mathcal{I}$ via R_3. Therefore $\varphi_3 \in \mathcal{I}$ and we can conclude that $\mathcal{P} \models_{\mathcal{S},\mathcal{U},\mathcal{R}} \varphi_3$. □

Now we are ready to obtain results on the declarative semantics of programs in the SQCLP scheme. We will characterize the observable consequences of a given program $\mathcal{P}$ in two different, but equivalent, ways: either using the interpretation transformer presented in Subsection 3.1.1, or using the extension of Horn Logic presented in Subsection 3.1.2. In both approaches, we will prove the existence of a least model $\mathcal{M}_{\mathcal{P}}$ for each given program $\mathcal{P}$.

3.1.1 A Fixpoint Semantics

A well-known way of characterizing models and least models of programs in declarative languages proceeds by considering a lattice structure for the family of all program interpretations, and using an interpretation transformer to compute the immediate consequences obtained from program rules. This kind of approach is well known for logic programming (van Emden and Kowalski 1976; Apt and van Emden 1982; Lloyd 1987; Apt 1990) and constraint logic programming (Gabbrielli and Levi 1991; Gabbrielli et al. 1995; Jaffar et al. 1998). It has been used also in various extensions of logic programming designed to support uncertain reasoning, such as quantitative logic programming (van Emden 1986), its extension to qualified logic programming (Rodríguez-Artalejo and Romero-Díaz 2008b) quantitative constraint logic programming (Riezler 1996; Riezler 1998), similarity-based logic programming (Sessa 2002) and proximity-based logic programming in the sense of Bousi∼Prolog (Julián-Iranzo and Rubio-Manzano 2009a).

The SQCLP scheme is intended to unify all these logic programming extensions in a common framework. This subsection is based on the declarative semantics given in (Rodríguez-Artalejo and Romero-Díaz 2008b; Rodríguez-Artalejo and Romero-Díaz 2008a), extended to deal with constraints and proximity relations. Our first result provides a lattice of program interpretations.

Proposition 3.2 (*Lattice of Interpretations*)
$\mathrm{Int}_{\mathcal{D},\mathcal{C}}$, defined as the set of all qc-interpretations over the qualification domain $\mathcal{D}$ and the constraint domain $\mathcal{C}$, is a complete lattice w.r.t. the set inclusion ordering $\subseteq$. Moreover, the bottom element $\perp\!\!\!\perp$ and the top element $\top\!\!\!\top$ of this lattice are characterized as $\perp\!\!\!\perp = \emptyset$ and $\top\!\!\!\top = \{\varphi \mid \varphi \text{ is a defined observable qc-atom}\}$ and for any subset $I \subseteq \mathrm{Int}_{\mathcal{D},\mathcal{C}}$ its greatest lower bound (glb) and least upper bound (lub) are characterized as follows:

1. The glb of I (written as $\bigsqcap I$) is $\bigcap_{\mathcal{I}\in I} \mathcal{I}$, understood as $\top\!\!\!\top$ if $I = \emptyset$; and
2. The lub of I (written as $\bigsqcup I$) is $\bigcup_{\mathcal{I}\in I} \mathcal{I}$, understood as $\perp\!\!\!\perp$ if $I = \emptyset$.

Proof
Both $\perp\!\!\!\perp$ and $\top\!\!\!\top$ are qc-interpretations because they are sets of defined observable qc-atoms and they are closed under $(\mathcal{D},\mathcal{C})$-entailment for trivial reasons, namely: $\perp\!\!\!\perp$ is empty and $\top\!\!\!\top$ includes all the defined observables. Moreover, they are the minimum and the maximum of $\mathrm{Int}_{\mathcal{D},\mathcal{C}}$ w.r.t. $\subseteq$ because $\perp\!\!\!\perp \subseteq \mathcal{I} \subseteq \top\!\!\!\top$ is trivially true for each $\mathcal{I} \in \mathrm{Int}_{\mathcal{D},\mathcal{C}}$. Thus, we have only left to prove 1. and 2.:

1. $\bigcap_{\mathcal{I}\in I} \mathcal{I}$ is obviously a set of defined observable qc-atoms because this is the case for each $\mathcal{I} \in I$. Given any $\varphi \in \bigcap_{\mathcal{I}\in I}$ and any observable defined qc-atom φ' such that $\varphi \succcurlyeq_{\mathcal{D},\mathcal{C}} \varphi'$, we get $\varphi' \in \bigcap_{\mathcal{I}\in I} \mathcal{I}$ as an obvious consequence of the fact that each $\mathcal{I} \in I$ is closed under $(\mathcal{D},\mathcal{C})$-entailment. Therefore, $\bigcap_{\mathcal{I}\in I} \mathcal{I} \in \mathrm{Int}_{\mathcal{D},\mathcal{C}}$. Obviously, $\bigcap_{\mathcal{I}\in I} \mathcal{I}$ is trivially a lower bound of I w.r.t. $\subseteq$. Moreover, $\bigcap_{\mathcal{I}\in I} \mathcal{I}$ is the glb of I, because any given lower bound $\mathcal{J}$ of I verifies $\mathcal{J} \subseteq \mathcal{I}$ for every $\mathcal{I} \in I$ and thus $\mathcal{J} \subseteq \bigcap_{\mathcal{I}\in I} \mathcal{I}$. Therefore, $\bigcap_{\mathcal{I}\in I} \mathcal{I} = \bigsqcap I$.

2. Using the properties of the union of a family of sets it is easy to prove that $\bigcup_{\mathcal{I}\in I}\mathcal{I} \in \mathrm{Int}_{\mathcal{D},\mathcal{C}}$ and also that $\bigcup_{\mathcal{I}\in I}\mathcal{I}$ is the lub of I w.r.t. $\subseteq$. A more detailed reasoning would be similar to the previous item. Therefore, $\bigcup_{\mathcal{I}\in I}\mathcal{I} = \bigsqcup I$. □

Next we define an *interpretation transformer* $\mathrm{T}_{\mathcal{P}}$, intended to compute the immediate consequences obtained from a given qc-interpretation via the program rules belonging to $\mathcal{P}$.

Definition 3.4 (*Interpretations Transformer*)
Let $\mathcal{P}$ be a fixed SQCLP($\mathcal{S},\mathcal{D},\mathcal{C}$)-program. The interpretations transformer $\mathrm{T}_{\mathcal{P}} : \mathrm{Int}_{\mathcal{D},\mathcal{C}} \to \mathrm{Int}_{\mathcal{D},\mathcal{C}}$ is defined by the condition:

$$\mathrm{T}_{\mathcal{P}}(\mathcal{I}) =_{\mathrm{def}} \{\varphi \mid \varphi \text{ is an immediate consequence of } \mathcal{I} \text{ via some } R_l \in \mathcal{P}\} \ . \quad \square$$

The computation of immediate consequences of a given qc-interpretation $\mathcal{I}$ via a given program rule R_l has been already explained in Definition 3.3. The following example illustrates the workings of $\mathrm{T}_{\mathcal{P}}$.

Example 3.4 (*Interpretation transformer in action*)
Recall again the SQCLP($\mathcal{S},\mathcal{U},\mathcal{R}$)-program $\mathcal{P}$ from Example 3.1 and the observable defined qc-atoms φ_1, φ_2 and φ_3 from Example 3.2. Then:

1. The arguments given in Example 3.3(1) can be easily reused to show that φ_1 is an immediate consequence of the empty interpretation $\perp\!\!\!\perp$ via the program rule R_1. Therefore, $\varphi_1 \in \mathrm{T}_{\mathcal{P}}(\perp\!\!\!\perp)$.
2. The arguments given in Example 3.3(2) can be easily reused to show that φ_1 is an immediate consequence of $\mathcal{I}$ via the program rule R_2, provided that $\varphi_1 \in \mathcal{I}$. Therefore, $\varphi_2 \in \mathrm{T}_{\mathcal{P}}(\mathrm{T}_{\mathcal{P}}(\perp\!\!\!\perp))$.
3. The arguments given in Example 3.3(3) can be easily reused to show that φ_3 is an immediate consequence of $\mathcal{I}$ via the program rule R_3, provided that $\varphi_1 \in \mathcal{I}$. Therefore, $\varphi_3 \in \mathrm{T}_{\mathcal{P}}(\mathrm{T}_{\mathcal{P}}(\perp\!\!\!\perp))$. □

The next proposition states the main properties of interpretation transformers.

Proposition 3.3 (*Properties of interpretation transformers*)
Let $\mathcal{P}$ be any fixed SQCLP($\mathcal{S},\mathcal{D},\mathcal{C}$)-program. Then:

1. $\mathrm{T}_{\mathcal{P}}$ is a well defined mapping, i.e. for all $\mathcal{I} \in \mathrm{Int}_{\mathcal{D},\mathcal{C}}$ one has $\mathrm{T}_{\mathcal{P}}(\mathcal{I}) \in \mathrm{Int}_{\mathcal{D},\mathcal{C}}$.
2. $\mathrm{T}_{\mathcal{P}}$ is monotonic and continuous.
3. For all $\mathcal{I} \in \mathrm{Int}_{\mathcal{D},\mathcal{C}}$ one has: $\mathcal{I} \models_{\mathcal{S},\mathcal{D},\mathcal{C}} \mathcal{P} \iff \mathrm{T}_{\mathcal{P}}(\mathcal{I}) \subseteq \mathcal{I}$, That is, the models of $\mathcal{P}$ are precisely the pre-fixpoints of $\mathrm{T}_{\mathcal{P}}$.

Proof
1. By definition, $\mathrm{T}_{\mathcal{P}}(\mathcal{I})$ is a set of observable defined qc-atoms. It is sufficient to prove that it is closed under $(\mathcal{D},\mathcal{C})$-entailment. Let us assume two observable defined qc-atoms φ and φ' such that $\varphi \in \mathrm{T}_{\mathcal{P}}(\mathcal{I})$ and $\varphi \succcurlyeq_{\mathcal{D},\mathcal{C}} \varphi'$. Because of $\varphi \succcurlyeq_{\mathcal{D},\mathcal{C}} \varphi'$ we can assume $\varphi : p(\overline{t}_n)\sharp d \Leftarrow \Pi$, $\varphi' : p(\overline{t'}_n)\sharp d' \Leftarrow \Pi'$ and some substitution θ such that $p(\overline{t'}_n) = p(\overline{t}_n)\theta$, $d' \preccurlyeq d$ and $\Pi' \models_{\mathcal{C}} \Pi\theta$. Because of $\varphi \in \mathrm{T}_{\mathcal{P}}(\mathcal{I})$, we can assume that φ is an immediate consequence of $\mathcal{I}$ via some $R_l \in \mathcal{P}$. More precisely, we can assume $(R_l : q(\overline{s}_n) \stackrel{\alpha}{\leftarrow} B_1\sharp w_1, \ldots, B_m\sharp w_m) \in \mathcal{P}$, some substitution σ and some qualification values $d_0, d_1, \ldots, d_n, e_1, \ldots, e_m \in D \setminus \{\mathbf{b}\}$ such that

(a) $\mathcal{S}(p,q) = d_0$,
(b) $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} (t_i == s_i\sigma)\sharp d_i \Leftarrow \Pi$ for $i = 1 \ldots n$,
(c) $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} B_j\sigma\sharp e_j \Leftarrow \Pi$ with $e_j \trianglerighteq^? w_j$ for $j = 1 \ldots m$,
(d) $d \trianglelefteqslant \bigsqcap_{i=0}^{n} d_i \sqcap \alpha \circ \bigsqcap_{j=1}^{m} e_j$ [i.e., $d \trianglelefteqslant d_i$ $(0 \leq i \leq n)$ and $d \trianglelefteqslant \alpha \circ e_j$ $(1 \leq j \leq m)$].

In order to show that $\varphi' \in \mathrm{T}_{\mathcal{P}}(\mathcal{I})$, we claim that φ' can be computed as an immediate consequence of $\mathcal{I}$ via the same program rule R_l, using the substitution $\sigma\theta$ and the qualification values $d_0, d_1, \ldots, d_n, e_1, \ldots, e_m \in D \setminus \{\mathbf{b}\}$. To justify this claim it is enough to check the following items:

(a') $\mathcal{S}(p,q) = d_0$,
(b') $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} (t'_i == s_i\sigma\theta)\sharp d_i \Leftarrow \Pi'$ for $i = 1 \ldots n$,
(c') $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} B_j\sigma\theta\sharp e_j \Leftarrow \Pi'$ with $e_j \trianglerighteq^? w_j$ for $j = 1 \ldots m$,
(d') $d \trianglelefteqslant \bigsqcap_{i=0}^{n} d_i \sqcap \alpha \circ \bigsqcap_{j=1}^{m} e_j$ [i.e., $d \trianglelefteqslant d_i$ $(0 \leq i \leq n)$ and $d \trianglelefteqslant \alpha \circ e_j$ $(1 \leq j \leq m)$].

These four items closely correspond to items (a)-(d) above. More specifically:
— Items (a') and (d') are identical to items (a) and (d), respectively.
— Regarding item (b'): For $i = 1 \ldots n$, $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} (t'_i == s_i\sigma\theta)\sharp d_i \Leftarrow \Pi$ is the same as $t_i\theta \approx_{d_i,\Pi'} s_i\sigma\theta$. Because of Lemma 2.8, this is a consequence of $\Pi' \models_{\mathcal{C}} \Pi\theta$ and $t_i \approx_{d_i,\Pi} s_i\sigma$, which is ensured by item (b).
— Regarding item (c'): For $j = 1 \ldots m$, $e_j \trianglerighteq^? w_j$ is ensured by item (c), and $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} B_j\sigma\theta\sharp e_j \Leftarrow \Pi'$ follows from $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} B_j\sigma\sharp e_j \Leftarrow \Pi$ –also ensured by item (c)– and the entailment property for interpretations (Proposition 3.1), which can be applied because $B_j\sigma\sharp e_j \Leftarrow \Pi \trianglerighteqslant_{\mathcal{D},\mathcal{C}} B_j\sigma\theta\sharp e_j \Leftarrow \Pi'$.

2. Monotonicity means that the inclusion $\mathrm{T}_{\mathcal{P}}(\mathcal{I}) \subseteq \mathrm{T}_{\mathcal{P}}(\mathcal{J})$ holds whenever $\mathcal{I} \subseteq \mathcal{J}$. This follows very easily from

$$(\spadesuit) \quad \mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi \text{ and } \mathcal{I} \subseteq \mathcal{J} \Longrightarrow \mathcal{J} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$$

which is a trivial consequence of Definition 3.2.
Continuity means that the equation $\mathrm{T}_{\mathcal{P}}(\bigsqcup I) = \bigsqcup\{\mathrm{T}_{\mathcal{P}}(\mathcal{I}) \mid \mathcal{I} \in I\}$ holds for any directed set $I \subseteq \mathrm{Int}_{\mathcal{D},\mathcal{C}}$ of qc-interpretations. Recall that $I \subseteq \mathrm{Int}_{\mathcal{D},\mathcal{C}}$ is called directed iff every finite subset $I_0 \subseteq I$ has some upper bound $\mathcal{I} \in I$. We show that $\mathrm{T}_{\mathcal{P}}(\bigsqcup I) = \bigsqcup\{Tp(\mathcal{I}) \mid \mathcal{I} \in I\}$ holds by proving the two inclusions separately:

(a) For each fixed $\mathcal{I}_0 \in I$, $\mathrm{T}_{\mathcal{P}}(\mathcal{I}_0) \subseteq \mathrm{T}_{\mathcal{P}}(\bigsqcup I)$ follows from $\mathcal{I}_0 \subseteq \bigsqcup I$ and monotonicity of $\mathrm{T}_{\mathcal{P}}$. Then, the inclusion $\bigsqcup\{\mathrm{T}_{\mathcal{P}}(\mathcal{I}) \mid \mathcal{I} \in I\} \subseteq \mathrm{T}_{\mathcal{P}}(\bigsqcup I)$ holds by definition of supremum.
(b) In order to prove the opposite inclusion $\mathrm{T}_{\mathcal{P}}(\bigsqcup I) \subseteq \bigsqcup\{\mathrm{T}_{\mathcal{P}}(\mathcal{I}) \mid \mathcal{I} \in I\}$, consider an arbitrary $\varphi \in \mathrm{T}_{\mathcal{P}}(\bigsqcup I)$. Due to Definition 3.4, φ is an immediate consequence of $\bigsqcup I$ via some program rule $R_l \in \mathcal{P}$. Because of the first item of Definition 3.3, φ is an immediate consequence of $\bigsqcup I$ via R_l due to finitely many qc-facts of the form $B_j\theta\sharp e_j \Leftarrow \Pi$ (coming from the body of a suitable instance of R_l) that are valid in $\bigsqcup I$. Because of $(\spadesuit)$ and the assumption that I is a directed set, it is possible to choose some $\mathcal{I}_0 \in I$ such that all the qc-facts $B_j\theta\sharp e_j \Leftarrow \Pi$ are valid in $\mathcal{I}_0$. Then, φ is an immediate consequence of this particular $\mathcal{I}_0 \in I$ via R_l. Therefore, $\varphi \in \mathrm{T}_{\mathcal{P}}(\mathcal{I}_0) \subseteq \bigsqcup\{\mathrm{T}_{\mathcal{P}}(\mathcal{I}) \mid \mathcal{I} \in I\}$.

3. According to Definition 3.3, $\mathcal{I} \models_{\mathcal{S},\mathcal{D},\mathcal{C}} \mathcal{P}$ holds iff every observable defined qc-atom φ which is an immediate consequence of $\mathcal{I}$ via the program rules $R_l \in \mathcal{P}$ verifies $\varphi \in \mathcal{I}$. According to Definition 3.4, $\mathrm{T}_{\mathcal{P}}(\mathcal{I})$ is just the set of all the defined observable qc-atoms φ that can be obtained as immediate consequences of $\mathcal{I}$ via the program rules $R_l \in \mathcal{P}$. Consequently, $\mathcal{I} \models_{\mathcal{S},\mathcal{D},\mathcal{C}} \mathcal{P}$ holds iff $\mathrm{T}_{\mathcal{P}}(\mathcal{I}) \subseteq \mathcal{I}$. □

The theorem below is the main result in this subsection.

Theorem 3.1 (*Fixpoint characterization of least program models*)
Every SQCLP($\mathcal{S},\mathcal{D},\mathcal{C}$)-program $\mathcal{P}$ has a *least model* $\mathcal{M}_{\mathcal{P}}$, smaller than any other model of $\mathcal{P}$ w.r.t. the set inclusion ordering of the interpretation lattice $\mathrm{Int}_{\mathcal{D},\mathcal{C}}$. Moreover, $\mathcal{M}_{\mathcal{P}}$ can be characterized as the *least fixpoint* of $\mathrm{T}_{\mathcal{P}}$ as follows:

$$\mathcal{M}_{\mathcal{P}} = lfp(\mathrm{T}_{\mathcal{P}}) = \bigcup_{k \in \mathbb{N}} \mathrm{T}_{\mathcal{P}}{\uparrow}^k(\bot\!\!\!\bot) \ . \quad \square$$

Proof
As usual, a given $\mathcal{I} \in \mathrm{Int}_{\mathcal{D},\mathcal{C}}$ is called a fixpoint of $\mathrm{T}_{\mathcal{P}}$ iff $\mathrm{T}_{\mathcal{P}}(\mathcal{I}) = \mathcal{I}$, and $\mathcal{I}$ is called a pre-fixpoint of $\mathrm{T}_{\mathcal{P}}$ iff $\mathrm{T}_{\mathcal{P}}(\mathcal{I}) \subseteq \mathcal{I}$. Due to a well-known theorem by Knaster and Tarski, see (Tarski 1955), a monotonic mapping from a complete lattice into itself always has a least fixpoint which is also its least pre-fixpoint. In the case that the mapping is continuous, its least fixpoint can be characterized as the lub of the sequence of lattice elements obtained by reiterated application of the mapping to the bottom element. Combining these results with Proposition 3.3 trivially proves the theorem. □

3.1.2 An equivalent Proof-theoretic Semantics

In order to give a logical view of program semantics and an alternative characterization of least program models, we define the *Proximity-based Qualified Constrained Horn Logic* SQCHL($\mathcal{S},\mathcal{D},\mathcal{C}$) as a formal inference system consisting of the three inference rules displayed in Figure 2.

SQDA $\dfrac{(\ (t'_i == t_i\theta)\sharp d_i \Leftarrow \Pi\ )_{i=1\ldots n} \quad (\ B_j\theta\sharp e_j \Leftarrow \Pi\ )_{j=1\ldots m}}{p'(\overline{t'}_n)\sharp d \Leftarrow \Pi}$

if $(p(\overline{t}_n) \overset{\alpha}{\Leftarrow} B_1\sharp w_1, \ldots, B_m\sharp w_m) \in \mathcal{P}$, θ subst., $\mathcal{S}(p',p) = d_0 \neq \mathbf{b}$, $e_j \ni^? w_j$ $(1 \leq j \leq m)$ and $d \trianglelefteq \bigsqcap_{i=0}^{n} d_i \sqcap \alpha \circ \bigsqcap_{j=1}^{m} e_j$.

SQEA $\dfrac{}{(t == s)\sharp d \Leftarrow \Pi}$ if $t \approx_{d,\Pi} s$.

SQPA $\dfrac{}{\kappa\sharp d \Leftarrow \Pi}$ if $\Pi \models_{\mathcal{C}} \kappa$.

Fig. 2. Proximity-based Qualified Constrained Horn Logic

The three inference rules are intended to work with observable qc-atoms. Rule **SQDA** is used to infer defined qc-atoms. It formalizes an extension of the classical *Modus Ponens* inference, allowing to infer a defined qc-atom $p'(\overline{t'}_n)\sharp d \Leftarrow \Pi$ by means of an instance of a program clause with head $p(\overline{t}_n)\theta$ and body atoms $B_j\theta\sharp w_j$. The n premises $(t'_i == t_i\theta)\sharp d_i \Leftarrow \Pi$ combined with the side condition $\mathcal{S}(p', p) = d_0 \neq \mathbf{b}$ ensure the "equality" between $p'(\overline{t'}_n)$ and $p(\overline{t}_n)\theta$ modulo $\mathcal{S}$; the m premises $B_j\theta\sharp e_j \Leftarrow \Pi$ require to prove the body atoms; and the side conditions $e_j \trianglerighteq^? w_j$ and $d \trianglelefteq \bigsqcap_{i=0}^{n} d_i \sqcap \alpha \circ \bigsqcap_{j=1}^{m} e_j$ check the threshold conditions of the body atoms and impose the proper relationships between the qualification value attached to the conclusion and the qualification values attached to the premises. In particular, the inequality $d \trianglelefteq \alpha \circ \bigsqcap_{j=1}^{m} e_j$ is imposed, meaning that the qualification value attached to a clause's head cannot exceed the glb of the qualification values attached to the body atoms attenuated by the clause's attenuation factor. Rules **SQEA** and **SQPA** are used to infer equational and primitive qc-atoms, respectively. Rule **SQEA** is designed to work with term proximity w.r.t. Π in the sense of Definition 2.16, inferring $(t == s)\sharp d \Leftarrow \Pi$ just in the case that $t \approx_{d,\Pi} s$ holds. Rule **SQPA** infers $\kappa\sharp d \Leftarrow \Pi$ for an arbitrary $d \in D \setminus \{\mathbf{b}\}$, provided that $\Pi \models_{\mathcal{C}} \kappa$ holds. This makes sense because the requirements for admissible triples in Definition 2.14 include the assumption that $\mathcal{S}(p, p') \neq \mathbf{b}$ cannot happen if $p, p' \in PP$ are syntactically different primitive predicate symbols.

As usual in formal inference systems, SQCHL$(\mathcal{S}, \mathcal{D}, \mathcal{C})$ proofs can be represented as *proof trees* T whose nodes correspond to qc-atoms, each node being inferred from its children by means of some SQCHL$(\mathcal{S}, \mathcal{D}, \mathcal{C})$ inference step. In the rest of the report we will use the following notations:

- $\|T\|$ will denote the *size* of the proof tree T, measured as its number of nodes, which equals the number of inference steps in the SQCHL$(\mathcal{S}, \mathcal{D}, \mathcal{C})$ proof represented by T.
- $\|T\|_d$ will denote the number of nodes of the proof tree T that represent conclusions of **SQDA** inference steps. Obviously, $\|T\|_d \leq \|T\|$.
- $\mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ will indicate that φ can be inferred from $\mathcal{P}$ in SQCHL$(\mathcal{S}, \mathcal{D}, \mathcal{C})$.
- $\mathcal{P} \vdash^{k}_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ will indicate that φ can be inferred from $\mathcal{P}$ in SQCHL$(\mathcal{S}, \mathcal{D}, \mathcal{C})$ using some proof tree T such that $\|T\|_d = k$.

The next example shows a SQCHL$(\mathcal{S}, \mathcal{U}, \mathcal{R})$ proof tree.

Example 3.5 (*SQCHL$(\mathcal{S}, \mathcal{D}, \mathcal{C})$ proof tree*)
Recall the proximity relation $\mathcal{S}$ and the program $\mathcal{P}$ from our running Example 3.1, as well as the observable qc-statement $\varphi_2 = p'(c'(Y), c(X))\sharp 0.8 \Leftarrow \Pi$ already known from Example 3.2. A SQCHL$(\mathcal{S}, \mathcal{U}, \mathcal{R})$ proof tree witnessing $\mathcal{P} \vdash_{\mathcal{S},\mathcal{U},\mathcal{R}} \varphi_2$ can be displayed as follows:

$$\spadesuit = \dfrac{\dfrac{}{(Y == Y)\sharp 1.0 \Leftarrow \Pi}\,(5) \qquad \dfrac{}{(c(X) == c(Y))\sharp 1.0 \Leftarrow \Pi}\,(6)}{q(Y, c(X))\sharp 1.0 \Leftarrow \Pi}\,(4)$$

$$\dfrac{\dfrac{}{(c'(Y) == c(Y))\sharp 0.8 \Leftarrow \Pi}\ (2) \qquad \dfrac{}{(c(X) == c(X))\sharp 1.0 \Leftarrow \Pi}\ (3) \qquad \spadesuit\ (4)}{p'(c'(Y), c(X))\sharp 0.8 \Leftarrow \Pi}\ (1)$$

The inference steps in this proof are commented below. For the sake of clarity, we have used a different variant of the corresponding program clause for each each application of the inference rule **SQDA**.

(1) **SQDA** step with clause $R_1 = (\ p(c(X_1), Y_1) \xleftarrow{0.9} q(X_1, Y_1)\ )$ instantiated by substitution $\theta_1 = \{X_1 \mapsto Y, Y_1 \mapsto c(X)\}$. Note that 0.8 satisfies $0.8 \leq \mathcal{S}(p, p') = 0.8$, $0.8 \leq 0.8$, $0.8 \leq 1.0$, $0.8 \leq 0.9 \times 1.0$.
(2) **SQEA** step. $c'(Y) \approx_{0.8,\Pi} c(Y)$ holds due to $c'(Y) \approx_{\Pi} c'(Y)$, $c(Y) \approx_{\Pi} c(Y)$ and $c'(Y) \approx_{0.8} c(Y)$.
(3) **SQEA** step. $c(X) \approx_{1.0,\Pi} c(X)$ holds for trivial reasons.
(4) **SQDA** step with clause $R_2 = (\ q(X_2, c(X_2)) \xleftarrow{1.0}\ )$ instantiated by substitution $\theta_2 = \{X_2 \mapsto Y\}$. Note that 1.0 satisfies $1.0 \leq \mathcal{S}(q, q) = 1.0$ and $1.0 \leq 1.0$.
(5) **SQEA** step. $Y \approx_{1.0,\Pi} Y$ holds for trivial reasons.
(6) **SQEA** step. $c(X) \approx_{1.0,\Pi} c(Y)$ holds due to $c(X) \approx_{\Pi} c(Y)$ (which follows from $\Pi \models_{\mathcal{R}} X == Y$) and $c(X) \approx_{1.0} c(X)$. $\square$

The next technical lemma establishes two basic properties of formal inference in the SQCHL$(\mathcal{S}, \mathcal{D}, \mathcal{C})$ logic.

Lemma 3.1 (*Properties of SQCHL$(\mathcal{S}, \mathcal{D}, \mathcal{C})$ derivability*)
Let $\mathcal{P}$ be any SQCLP$(\mathcal{S}, \mathcal{D}, \mathcal{C})$-program. Then:

1. *$\mathcal{P}$-independent Inferences*:
   Given any $\mathcal{C}$-based qc-atom φ and any qc-interpretation $\mathcal{I}$, one has:
   $$\mathcal{P} \vdash^{0}_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi \Longleftrightarrow \mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi \Longleftrightarrow \mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi\ .$$
2. *Entailment Property for Programs*:
   Given any pair of qc-atoms φ and φ' such that $\mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ with inference proof tree T and $\varphi \succcurlyeq_{\mathcal{D},\mathcal{C}} \varphi'$, then $\mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi'$ with an inference proof tree T' of the same size and structure as T.

Proof of $\mathcal{P}$-independent Inferences
Since φ is $\mathcal{C}$-based, we can assume $\varphi = A\sharp d \Leftarrow \Pi$ where A is either an equation or a primitive atom. In both cases the equivalence $\mathcal{P} \vdash^{0}_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi \Longleftrightarrow \mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ is obvious. In order to prove the equivalence $\mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi \Longleftrightarrow \mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ we distinguish the two cases:

1. φ is equational. Then A has the form $t == s$. Considering the SQCHL$(\mathcal{S}, \mathcal{D}, \mathcal{C})$-inference rule **SQEA** and the second item of Definition 3.2, we get
   $$\mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi \Longleftrightarrow s \approx_{d,\Pi} t \Longleftrightarrow \mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi\ .$$

2. φ is primitive. Then A is a primitive atom κ. Considering the SQCHL$(\mathcal{S},\mathcal{D},\mathcal{C})$-inference rule **SQPA** and the second item of Definition 3.2, we get

$$\mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi \Longleftrightarrow \Pi \models_{\mathcal{C}} \kappa \Longleftrightarrow \mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi \ . \quad \square$$

Proof of Entailment Property for Programs
Due to the hypothesis $\varphi \succcurlyeq_{\mathcal{D},\mathcal{C}} \varphi'$ and Definition 3.1, we can assume $\varphi = A\sharp d \Leftarrow \Pi$ and $\varphi' = A'\sharp d' \Leftarrow \Pi'$ with $A' = A\theta$, $d' \trianglelefteq d$ and $\Pi' \models_{\mathcal{C}} \Pi\theta$ for some substitution θ. We reason by complete induction on $\|T\|$. There are three possible cases, according to the the syntactic form of the atom A. In each case we argue how to build the desired proof tree T'.

1. A is a defined atom: In this case, A is $p(\overline{t}_n)$ with $p \in DP^n$, and A': is $p(\overline{t'}_n)$ with $p(\overline{t'}_n) = p(\overline{t}_n)\theta$. Moreover, T must be a proof tree of the following form:

$$T: \dfrac{\left(\dfrac{}{(t_i == s_i\sigma)\sharp d_i \Leftarrow \Pi}\right)_{i=1\ldots n} \quad \left(\dfrac{\cdots}{B_j\sigma\sharp e_j \Leftarrow \Pi}\right)_{j=1\ldots m}}{p(\overline{t}_n)\sharp d \Leftarrow \Pi}\ \textbf{SQDA}$$

where:

- The **SQDA** root inference uses some $R_l : (q(\overline{s}_n) \overset{\alpha}{\leftarrow} B_1\sharp w_1, \ldots, B_m\sharp w_m) \in \mathcal{P}$, some substitution σ and some qualification values $d_0, d_1, \ldots, d_n, e_1, \ldots e_m \in D \setminus \{\mathbf{b}\}$ such that $\mathcal{S}(p,q) = d_0 \neq \mathbf{b}$, $d \trianglelefteq d_i$ $(0 \leq i \leq n)$ and $d \trianglelefteq \alpha \circ e_j$ $(1 \leq j \leq m)$.
- For $i = 1 \ldots n$, $(t_i == s_i\sigma)\sharp d_i \Leftarrow \Pi$ has a proof tree T_i^h with $\|T_i^h\| < \|T\|$.
- For $j = 1 \ldots m$, $B_j\sigma\sharp e_j \Leftarrow \Pi$ has a proof tree T_j^b with $\|T_j^b\| < \|T\|$.

Then, T' can be built as a proof tree of the form:

$$T': \dfrac{\left(\dfrac{}{(t'_i == s_i\sigma\theta)\sharp d_i \Leftarrow \Pi'}\right)_{i=1\ldots n} \quad \left(\dfrac{\cdots}{B_j\sigma\theta\sharp e_j \Leftarrow \Pi'}\right)_{j=1\ldots m}}{p(\overline{t'}_n)\sharp d' \Leftarrow \Pi'}\ \textbf{SQDA}$$

where:

- The **SQDA** root inference uses the same program clause $R_l \in \mathcal{P}$, the substitution $\sigma\theta$ and the same qualification values d_i $(0 \leq i \leq n)$ and e_j $(1 \leq j \leq m)$, satisfying $\mathcal{S}(p,q) = d_0 \neq \mathbf{b}$, $d' \trianglelefteq d \trianglelefteq d_i$ $(0 \leq i \leq n)$ and $d' \trianglelefteq d \trianglelefteq \alpha \circ e_j$ $(1 \leq j \leq m)$.
- For $i = 1 \ldots n$, $(t'_i == s_i\sigma\theta)\sharp d_i \Leftarrow \Pi'$ has a proof tree $T_i^{'h}$ of the same size and structure as T_i^h. In fact, $T_i^{'h}$ can be obtained by induction hypothesis applied to T_i^h, which is allowed because $\|T_i^h\| < \|T\|$ and $(t_i == s_i\sigma)\sharp d_i \Leftarrow \Pi \succcurlyeq_{\mathcal{D},\mathcal{C}} (t'_i == s_i\sigma\theta)\sharp d_i \Leftarrow \Pi'$. Note that this entailment holds thanks to substitution θ, since $t'_i = t_i\theta$ and $\Pi' \models_{\mathcal{C}} \Pi\theta$.
- For $j = 1 \ldots m$, $B_j\sigma\theta\sharp e_j \Leftarrow \Pi'$ has a proof tree $T_j^{'b}$ of the same size and structure as T_j^b. In fact, $T_j^{'b}$ can be obtained by induction hypothesis applied to T_j^b, which is allowed because $\|T_j^b\| < \|T\|$ and $B_j\sigma\sharp e_j \Leftarrow \Pi \succcurlyeq_{\mathcal{D},\mathcal{C}} B_j\sigma\theta\sharp e_j \Leftarrow \Pi'$. Note that this entailment holds thanks to substitution θ, since $\Pi' \models_{\mathcal{C}} \Pi\theta$.

By construction, T' has the same size and structure as T, as desired.

2. A is an equation: In this case, $A : t == s$ and $A' : t' == s'$ with $t' = t\theta$, $s' = s\theta$. Moreover, T must consist of one single node $(t == s)\sharp d \Leftarrow \Pi$ inferred by means of **SQEA**. Therefore, $t \approx_{d,\Pi} s$ holds. This implies $t\theta \approx_{d,\Pi\theta} s\theta$ (i.e. $t' \approx_{d,\Pi\theta} s'$) due to the Substitution Lemma 2.8. From this we conclude $t' \approx_{\Pi'} s'$ due to $d' \trianglelefteq d$ and $\Pi' \models_{\mathcal{C}} \Pi\theta$. Therefore, T' can be built as a proof tree consisting of one single node $(t' == s')\sharp d' \Leftarrow \Pi'$ inferred by means of **SQEA**.
3. A is a primitive atom: In this case, $A : \kappa$ and $A' : \kappa' = \kappa\theta$. Moreover, T must consist of one single node $\kappa\sharp d \Leftarrow \Pi$ inferred by means of **SQPA**. Therefore, $\Pi \models_{\mathcal{C}} \kappa$ holds. This implies $\Pi\theta \models_{\mathcal{C}} \kappa\theta$ due to the Substitution Lemma 2.1. From this we conclude $\Pi' \models_{\mathcal{C}} \kappa'$ due to $\kappa' = \kappa\theta$ and $\Pi' \models_{\mathcal{C}} \Pi\theta$. Therefore, T' can be built as a proof tree consisting of one single node $\kappa'\sharp d' \Leftarrow \Pi'$ inferred by means of **SQPA**. □

The following theorem is the main result in this subsection. It characterizes the least model of a SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-program $\mathcal{P}$ w.r.t. the logic SQCHL($\mathcal{S}, \mathcal{D}, \mathcal{C}$):

Theorem 3.2 (*Logical characterization of least program models*)
For any SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-program $\mathcal{P}$, its least model can be characterized as:

$$\mathcal{M}_{\mathcal{P}} = \{\varphi \mid \varphi \text{ is a defined observable qc-atom and } \mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi\} \ .$$

Proof
By Theorem 3.1, we already know that $\mathcal{M}_{\mathcal{P}} = \bigcup_{k\in\mathbb{N}} \mathrm{T}_{\mathcal{P}}\uparrow^k(\perp\!\!\!\perp)$. Therefore, it is sufficient to prove that the two implications

1. $\mathcal{P} \vdash^k_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi \Longrightarrow \exists k' : \varphi \in \mathrm{T}_{\mathcal{P}}\uparrow^{k'}(\perp\!\!\!\perp)$
2. $\varphi \in \mathrm{T}_{\mathcal{P}}\uparrow^k(\perp\!\!\!\perp) \Longrightarrow \exists k' : \mathcal{P} \vdash^{k'}_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$

hold for any defined observable qc-atom $\varphi = p(\bar{t}_n)\sharp d \Leftarrow \Pi$ and for any integer value $k \geq 1$. We prove both implications within one single inductive reasoning on k.

Basis ($k = 1$)**.**
— *Implication 1.* Assume $\mathcal{P} \vdash^1_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$. Then, due to the single **SQDA** inference, there must exist some $R_l = (q(\overline{s}_n) \xleftarrow{\alpha}) \in \mathcal{P}$ with empty body, some substitution θ and some $d_0, d_1, \ldots, d_n \in D \setminus \{\mathbf{b}\}$ such that $\mathcal{P} \vdash^0_{\mathcal{S},\mathcal{D},\mathcal{C}} (t_i == s_i\theta)\sharp d_i \Leftarrow \Pi$ for $i = 1 \ldots n$, $\mathcal{S}(p, q) = d_0 \neq \mathbf{b}$, $d \trianglelefteq d_i$ $(0 \leq i \leq n)$ and $d \trianglelefteq \alpha$. Then $\perp\!\!\!\perp \vdash^0_{\mathcal{S},\mathcal{D},\mathcal{C}} (t_i == s_i\theta)\sharp d_i \Leftarrow \Pi$ holds for $i = 1 \ldots n$, because of Lemma 3.1(1). Therefore φ is an immediate consequence of $\perp\!\!\!\perp$ via R_l, which guarantees $\varphi \in \mathrm{T}_{\mathcal{P}}\uparrow^1(\perp\!\!\!\perp)$.
— *Implication 2.* Assume now $\varphi \in \mathrm{T}_{\mathcal{P}}\uparrow^1(\perp\!\!\!\perp)$. Then φ must be an immediate consequence of $\perp\!\!\!\perp$ via some $R_l = (q(\overline{s}_n) \xleftarrow{\alpha}) \in \mathcal{P}$ with empty body. Then there are some substitution θ and some $d_0, d_1, \ldots, d_n \in D \setminus \{\mathbf{b}\}$ such that $\perp\!\!\!\perp \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} (t_i == s_i\theta)\sharp d_i \Leftarrow \Pi$ for $i = 1 \ldots n$, $\mathcal{S}(p, q) = d_0 \neq \mathbf{b}$, $d \trianglelefteq d_i$ $(0 \leq i \leq n)$ and $d \trianglelefteq \alpha$. Again because of Lemma 3.1(1), we get $\mathcal{P} \vdash^0_{\mathcal{S},\mathcal{D},\mathcal{C}} (t_i == s_i\theta)\sharp d_i \Leftarrow \Pi$ for $i = 1 \ldots n$, which guarantees $\mathcal{P} \vdash^1_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ with one single **SQDA** inference using R_l instantiated by θ.

Inductive step ($k > 1$).

— *Implication 1.* Assume $\mathcal{P} \vdash^{k}_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$. Since the root inference must be **SQDA**, there must exist some program rule $(R_l : q(\overline{s}_n) \overset{\alpha}{\Leftarrow} B_1 \sharp w_1, \ldots, B_m \sharp w_m) \in \mathcal{P}$, some substitution θ and some qualification values $d_0, d_1, \ldots, d_n, e_1, \ldots, e_m \in D \setminus \{\mathbf{b}\}$ such that

- $\mathcal{P} \vdash^{0}_{\mathcal{S},\mathcal{D},\mathcal{C}} \phi_i = ((t_i == s_i\theta)\sharp d_i \Leftarrow \Pi)$ for $i = 1 \ldots n$,
- $\mathcal{P} \vdash^{k_j}_{\mathcal{S},\mathcal{D},\mathcal{C}} \psi_j = (B_j\theta\sharp e_j \Leftarrow \Pi)$ with $e_j \trianglerighteq^{?} w_j$ for $j = 1 \ldots m$, and
- $\mathcal{S}(p,q) = d_0 \neq \mathbf{b}$, $d \trianglelefteq d_i$ $(0 \leq i \leq n)$ and $d \trianglelefteq \alpha \circ e_j$ $(1 \leq j \leq m)$

where $\Sigma_{j=1}^{m} k_j = k - 1$. For each $j = 1 \ldots m$, either ψ_j is defined, and then induction hypothesis yields some k'_j such that $\psi_j \in \mathrm{T}_{\mathcal{P}}\uparrow^{k'_j}(\perp\!\!\!\perp)$ and therefore also $\mathrm{T}_{\mathcal{P}}\uparrow^{k'_j}(\perp\!\!\!\perp) \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \psi_j$; or else ψ_j is not defined and then $\mathrm{T}_{\mathcal{P}}\uparrow^{k'_j}(\perp\!\!\!\perp) \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \psi_j$ for any arbitrarily chosen k'_j, by Lemma 3.1(1). Then $l = max\{k'_j \mid 1 \leq j \leq m\}$ verifies that φ is an immediate consequence of $\mathrm{T}_{\mathcal{P}}\uparrow^{l}(\perp\!\!\!\perp)$ via R_l, which implies $\varphi \in \mathrm{T}_{\mathcal{P}}\uparrow^{k'}(\perp\!\!\!\perp)$ for $k' = l + 1$.

— *Implication 2.* Assume $\varphi \in \mathrm{T}_{\mathcal{P}}\uparrow^{k}(\perp\!\!\!\perp) = \mathrm{T}_{\mathcal{P}}(\mathrm{T}_{\mathcal{P}}\uparrow^{k-1}(\perp\!\!\!\perp))$. Then φ is an immediate consequence of $Tp\uparrow^{k-1}(\perp\!\!\!\perp)$ via some clause $(R_l : q(\overline{s}_n) \overset{\alpha}{\Leftarrow} B_1 \sharp w_1, \ldots, B_m \sharp w_m) \in \mathcal{P}$. Therefore, there exist some substitution θ and some qualification values $d_0, d_1, \ldots, d_n, e_1, \ldots, e_m \in D \setminus \{\mathbf{b}\}$ such that:

- $\mathrm{T}_{\mathcal{P}}\uparrow^{k-1}(\perp\!\!\!\perp) \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \phi_i = ((t_i == s_i\theta)\sharp d_i \Leftarrow \Pi)$ for $i = 1 \ldots n$,
- $\mathrm{T}_{\mathcal{P}}\uparrow^{k-1}(\perp\!\!\!\perp) \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \psi_j = (B_j\theta\sharp e_j \Leftarrow \Pi)$ with $e_j \trianglerighteq^{?} w_j$ for $j = 1 \ldots m$, and
- $\mathcal{S}(p,q) = d_0 \neq \mathbf{b}$, $d \trianglelefteq d_i$ $(0 \leq i \leq n)$ and $d \trianglelefteq \alpha \circ e_j$ $(1 \leq j \leq m)$.

For each $i = 1 \ldots n$, Lemma 3.1(1) yields $\mathcal{P} \vdash^{0}_{\mathcal{S},\mathcal{D},\mathcal{C}} \phi_i$. For each $j = 1 \ldots m$, either ψ_j is defined, in which case $\psi_j \in \mathrm{T}_{\mathcal{P}}\uparrow^{k-1}(\perp\!\!\!\perp)$, $k - 1 \geq 1$, and induction hypothesis yields some k'_j such that $\mathcal{P} \vdash^{k'_j}_{\mathcal{S},\mathcal{D},\mathcal{C}} \psi_j$; or else ψ_j is not defined, in which case $\mathcal{P} \vdash^{k'_j}_{\mathcal{S},\mathcal{D},\mathcal{C}} \psi_j$ for $k'_j = 0$, by Lemma 3.1(1). In these conditions, $\mathcal{P} \vdash^{k'}_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ holds for $k' = 1 + \Sigma_{j=1}^{m} k'_j$, with a proof tree using a **SQDA** root inference based on R_l instantiated by θ. □

As an easy consequence of the previous theorem we get:

Corollary 3.1 (*SQCHL*$(\mathcal{S},\mathcal{D},\mathcal{C})$ *is sound and complete*)
For any SQCLP$(\mathcal{S},\mathcal{D},\mathcal{C})$-program $\mathcal{P}$ and any observable qc-atom φ, the following three statements are equivalent:

$$(a)\ \mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi \qquad (b)\ \mathcal{P} \models_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi \qquad (c)\ \mathcal{M}_{\mathcal{P}} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$$

Moreover, we also have:

1. *Soundness*: $\mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi \Longrightarrow \mathcal{P} \models_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$.
2. *Completeness*: $\mathcal{P} \models_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi \Longrightarrow \mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$.

Proof
Soundness and completeness are just a trivial consequence of $(a) \Leftrightarrow (b)$. To finish the proof it suffices to prove the two equivalences $(a) \Leftrightarrow (c)$ and $(b) \Leftrightarrow (c)$. This is done as follows:

$[(a) \Leftrightarrow (c)]$ In the case that φ is a defined qc-atom, $\mathcal{M}_\mathcal{P} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ reduces to $\varphi \in \mathcal{M}_\mathcal{P}$ which is equivalent to $\mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ by Theorem 3.2. Otherwise, $\mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi \iff \mathcal{M}_\mathcal{P} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ holds because of Lemma 3.1(1).

$[(b) \Rightarrow (c)]$ Assume $\mathcal{P} \models_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ and recall Definition 3.3. Then $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ for every qc-interpretation $\mathcal{I}$ such that $\mathcal{I} \models_{\mathcal{S},\mathcal{D},\mathcal{C}} \mathcal{P}$. In particular, $\mathcal{M}_\mathcal{P} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$, since $\mathcal{M}_\mathcal{P} \models_{\mathcal{S},\mathcal{D},\mathcal{C}} \mathcal{P}$ was proved in Theorem 3.1.

$[(c) \Rightarrow (b)]$ Assume $\mathcal{M}_\mathcal{P} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$. In order to obtain $\mathcal{P} \models_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ we must prove:

$$(\star) \quad \mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi \text{ holds for any qc-interpretation } \mathcal{I} \text{ such that } \mathcal{I} \models_{\mathcal{S},\mathcal{D},\mathcal{C}} \mathcal{P} \ .$$

In the case that φ is a defined qc-atom, $\mathcal{M}_\mathcal{P} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ reduces to $\varphi \in \mathcal{M}_\mathcal{P}$, which implies $(\star)$ because $\mathcal{M}_\mathcal{P}$ is the least model of $\mathcal{P}$, as proved in Theorem 3.1. In the case that φ is not defined but $\mathcal{C}$-based, $(\star)$ follows form the fact that $\mathcal{I} \Vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ holds for any arbitrary qc-interpretation $\mathcal{I}$, as proved in Lemma 3.1(1). □

We close this subsection with a brief discussion on the relationship between the entailment relation $\succcurlyeq_{\mathcal{D},\mathcal{C}}$ used in this report and a different one that was proposed in (Caballero et al. 2008) and noted $\succcurlyeq_{\mathcal{S},\mathcal{D}}$. In contrast to $\succcurlyeq_{\mathcal{D},\mathcal{C}}$, the entailment $\succcurlyeq_{\mathcal{S},\mathcal{D}}$ depended on a given *similarity* relation $\mathcal{S}$. In the context of the SQCLP scheme, one could think of an entailment $\succcurlyeq_{\mathcal{S},\mathcal{D},\mathcal{C}}$ depending on $\mathcal{S}$ and defined in the following way: given two qc-atoms φ and φ', we could say that φ $(\mathcal{S},\mathcal{D},\mathcal{C})$-entails φ' (in symbols, $\varphi \succcurlyeq_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi'$) iff $\varphi : A\sharp d \Leftarrow \Pi$ and $\varphi' : A'\sharp d' \Leftarrow \Pi'$ such that there is some substitution θ satisfying $\mathcal{S}(A', A\theta) = \lambda \neq \mathbf{b}$, $d' \trianglelefteq \lambda$, $d' \trianglelefteq d$ and $\Pi' \models_\mathcal{C} \Pi\theta$.

However, $\succcurlyeq_{\mathcal{S},\mathcal{D},\mathcal{C}}$ would not work properly in the case that $\mathcal{S}$ is not transitive, as shown by the following simple example: think of a SQCLP$(\mathcal{S},\mathcal{U},\mathcal{R})$-program $\mathcal{P}$ including just a clause

$$R_1 : p_1 \xleftarrow{1.0}$$

and assume that $\mathcal{S}$ verifies $\mathcal{S}(p_1, p_2) = 0.9$, $\mathcal{S}(p_2, p_3) = 0.9$ and $\mathcal{S}(p_1, p_3) = 0.4$ where $p_1, p_2, p_3 \in DP^0$. Then, $\mathcal{P} \vdash_{\mathcal{S},\mathcal{U},\mathcal{R}} p_2\sharp 0.9 \Leftarrow \emptyset$ can be easily proved with the SQCHL rule **SQDA** and $p_2\sharp 0.9 \Leftarrow \emptyset \succcurlyeq_{\mathcal{S},\mathcal{U},\mathcal{R}} p_3\sharp 0.9 \Leftarrow \emptyset$ holds because of $\mathcal{S}(p_2, p_3) = 0.9$, but $\mathcal{P} \vdash_{\mathcal{S},\mathcal{U},\mathcal{R}} p_3\sharp 0.9 \Leftarrow \emptyset$ does not hold. Therefore, the Entailment Property for Programs (Lemma 3.1(2)) would fail if the entailment $\succcurlyeq_{\mathcal{S},\mathcal{D},\mathcal{C}}$ were adopted in place of $\succcurlyeq_{\mathcal{D},\mathcal{C}}$.

Since the Entailment Property for Programs is a very natural condition that must be preserved, we conclude that the entailment relation $\succcurlyeq_{\mathcal{D},\mathcal{C}}$ used in this report is the right choice in a framework where the underlaying proximity relation is not guaranteed to be a similarity.

3.2 Goals and their Solutions

In this brief subsection we present the syntax and declarative semantics of goals in the SQCLP scheme, and we define natural soundness and completeness properties

which are expected to be fulfilled by goal solving devices. These notions are intended as a useful tool to reason about the correctness of SQCLP implementations to be developed in the future.

In order to build goals for SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-programs, we assume a countably infinite set $\mathcal{W}ar$ of so-called *qualification variables* W, disjoint from $\mathcal{V}ar$ and $\mathcal{C}$'s signature Σ. Goals for a given program $\mathcal{P}$ have the form

$$G \ : \ A_1 \sharp W_1, \ \ldots, \ A_m \sharp W_m \ [] \ W_1 \trianglerighteq^? \beta_1, \ \ldots, \ W_m \trianglerighteq^? \beta_m$$

abbreviated as $(A_i \sharp W_i, \ W_i \trianglerighteq^? \beta_i)_{i=1\ldots m}$, where $A_i \sharp W_i$ $(1 \leq i \leq m)$ are atoms annotated with different qualification variables W_i; and $W_i \trianglerighteq^? \beta_i$ are so-called *threshold conditions* with $\beta_i \in (D \setminus \{\mathbf{b}\}) \uplus \{?\}$ $(1 \leq i \leq m)$. The notations ? and $\trianglerighteq^?$ have been already explained in Section 3.1.

In the sequel, the notation war(o) will denote the set of all qualification variables occurring in the syntactic object o. In particular, for a goal G as displayed above, war(G) denotes the set $\{W_i \mid 1 \leq i \leq m\}$. In the case $m = 1$ the goal is called *atomic*. The declarative semantics of goals is provided by their solutions, that are defined as follows:

Definition 3.5 (*Goal Solutions*)
Assume a given SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-program $\mathcal{P}$ and a goal G for the program $\mathcal{P}$ with the syntax displayed above. Then:

1. A *solution* for G is any triple $\langle \sigma, \mu, \Pi \rangle$ such that σ is a $\mathcal{C}$-substitution, μ : war(G) $\to D \setminus \{\mathbf{b}\}$, Π is a satisfiable and finite set of atomic $\mathcal{C}$-constraints and the following two conditions hold for all $i = 1 \ldots m$:

   (a) $W_i\mu = d_i \trianglerighteq^? \beta_i$ and
   (b) $\mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} A_i\sigma \sharp W_i\mu \Leftarrow \Pi$.

   The set of all solutions for G is noted $\mathrm{Sol}_{\mathcal{P}}(G)$. Note that solutions are *open* in the sense that the substitution σ is not required to be ground.
2. A solution $\langle \eta, \rho, \Pi \rangle$ for G is called *ground* iff $\Pi = \emptyset$ and $\eta \in \mathrm{Val}_{\mathcal{C}}$ is a variable valuation such that $A_i\eta$ is a ground atom for all $i = 1 \ldots m$. The set of all ground solutions for G is noted $\mathrm{GSol}_{\mathcal{P}}(G)$. Obviously, $\mathrm{GSol}_{\mathcal{P}}(G) \subseteq \mathrm{Sol}_{\mathcal{P}}(G)$.
3. A ground solution $\langle \eta, \rho, \emptyset \rangle \in \mathrm{GSol}_{\mathcal{P}}(G)$ is *subsumed* by $\langle \sigma, \mu, \Pi \rangle$ iff there is some $\nu \in \mathrm{Sol}_{\mathcal{C}}(\Pi)$ s.t. $\eta =_{\mathrm{var}(G)} \sigma\nu$ and $W_i\rho \trianglelefteqslant W_i\mu$ for $i = 1 \ldots m$. $\square$

Implicitly, the first item in the previous definition requires $A_i\sigma \sharp W_i\mu \Leftarrow \Pi$ to be observable qc-atoms in the sense of Definition 3.1, which is trivially true because $W_i\mu = d_i \in D \setminus \{\mathbf{b}\}$ and Π is satisfiable. In fact, Definition 3.1 was designed with the aim of using observable qc-atoms as observations of valid open solutions for atomic goals. The next example illustrates the definition:

Example 3.6 (*Solutions for an atomic goals*)
1. G : `goodWork(X)#W [] W |> (0.55,30)` is a goal for the program fragment $\mathcal{P}$ shown in Figure 1, and the arguments given near the beginning of Subsection 3.1 can be formalized to prove that $\langle \{\texttt{X} \mapsto \texttt{king\_liar}\}, \{\texttt{W} \mapsto \texttt{(0.6,5)}\}, \emptyset \rangle \in \mathrm{Sol}_{\mathcal{P}}(G)$.

2. As an additional example involving constraints, recall the SQCLP($\mathcal{S},\mathcal{U},\mathcal{R}$)-program $\mathcal{P}$ presented in Example 3.1. An atomic goal G for this program is $p'(c'(Y),Z)\sharp W \;[\!]\; W \geq^{?} 0.75$. Consider $\sigma = \{Z \mapsto c(X)\}$, $\mu = \{W \mapsto 0.8\}$ and $\Pi = \{cp_{>}(X, 1.0), op_{+}(A, A,\ X), op_{\times}(2.0, A, Y)\}$. Note that $0.8 \geq 0.75$ and $\mathcal{P} \vdash_{\mathcal{S},\mathcal{U},\mathcal{R}} p'(c'(Y),Z)\sigma\sharp W\mu \Leftarrow \Pi$, as we have seen in Example 3.5. Therefore, the requirements of Definition 3.5 are fulfilled, and $\langle \sigma, \mu, \Pi\rangle \in \mathrm{Sol}_{\mathcal{P}}(G)$. □

In practice, users of SQCLP languages will rely on some available *goal solving system* for computing goal solutions. The following definition specifies two important properties of goal solving systems:

Definition 3.6 (*Correct Goal Solving Systems*)
At a high abstraction level, a *goal solving system* for SQCLP($\mathcal{S},\mathcal{D},\mathcal{C}$) can be thought as a device that takes a program $\mathcal{P}$ and a goal G as input and yields various triples $\langle \sigma, \mu, \Pi\rangle$, called *computed answers*, as outputs. Such a goal solving system is called:

1. *Sound* iff every computed answer is a solution $\langle \sigma, \mu, \Pi\rangle \in \mathrm{Sol}_{\mathcal{P}}(G)$.
2. *Weakly complete* iff every ground solution $\langle \eta, \rho, \emptyset\rangle \in \mathrm{GSol}_{\mathcal{P}}(G)$ is subssumed by some computed answer.
3. *Correct* iff it is both sound and weakly complete. □

Every goal solving system for a SQCLP instance should be sound and ideally also weakly complete. Implementing such systems is one of the major lines of future research mentioned in the Conclusions of this report.

4 Conclusions

We have extended the classical CLP scheme to a new scheme SQCLP whose instances SQCLP($\mathcal{S},\mathcal{D},\mathcal{C}$) are parameterized by a proximity relation $\mathcal{S}$, a qualification domain $\mathcal{D}$ and a constraint domain $\mathcal{C}$. In addition to the known features of CLP programming, the new scheme offers extra facilities for dealing with expert knowledge representation and flexible query answering. Inspired by the observable CLP semantics in (Gabbrielli and Levi 1991; Gabbrielli et al. 1995), we have presented a declarative semantics for SQCLP that provides fixpoint and proof-theoretical characterizations of least program models as well as an implementation-independent notion of goal solutions.

SQCLP is a quite general scheme. Different partial instantiations of its three parameters lead to more particular schemes, most of which can be placed in close correspondence to previous proposals. The items below present seven particularizations, along with some comments which make use of the notions *threshold-free*, *attenuation-free* and *constraint-free* which have been explained at the beginning of Section 3.1.

1. By definition, QCLP has instances QCLP($\mathcal{D},\mathcal{C}$) $=_{\mathrm{def}}$ SQCLP($\mathcal{S}_{\mathrm{id}},\mathcal{D},\mathcal{C}$), where $\mathcal{S}_{\mathrm{id}}$ is the *identity* proximity relation. The *quantitative* CLP scheme proposed in (Riezler 1998) can be understood as a further particularization of QCLP that works with threshold-free QCLP($\mathcal{U},\mathcal{C}$) programs, where $\mathcal{U}$ is the qualification domain of uncertainty values (see Subsection 2.2.2).

2. By definition, SQLP has instances $\mathrm{SQLP}(\mathcal{S},\mathcal{D}) =_{\mathrm{def}} \mathrm{SQCLP}(\mathcal{S},\mathcal{D},\mathcal{R})$, where $\mathcal{R}$ is the real constraint domain (see Subsection 2.1.2). The scheme with the same name originally proposed in (Caballero et al. 2008) can be understood as a restricted form of the present formulation; it worked with threshold-free and constraint-free $\mathrm{SQLP}(\mathcal{S},\mathcal{D})$ programs and it restricted the choice of the $\mathcal{S}$ parameter to transitive proximity (i.e. similarity) relations.
3. By definition, SCLP[3] has instances $\mathrm{SCLP}(\mathcal{S},\mathcal{C}) =_{\mathrm{def}} \mathrm{SQCLP}(\mathcal{S},\mathcal{B},\mathcal{C})$, where $\mathcal{B}$ is the qualification domain of classical boolean values (see Subsection 2.2.1). Due to the fixed parameter choice $\mathcal{D} = \mathcal{B}$, both attenuation values and threshold values become useless, and each choice of $\mathcal{S}$ must necessarily represent a crisp reflexive and symmetric relation. Therefore, this new scheme is not so interesting from the viewpoint of uncertain and qualified reasoning.
4. By definition, QLP has instances $\mathrm{QLP}(\mathcal{D}) =_{\mathrm{def}} \mathrm{SQCLP}(\mathcal{S}_{\mathrm{id}},\mathcal{D},\mathcal{R})$. The scheme with the same name originally proposed in (Rodríguez-Artalejo and Romero-Díaz 2008b) can be understood as a restricted form of the present formulation; it worked with threshold-free and constraint-free $\mathrm{QLP}(\mathcal{D})$ programs.
5. By definition, SLP has instances $\mathrm{SLP}(\mathcal{S}) =_{\mathrm{def}} \mathrm{SQCLP}(\mathcal{S},\mathcal{U},\mathcal{R})$. The pure fragment of Bousi~Prolog (Julián-Iranzo and Rubio-Manzano 2009a) can be understood as a restricted form of SLP in the present formulation; it works with threshold-free, attenuation-free and constraint-free $\mathrm{SLP}(\mathcal{S})$ programs. Moreover, restricting the choice of $\mathcal{S}$ to similarity relations leads to SLP in the sense of (Sessa 2002) and related papers.
6. The CLP scheme can be defined by instances $\mathrm{CLP}(\mathcal{C}) =_{\mathrm{def}} \mathrm{SQCLP}(\mathcal{S}_{\mathrm{id}},\mathcal{B},\mathcal{C})$. Both attenuation values and threshold values are useless in CLP programs, due to the fixed parameter choice $\mathcal{D} = \mathcal{B}$.
7. Finally, the pure LP paradigm can be defined as $\mathrm{LP} =_{\mathrm{def}} \mathrm{SQCLP}(\mathcal{S}_{\mathrm{id}},\mathcal{B},\mathcal{H})$, where $\mathcal{H}$ is the *Herbrand* constraint domain. Again, attenuation values and threshold values are useless in LP due to the fixed parameter choice $\mathcal{D} = \mathcal{B}$.

In all the previous items, the schemes obtained by partial instantiation inherit the declarative semantics from SQCLP, using sets of observables of the form $A\sharp d \Leftarrow \Pi$ as interpretations. A similar semantic approach were used in our previous papers (Rodríguez-Artalejo and Romero-Díaz 2008b; Caballero et al. 2008), except that Π and equations were absent due to the lack of CLP features. The other related works discussed in the Introduction view program interpretations as mappings $\mathcal{I}$ from the ground Herbrand base into some set of lattice elements (the real interval $[0,1]$ in many cases), as already discussed in the explanations following Definition 3.2.

As seen in Subsection 3.2, SQCLP's semantics enables a declarative characterization of valid goal solutions. This fact is relevant for modeling the expected behavior of goal solving devices and reasoning about their correctness. Moreover, the relations $\approx_{\lambda,\Pi}$ introduced for the first time in the present paper (see Definition 2.16) allow to specify the semantic role of $\mathcal{S}$ in a constraint-based framework, with less technical overhead than in previous related approaches.

[3] Not to be confused with SCLP in the sense of (Bistarelli et al. 2001), discussed below.

A related work not mentioned in items 1–7 above is the semiring-based CLP of (Bistarelli et al. 2001), a scheme with instances SCLP(S) parameterized by a semiring $\mathrm{S} = \langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$ whose elements are used to represent consistency levels in soft constraint solving. The semirings used in this approach can be equipped with a lattice structure whose *lub* operation is always $+$, but whose *glb* operation may be different from $\times$. On the other hand, our qualification domains are defined as lattices with an additional attenuation operation $\circ$. It turns out that the kind of semirings used in SCLP(S) correspond to qualification domains only in some cases. Moreover, $\times$ is used in SCLP(S) to interpret logical conjunction in clause bodies and goals, while the *glb* operation is used in SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$) for the same purpose. For this reason, even if $\mathcal{D}$ is "equivalent" to S, SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$) cannot be naturally used to express SCLP(S) in the case that $\times$ is not the *glb*. Assuming that $\mathcal{D}$ is "equivalent" to S and that $\times$ behaves as the *glb* in S, program clauses in SCLP(S) can be viewed as a particular case of program clauses in SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$) which use an attenuation factor different from $\mathbf{t}$ only for facts. Other relevant differences between SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$) and SCLP(S) can be explained by comparing the parameters. As said before $\mathcal{D}$ may be "equivalent" to S in some cases, but $\mathcal{S}$ is absent and $\mathcal{C}$ is not made explicit in SCLP(S). Seemingly, the intended use of SCLP(S) is related to finite domain constraints and no parametrically given constraint domain is provided.

In the future we plan to implement some SQCLP instances by extending the semantically correct program transformation techniques from (Caballero et al. 2008), and to investigate applications which can profit from flexible query answering. Other interesting lines of future work include: a) extension of the qualified SLD resolution presented in (Rodríguez-Artalejo and Romero-Díaz 2008b) to a SQCLP goal solving procedure able to work with constraints and a proximity relation; and b) extension of the QCFLP scheme in (Caballero et al. 2009) to work with a proximity relation and higher-order functions.

Acknowledgements

This report is a widely extended version of (Rodríguez-Artalejo and Romero-Díaz 2010). The authors are thankful to the anonymous referees of (Rodríguez-Artalejo and Romero-Díaz 2010) for constructive remarks and suggestions which helped to improve the presentation. They are also thankful to Rafael Caballero for useful discussions on the report's topics and to Jesús Almendros for pointing to bibliographic references in the area of flexible query answering.

References

Apt, K. R. 1990. Logic programming. In *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed. Vol. B: Formal Models and Semantics. Elsevier and The MIT Press, 493–574.

Apt, K. R. and Gabbrielli, M. 1994. Declarative interpretations reconsidered. In *Proceedings of the 11th International Conference on Logic Programming (ICLP'94)*, P. van Hentenryck, Ed. The MIT Press, 74–89.

Apt, K. R. and van Emden, M. H. 1982. Contributions to the theory of logic programming. *Journal of the Association for Computing Machinery (JACM) 29,* 3, 841–862.

Arcelli, F. and Formato, F. 1999. Likelog: a logic programming language for flexible data retrieval. In *Proceedings of the 1999 ACM Symposium on Applied computing (SAC'99)*. ACM Press, New York, NY, USA, 260–267.

Arenas, P., Fernández, A. J., Gil, A., López-Fraguas, F. J., Rodríguez-Artalejo, M., and Sáenz-Pérez, F. 2007. $\mathcal{TOY}$, a multiparadigm declarative language (version 2.3.1). In R. Caballero and J. Sánchez, editors, *User Manual*, available at `http://toy.sourceforge.net`.

Baader, F. and Nipkow, T. 1998. *Term Rewriting and All That.* Cambridge University Press.

Bistarelli, S., Montanari, U., and Rossi, F. 2001. Semiring-based constraint logic programming: Syntax and semantics. *ACM Transactions on Programming Languages and Systems 3,* 1 (January), 1–29.

Bossi, A., Gabbrielli, M., Levi, G., and Martelli, M. 1994. The s-semantics approach: Theory and applications. *Journal of Logic Programming 19/20*, 149–197.

Caballero, R., Rodríguez-Artalejo, M., and Romero-Díaz, C. A. 2008. Similarity-based reasoning in qualified logic programming. In *PPDP '08: Proceedings of the 10th international ACM SIGPLAN conference on Principles and Practice of Declarative Programming.* ACM, Valencia, Spain, 185–194.

Caballero, R., Rodríguez-Artalejo, M., and Romero-Díaz, C. A. 2009. Qualified computations in functional logic programming. In *Logic Programming (ICLP'09)*, P. Hill and D. Warren, Eds. LNCS, vol. 5649. Springer-Verlag Berlin Heidelberg, Pasadena, CA, USA, 449–463.

Clark, K. L. 1979. Predicate logic as a computational formalism (res. report doc 79/59). Tech. rep., Imperial College, Dept. of Computing, London.

Dubois, D. and Prade, H. 1980. *Fuzzy Sets and Systems: Theory and Applications.* Academic Press, New York, NY, USA.

Estévez-Martín, S., Hortalá-González, T., Rodríguez-Artalejo, M., del Vado Vírseda, R., Sáenz-Pérez, F., and Fernández, A. J. 2009. On the cooperation of the constraint domains $\mathcal{H}$, $\mathcal{R}$ and $\mathcal{FD}$ in $cflp$. *Theory and Practice of Logic Programming 9,* 4, 415–527.

Falaschi, M., Levi, G., Martelli, M., and Palamidessi, C. 1993. A model-theoretic reconstruction of the operational semantics of logic programs. *Information and Computation 102,* 1, 86–113.

Falaschi, M., Levi, G., Palamidessi, C., and Martelli, M. 1989. Declarative modeling of the operational behavior of logic languages. *Theoretical Computer Science 69,* 3 (December), 289–318.

Freuder, E. C. and Wallace, R. J. 1992. Partial constraint satisfaction. *Artificial Intelligence 58,* 1–3, 21–70.

Gabbrielli, M., Dore, G. M., and Levi, G. 1995. Observable semantics for constraint logic programs. *Journal of Logic and Computation 5,* 2, 133–171.

Gabbrielli, M. and Levi, G. 1991. Modeling answer constraints in constraint logic programs. In *Proceedings of the 8th International Conference on Logic Programming (ICLP'91).* The MIT Press, 238–252.

Georget, Y. and Codognet, P. 1998. Compiling semiring-based constraints with CLP(FD,S). In *Proceedings of the 4th International Conference on Principles and Practice of Constraint Programming.* LNCS, vol. 1520. Springer-Verlag, 205–219.

Guadarrama, S., Muñoz, S., and Vaucheret, C. 2004. Fuzzy prolog: A new approach using soft constraint propagation. *Fuzzy Sets and Systems 144,* 1, 127–150.

Hájek, P. 1998. *Metamathematics of Fuzzy Logic.* Dordrecht: Kluwer.

Hanus, Ed., M. Curry: An integrated functional logic language (vers. 0.8.2, 2006); `http://www.curry-language.org`.

Höhfeld, M. and Smolka, G. 1988. Definite relations over constraint languages. Tech. Rep. LILOG Report 53, IBM Deutschland.

Jaffar, J. and Lassez, J. L. 1987. Constraint logic programming. In *Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of Programming Languages (POPL'87)*. ACM New York, NY, USA, Munich, West Germany, 111–119.

Jaffar, J. and Maher, M. 1994. Constraint logic programming: a survey. *Journal of Logic Programming 19&20*, 503–581.

Jaffar, J., Maher, M., Marriott, K., and Stuckey, P. J. 1998. Semantics of constraints logic programs. *Journal of Logic Programming 37,* 1-3, 1–46.

Jaffar, J., Michaylov, S., Stuckey, P. J., and Yap, R. H. C. 1992. The CLP(R) language and system. *ACM Transactions on Programming Languages and Systems 14(3)*, 339–395.

Julián-Iranzo, P., Rubio, C., and Gallardo, J. 2009. Bousi∼Prolog: a prolog extension language for flexible query answering. In *Proceedings of the Eighth Spanish Conference on Programming and Computer Languages (PROLE 2008)*, J. M. Almendros-Jiménez, Ed. ENTCS, vol. 248. Elsevier, Gijón, Spain, 131–147.

Julián-Iranzo, P. and Rubio-Manzano, C. 2009a. A declarative semantics for Bousi∼Prolog. In *PPDP'09: Proceedings of the 11th ACM SIGPLAN conference on Principles and practice of declarative programming*. ACM, Coimbra, Portugal, 149–160.

Julián-Iranzo, P. and Rubio-Manzano, C. 2009b. A similarity-based WAM for Bousi∼Prolog. In *Bio-Inspired Systems: Computational and Ambient Intelligence (IWANN 2009)*. LNCS, vol. 5517. Springer Berlin / Heidelberg, Salamanca, Spain, 245–252.

Kifer, M. and Subrahmanian, V. S. 1992. Theory of generalized annotated logic programs and their applications. *Journal of Logic Programming 12,* 3&4, 335–367.

Krajči, S., Lencses, R., and Vojtáš, P. 2004. A comparison of fuzzy and annotated logic programming. *Fuzzy Sets and Systems 144*, 173–192.

Lloyd, J. W. 1987. *Foundations of Logic Programming, Second Edition.* Springer.

Loia, V., Senatore, S., and Sessa, M. I. 2004. Similarity-based SLD resolution and its role for web knowledge discovery. *Fuzzy Sets and Systems 144,* 1, 151–171.

López-Fraguas, F. J., Rodríguez-Artalejo, M., and del Vado-Vírseda, R. 2007. A new generic scheme for functional logic programming with constraints. *Journal of Higher-Order and Symbolic Computation 20,* 1&2, 73–122.

Lucio, P., Orejas, F., Pasarella, E., and Pino, E. 2008. A functorial framework for constraint normal logic programming. *Applied Categoriacal Structures 16,* 3, 421–450.

Medina, J., Ojeda-Aciego, M., and Vojtáš, P. 2001a. Multi-adjoint logic programming with continuous semantics. In *Logic Programming and Non-Monotonic Reasoning (LPNMR'01)*, T. Eiter, W. Faber, and M. Truszczyinski, Eds. LNAI, vol. 2173. Springer-Verlag, 351–364.

Medina, J., Ojeda-Aciego, M., and Vojtáš, P. 2001b. A procedural semantics for multi-adjoint logic programming. In *Progress in Artificial Intelligence (EPIA'01)*, P. Brazdil and A. Jorge, Eds. LNAI, vol. 2258. Springer-Verlag, 290–297.

Moreno, G. and Pascual, V. 2007. Formal properties of needed narrowing with similarity relations. *Electronic Notes in Theoretical Computer Science 188*, 21–35.

Ng, R. T. and Subrahmanian, V. S. 1992. Probabilistic logic programming. *Information and Computation 101,* 2, 150–201.

Riezler, S. 1996. Quantitative constraint logic programming for weighted grammar applications. In *Proceedings of the Logical Aspects of Computational Linguistics (LACL'96)*, C. Retoré, Ed. LNCS, vol. 1328. Springer-Verlag, 346–365.

Riezler, S. 1998. Probabilistic constraint logic programming. Ph.D. thesis, Neuphilologischen Fakultät del Universität Tübingen.

Rodríguez-Artalejo, M. and Romero-Díaz, C. A. 2008a. A generic scheme for qualified logic programming. Tech. Rep. SIC-1-08 (CoRR abs/1008.3863), Universidad Complutense, Departamento de Sistemas Informáticos y Computación, Madrid, Spain.

Rodríguez-Artalejo, M. and Romero-Díaz, C. A. 2008b. Quantitative logic programming revisited. In *Functional and Logic Programming (FLOPS'08)*, J. Garrigue and M. Hermenegildo, Eds. LNCS, vol. 4989. Springer-Verlag, Ise, Japan, 272–288.

Rodríguez-Artalejo, M. and Romero-Díaz, C. A. 2009. Qualified logic programming with bivalued predicates. In *Proceedings of the Eighth Spanish Conference on Programming and Computer Languages (PROLE 2008)*, J. M. Almendros-Jiménez, Ed. ENTCS, vol. 248. Elsevier, Gijón, Spain, 67–82.

Rodríguez-Artalejo, M. and Romero-Díaz, C. A. 2010. A declarative semantics for CLP with qualification and proximity. *Theory and Practice of Logic Programming, 26th Int'l. Conference on Logic Programming (ICLP'10) Special Issue 10,* 4–6, 627–642.

Saraswat, V. A. 1992. The category of constraint systems is cartesian-closed. In *Proceedings of the Seventh Annual IEEE Symposium on Logic in Computer Science (LICS '92)*. 341–345.

Sessa, M. I. 2002. Approximate reasoning by similarity-based SLD resolution. *Theoretical Computer Science 275,* 1-2, 389–426.

Shenoi, S. and Melton, A. 1999. Proximity relations in the fuzzy relational database model. *Fuzzy Sets and Systems 100,* supl., 51–62.

Tarski, A. 1955. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics 5,* 2, 285–309.

van Emden, M. H. 1986. Quantitative deduction and its fixpoint theory. *Journal of Logic Programming 3,* 1, 37–53.

van Emden, M. H. and Kowalski, R. A. 1976. The semantics of predicate logic as a programming language. *Journal of the Association for Computing Machinery (JACM) 23,* 4, 733–742.

Vaucheret, C., Guadarrama, S., and Muñoz, S. 2002. Fuzzy prolog: A simple general implementation using CLP($\mathcal{R}$). In *Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'02)*, M. Baaz and A. Voronkov, Eds. LNCS, vol. 2514. Springer Berlin / Heidelberg, Tbilisi, Georgia, 450–463.

Vojtáš, P. 2001. Fuzzy logic programming. *Fuzzy Sets and Systems 124*, 361–370.

Zadeh, L. A. 1965. Fuzzy sets. *Information and Control 8,* 3, 338–353.

Zadeh, L. A. 1971. Similarity relations and fuzzy orderings. *Information Sciences 3,* 2, 177–200.

A Transformation-based Implementation for CLP with Qualification and Proximity ∗

Preliminary Version (Technical Report SIC-4-10)

R. CABALLERO, M. RODRÍGUEZ-ARTALEJO and C. A. ROMERO-DÍAZ

Departamento de Sistemas Informáticos y Computación, Universidad Complutense
Facultad de Informática, 28040 Madrid, Spain
(*e-mail:* {rafa,mario}@sip.ucm.es, cromdia@fdi.ucm.es)

Abstract

Uncertainty in logic programming has been widely investigated in the last decades, leading to multiple extensions of the classical LP paradigm. However, few of these are designed as extensions of the well-established and powerful CLP scheme for Constraint Logic Programming. In a previous work we have proposed the SQCLP (*proximity-based qualified constraint logic programming*) scheme as a quite expressive extension of CLP with support for qualification values and proximity relations as generalizations of uncertainty values and similarity relations, respectively. In this paper we provide a transformation technique for transforming SQCLP programs and goals into semantically equivalent CLP programs and goals, and a practical Prolog-based implementation of some particularly useful instances of the SQCLP scheme. We also illustrate, by showing some simple—and working—examples, how the prototype can be effectively used as a tool for solving problems where qualification values and proximity relations play a key role. Intended use of SQCLP includes flexible information retrieval applications.

KEYWORDS: Constraint Logic Programming, Program Transformation, Qualification Domains and Values, Similarity and Proximity Relations, Flexible Information Retrieval.

1 Introduction

Many extensions of LP (*logic programming*) to deal with uncertain knowledge and uncertainty have been proposed in the last decades. These extensions have been proposed from different and somewhat unrelated perspectives, leading to multiple approaches in the way of using uncertain knowledge and understanding uncertainty.

A recent work by us (Rodríguez-Artalejo and Romero-Díaz 2010a) focuses on the declarative semantics of a new proposal for an extension of the CLP scheme supporting qualification values and proximity relations. More specifically, this work defines a new generic scheme SQCLP (*proximity-based qualified constraint logic programming*) whose instances SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$) are parameterized by a proximity

∗ This work has been partially supported by the Spanish projects STAMP (TIN2008-06622-C03-01), PROMETIDOS–CM (S2009TIC-1465) and GPD–UCM (UCM–BSCH–GR58/08-910502).

relation $\mathcal{S}$, a qualification domain $\mathcal{D}$ and a constraint domain $\mathcal{C}$. The current paper is intended as a continuation of (Rodríguez-Artalejo and Romero-Díaz 2010a) with the aim of providing a semantically correct program transformation technique that allows us to implement a sound and complete implementation of some useful instances of SQCLP on top of existing CLP systems like *SICStus Prolog* or *SWI-Prolog*. In the introductory section of (Rodríguez-Artalejo and Romero-Díaz 2010a) we have already summarized some related approaches of SQCLP with a special emphasis on their declarative semantics and their main semantic differences with SQCLP. In the next paragraphs we present a similar overview but, this time, putting the emphasis on the goal resolution procedures and system implementation techniques, when available.

Within the extensions of LP using annotations in program clauses we can find the seminal proposal of *quantitative logic programming* by (van Emden 1986) that inspired later works such as the GAP (*generalized annotated programs*) framework by (Kifer and Subrahmanian 1992) and the QLP (*qualified logic programming*) scheme by us (Rodríguez-Artalejo and Romero-Díaz 2008). In the proposal of van Emden, one can find a primitive goal solving procedure based in and/or trees (these are similar to the alpha-beta trees used in game theory), used to prune the search space when proving some specific ground atom for some certainty value in the real interval $[0, 1]$. In the case of GAP, the goal solving procedure uses constrained SLD resolution in conjunction with a—costly—computation of so-called *reductants* between variants of program clauses. In contrast, QLP goal solving uses a more efficient resolution procedure called SLD($\mathcal{D}$) resolution, implemented by means of real domain constraints, used to compute the qualification value of the head atom based on the attenuation factor of the program clause and the previously computed qualification values of the body atoms. Admittedly, the gain in efficiency of SLD($\mathcal{D}$) w.r.t. GAP's goal solving procedure is possible because QLP focuses on a more specialized class of annotated programs. While in all these three approaches there are some results of soundness and completeness, the results for the QLP scheme are the stronger ones (again, thanks to its also more focused scope w.r.t. GAP).

From a different viewpoint, extensions of LP supporting uncertainty can be roughly classified into two major lines: approaches based in fuzzy logic (Zadeh 1965; Hájek 1998) and approaches based in similarity relations. Historically, Fuzzy LP languages were motivated by expert knowledge representation applications. Early Fuzzy LP languages implementing the resolution principle introduced in (Lee 1972) include Prolog-Elf (Ishizuka and Kanai 1985), Fril Prolog (Baldwin et al. 1995) and F-Prolog (Li and Liu 1990). More recent approaches such as the Fuzzy LP languages in (Vojtáš 2001; Guadarrama et al. 2004) and Multi-Adjoint LP (MALP for short) in the sense of (Medina et al. 2001a) use clause annotations and a fuzzy interpretation of the connectives and aggregation operators occurring in program clauses and goals. The Fuzzy Prolog system proposed in (Guadarrama et al. 2004) is implemented by means of real constrains on top of a CLP($\mathcal{R}$) system, using a syntactic expansion of the source code during the Prolog compilation. A complete procedural semantics for MALP using reductants has been presented in (Medina

et al. 2001b). A method for translating a MALP like program into standard Prolog has been described in (Julián et al. 2009).

The second line of research mentioned in the previous paragraph was motivated by applications in the field of flexible query answering. Classical LP is extended to Similarity-based LP (SLP for short), leading to languages which keep the classical syntax of LP clauses but use a similarity relation over a set of symbols S to allow "flexible" unification of syntactically different symbols with a certain approximation degree. Similarity relations over a given set S have been defined in (Zadeh 1971; Sessa 2002) and related literature as fuzzy relations represented by mappings $\mathcal{S} : S \times S \to [0, 1]$ which satisfy reflexivity, symmetry and transitivity axioms analogous to those required for classical equivalence relations. Resolution with flexible unification can be used as a sound and complete goal solving procedure for SLP languages as shown e.g. in (Sessa 2002). SLP languages include *Likelog* (Arcelli and Formato 1999; Arcelli Fontana 2002) and more recently *SiLog* (Loia et al. 2004), which has been implemented by means of an extended Prolog interpreter and proposed as a useful tool for web knowledge discovery.

In the last years, the SLP approach has been extended in various ways. The SQLP (*similarity-based qualified logic programming*) scheme proposed in (Caballero et al. 2008) extended SLP by allowing program clause annotations in QLP style and generalizing similarity relations to mappings $\mathcal{S} : S \times S \to D$ taking values in a qualification domain not necessarily identical to the real interval $[0, 1]$. As implementation technique for SQLP, (Caballero et al. 2008) proposed a semantically correct program transformation into QLP, whose goal solving procedure has been described above. Other related works on transformation-based implementations of SLP languages include (Sessa 2001; Medina et al. 2004). More recently, the SLP approach has been generalized to work with *proximity relations* in the sense of (Dubois and Prade 1980) represented by mappings $\mathcal{S} : S \times S \to [0, 1]$ which satisfy reflexivity and symmetry axioms but do not always satisfy transitivity. SLP like languages using proximity relations include Bousi∼Prolog (Julián-Iranzo and Rubio-Manzano 2009a) and the SQCLP scheme (Rodríguez-Artalejo and Romero-Díaz 2010a). Two prototype implementations of Bousi∼Prolog are available: a low-level implementation (Julián-Iranzo and Rubio-Manzano 2009b) based on an adaptation of the classical WAM (called *Similarity WAM*) implemented in Java and able to execute a Prolog program in the context of a similarity relation defined on the first order alphabet induced by that program; and a high-level implementation (Julián-Iranzo et al. 2009) done on top of *SWI-Prolog* by means of a program transformation from Bousi∼Prolog programs into a so-called *Translated BPL code* than can be executed according to the weak SLD resolution principle by a meta-interpreter.

Let us now refer to approaches related to constraint solving and CLP. An analogy of proximity relations in the context of partial constraint satisfaction can be found in (Freuder and Wallace 1992), where several metrics are proposed to measure the proximity between the solution sets of two different constraint satisfaction problems. Moreover, some extensions of LP supporting uncertain reasoning use constraint solving as implementation technique, as discussed in the previous paragraphs. However, we are only aware of three approaches which have been conceived

as extensions of the classical CLP scheme proposed for the first time in (Jaffar and Lassez 1987). These three approaches are: (Riezler 1998) that extends the formulation of CLP by (Höhfeld and Smolka 1988) with quantitative LP in the sense of (van Emden 1986) and adapts van Emden's idea of and/or trees to obtain a goal resolution procedure; (Bistarelli et al. 2001) that proposes a semiring-based approach to CLP, where constraints are solved in a soft way with levels of consistency represented by values of the semiring, and is implemented with `clp(FD,S)` for a particular class of semirings which enable to use local consistency algorithms, as described in (Georget and Codognet 1998); and the SQCLP scheme proposed in our previous work (Rodríguez-Artalejo and Romero-Díaz 2010a), which was designed as a common extension of SQLP and CLP.

As we have already said at the beginning of this introduction, this paper deals with transformation-based implementations of the SQCLP scheme. Our main results include: a) a transformation technique for transforming SQCLP programs into semantically equivalent CLP programs via two specific program transformations named $\text{elim}_{\mathcal{S}}$ and $\text{elim}_{\mathcal{D}}$; and b) and a practical Prolog-based implementation which relies on the aforementioned program transformations and supports several useful SQCLP instances. As far as we know, no previous work has dealt with the implementation of extended LP languages for uncertain reasoning which are able to support clause annotations, proximity relations and CLP style programming. In particular, our previous paper (Caballero et al. 2008) only presented a transformation analogous to $\text{elim}_{\mathcal{S}}$ for a programming scheme less expressive than SQCLP, which supported neither non-transitive proximity relations nor CLP programming. Moreover, the transformation-based implementation reported in (Caballero et al. 2008) was not implemented in a system.

The reader is assumed to be familiar with the semantic foundations of LP (Lloyd 1987; Apt 1990) and CLP (Jaffar and Lassez 1987; Jaffar et al. 1998). The rest of the paper is structured as follows: Section 2 presents a brief overview of the semantics of the SQCLP scheme, focusing on the essential notions needed to understand the following sections and concluding with an abstract discussion of goal solving procedures for SQCLP. Section 3 briefly discusses two specializations of SQCLP, namely QCLP and CLP, which are used as the targets of the program transformations $\text{elim}_{\mathcal{S}}$ and $\text{elim}_{\mathcal{D}}$, respectively. Section 4 presents these two program transformations along with mathematical results which prove their semantic correctness, relying on the declarative semantics of the SQCLP, QCLP and CLP schemes. Section 5 presents a Prolog-based prototype system which relies on the transformations proposed in the previous section and implements several useful SQCLP instances. Finally, Section 6 summarizes conclusions and points to some lines of planned future research.

2 The Scheme SQCLP and its Declarative Semantics

We present in this section a short overview of the declarative semantics of the SQCLP scheme originally presented in (Rodríguez-Artalejo and Romero-Díaz 2010a), focusing on the essential notions needed to understand the following sections. In-

terested readers are referred to (Rodríguez-Artalejo and Romero-Díaz 2010a) and its extended version (Rodríguez-Artalejo and Romero-Díaz 2010b) for a full-fledged exposition of SQCLP semantics and a discussion of various extended LP languages for uncertain reasoning which can be obtained as specializations and instances of SQCLP. Some technical notions and results from (Rodríguez-Artalejo and Romero-Díaz 2010b) will be cited along this paper when needed to support mathematical proofs.

Constraint domains $\mathcal{C}$, sets of constraints Π and their solutions, as well as terms, atoms and substitutions over a given $\mathcal{C}$ are well known notions underlying the CLP scheme. The reader is referred (Rodríguez-Artalejo and Romero-Díaz 2010b) for a relational formalization of constraint domains and some examples, including the real constraint domain $\mathcal{R}$. We assume the following classification of atomic $\mathcal{C}$-constraints: defined atomic constraints $p(\bar{t}_n)$, where p is a program-defined predicate symbol; primitive constraints $r(\bar{t}_n)$ where r is a $\mathcal{C}$-specific primitive predicate symbol; and equations $t == s$.

We use $\mathrm{Con}_\mathcal{C}$ as a notation for the set of all $\mathcal{C}$-constraints and κ as a notation for an atomic primitive constraint. Constraints are interpreted by means of *$\mathcal{C}$-valuations* $\eta \in \mathrm{Val}_\mathcal{C}$, which are ground substitutions. The set $\mathrm{Sol}_\mathcal{C}(\Pi)$ of solutions of $\Pi \subseteq \mathrm{Con}_\mathcal{C}$ includes all the valuations η such that $\Pi\eta$ is true when interpreted in $\mathcal{C}$. $\Pi \subseteq \mathrm{Con}_\mathcal{C}$ is called *satisfiable* if $\mathrm{Sol}_\mathcal{C}(\Pi) \neq \emptyset$ and *unsatisfiable* otherwise. $\pi \in \mathrm{Con}_\mathcal{C}$ *is entailed* by $\Pi \subseteq \mathrm{Con}_\mathcal{C}$ (noted $\Pi \models_\mathcal{C} \pi$) iff $\mathrm{Sol}_\mathcal{C}(\Pi) \subseteq \mathrm{Sol}_\mathcal{C}(\pi)$.

Qualification domains were first introduced in (Rodríguez-Artalejo and Romero-Díaz 2008) with the aim of providing elements, called qualification values, which can be attached to computed answers. They are defined as structures $\mathcal{D} = \langle D, \preccurlyeq, \mathbf{b}, \mathbf{t}, \circ \rangle$ verifying the following requirements:

1. $\langle D, \preccurlyeq, \mathbf{b}, \mathbf{t} \rangle$ is a lattice with extreme points $\mathbf{b}$ (called *infimum* or *bottom* element) and $\mathbf{t}$ (called *maximum* or *top* element) w.r.t. the partial ordering $\preccurlyeq$ (called *qualification ordering*). For given elements $d, e \in D$, we write $d \sqcap e$ for the *greatest lower bound* (*glb*) of d and e, and $d \sqcup e$ for the *least upper bound* (*lub*) of d and e. We also write $d \lhd e$ as abbreviation for $d \preccurlyeq e \wedge d \neq e$.
2. $\circ : D \times D \to D$, called *attenuation operation*, verifies the following axioms:
   (a) $\circ$ is associative, commutative and monotonic w.r.t. $\preccurlyeq$.
   (b) $\forall d \in D : d \circ \mathbf{t} = d$ and $d \circ \mathbf{b} = \mathbf{b}$.
   (c) $\forall d, e \in D : d \circ e \preccurlyeq e$ and even $\mathbf{b} \neq d \circ e \preccurlyeq e$ if $d, e \in D \setminus \{\mathbf{b}\}$.
   (d) $\forall d, e_1, e_2 \in D : d \circ (e_1 \sqcap e_2) = (d \circ e_1) \sqcap (d \circ e_2)$.

For any $S = \{e_1, e_2, \ldots, e_n\} \subseteq D$, the *glb* (also called *infimum* of S) exists and can be computed as $\bigsqcap S = e_1 \sqcap e_2 \sqcap \cdots \sqcap e_n$ (which reduces to $\mathbf{t}$ in the case $n = 0$). The dual claim concerning *lub*s is also true. As an easy consequence of the axioms, one gets the identity $d \circ \bigsqcap S = \bigsqcap \{d \circ e \mid e \in S\}$.

Technical details, explanations and examples can be found in (Rodríguez-Artalejo and Romero-Díaz 2010b), including: the qualification domain $\mathcal{B}$ of classical boolean values, the qualification domain $\mathcal{U}$ of uncertainty values, the qualification domain $\mathcal{W}$ of weight values, and other qualification domains built from these by means of

the strict cartesian product operation $\otimes$. The following definition is borrowed from (Rodríguez-Artalejo and Romero-Díaz 2010a):

Definition 2.1 (*Expressing $\mathcal{D}$ in $\mathcal{C}$*)
A qualification domain $\mathcal{D}$ is expressible in a constraint domain $\mathcal{C}$ if there is an injective embedding mapping $\imath : D \setminus \{\mathbf{b}\} \to C$ and moreover:

1. There is a $\mathcal{C}$-constraint $\mathsf{qVal}(X)$ such that $\mathrm{Sol}_{\mathcal{C}}(\mathsf{qVal}(X))$ is the set of all $\eta \in \mathrm{Val}_{\mathcal{C}}$ verifying $\eta(X) \in ran(\imath)$.
2. There is a $\mathcal{C}$-constraint $\mathsf{qBound}(X, Y, Z)$ encoding "$x \trianglelefteq y \circ z$" in the following sense: any $\eta \in \mathrm{Val}_{\mathcal{C}}$ such that $\eta(X) = \imath(x)$, $\eta(Y) = \imath(y)$ and $\eta(Z) = \imath(z)$ verifies $\eta \in \mathrm{Sol}_{\mathcal{C}}(\mathsf{qBound}(X, Y, Z))$ iff $x \trianglelefteq y \circ z$.

In addition, if $\mathsf{qVal}(X)$ and $\mathsf{qBound}(X, Y, Z)$ can be chosen as existential constraints of the form $\exists X_1 \ldots \exists X_n (B_1 \wedge \ldots \wedge B_m)$—where B_j $(1 \leq j \leq m)$ are atomic—we say that $\mathcal{D}$ is *existentially expressible* in $\mathcal{C}$. $\square$

It can be proved that $\mathcal{B}, \mathcal{U}, \mathcal{W}$ and and any qualification domain built from these with the help of $\otimes$ are existentially expressible in any constraint domain $\mathcal{C}$ that includes the basic values and computational features of $\mathcal{R}$.

Admissible triples $\langle \mathcal{S}, \mathcal{D}, \mathcal{C} \rangle$ consist of a constraint domain $\mathcal{C}$, a qualification domain $\mathcal{D}$ and a proximity relation $\mathcal{S} : S \times S \to D$—where D is the carrier set of $\mathcal{D}$ and S is the set of all variables, basic values and signature symbols available in $\mathcal{C}$—satisfying the following properties:

- $\forall x \in S : \mathcal{S}(x, x) = \mathbf{t}$ (reflexivity).
- $\forall x, y \in S : \mathcal{S}(x, y) = \mathcal{S}(y, x)$ (symmetry).
- Some additional technical conditions explained in (Rodríguez-Artalejo and Romero-Díaz 2010b).

A proximity relation $\mathcal{S}$ is called *similarity* iff it satisfies the additional property $\forall x, y, z \in S : \mathcal{S}(x, z) \trianglerighteq \mathcal{S}(x, y) \sqcap \mathcal{S}(y, z)$ (transitivity). The scheme SQCLP has instances SQCLP$(\mathcal{S}, \mathcal{D}, \mathcal{C})$ where $\langle \mathcal{S}, \mathcal{D}, \mathcal{C} \rangle$ is an admissible triple.

A SQCLP$(\mathcal{S}, \mathcal{D}, \mathcal{C})$-program is a set $\mathcal{P}$ of *qualified program rules* (also called *qualified clauses*) $C : A \xleftarrow{\alpha} B_1 \sharp w_1, \ldots, B_m \sharp w_m$, where A is a defined atom, $\alpha \in D \setminus \{\mathbf{b}\}$ is called the *attenuation factor* of the clause and each $B_j \sharp w_j$ $(1 \leq j \leq m)$ is an atom B_j annotated with a so-called *threshold value* $w_j \in (D \setminus \{\mathbf{b}\}) \uplus \{?\}$. The intended meaning of C is as follows: if for all $1 \leq j \leq m$ one has $B_j \sharp e_j$ (meaning that B_j holds with qualification value e_j) for some $e_j \trianglerighteq^? w_j$, then $A \sharp d$ (meaning that A holds with qualification value d) can be inferred for any $d \in D \setminus \{\mathbf{b}\}$ such that $d \trianglelefteq \alpha \circ \bigsqcap_{j=1}^{m} e_j$. By convention, $e_j \trianglerighteq^? w_j$ means $e_j \trianglerighteq w_j$ if $w_j \neq ?$ and is identically true otherwise. In practice threshold values equal to '?' and attenuation values equal to $\mathbf{t}$ can be omitted.

Figure 1 shows a simple SQCLP$(\mathcal{S}_s, \mathcal{U}, \mathcal{R})$-program $\mathcal{P}_s$ which illustrates the expressivity of the SQCLP scheme to deal with problems involving flexible information retrieval. Predicate *search* can be used to answer queries asking for books in the library matching some desired language, genre and reader level. Predicate *guessRdrLvl* takes advantage of attenuation factors to encode heuristic rules to

```
    % Book representation: book( ID, Title, Author, Lang, Genre, VocLvl, Pages ).
1   library([ book(1, 'Tintin', 'Hergé', french, comic, easy, 65),
2       book(2, 'Dune', 'F.P. Herbert', english, sciFi, medium, 345),
3       book(3, 'Kritik der reinen Vernunft', 'I. Kant', german, philosophy, difficult, 1011),
4       book(4, 'Beim Hauten der Zwiebel', 'G. Grass', german, biography, medium, 432) ])

    % Auxiliary predicate for computing list membership:
5   member(B, [B|_])
6   member(B, [_|T]) ← member(B, T)

    % Predicates for getting the explicit attributes of a given book:
7   getId(book(ID, _Title, _Author, _Lang, _Genre, _VocLvl, _Pages), ID)
8   getTitle(book(_ID, Title, _Author, _Lang, _Genre, _VocLvl, _Pages), Title)
9   getAuthor(book(_ID, _Title, Author, _Lang, _Genre, _VocLvl, _Pages), Author)
10  getLanguage(book(_ID, _Title, _Author, Lang, _Genre, _VocLvl, _Pages), Lang)
11  getGenre(book(_ID, _Title, _Author, _Lang, Genre, _VocLvl, _Pages), Genre)
12  getVocLvl(book(_ID, _Title, _Author, _Lang, _Genre, VocLvl, _Pages), VocLvl)
13  getPages(book(_ID, _Title, _Author, _Lang, _Genre, _VocLvl, Pages), Pages)

    % Function for guessing the reader level of a given book:
14  guessRdrLvl(B, basic) ← getVocLvl(B, easy), getPages(B, N), N < 50
15  guessRdrLvl(B, intermediate) ←0.8 getVocLvl(B, easy), getPages(B, N), N ≥ 50
16  guessRdrLvl(B, basic) ←0.9 getGenre(B, children)
17  guessRdrLvl(B, proficiency) ←0.9 getVocbLvl(B, difficult), getPages(B, N), N ≥ 200
18  guessRdrLvl(B, upper) ←0.8 getVocLvl(B, difficult), getPages(B, N), N < 200
19  guessRdrLvl(B, intermediate) ←0.8 getVocLvl(B, medium)
20  guessRdrLvl(B, upper) ←0.7 getVocLvl(B, medium)

    % Function for answering a particular kind of user queries:
21  search(Lang, Genre, Level, Id) ← library(L)#1.0, member(B, L)#1.0,
22      getLanguage(B, Lang), getGenre(B, Genre),
23      guessRdrLvl(B, Level), getId(B, Id)#1.0

    % Proximity relation S_s:
24  S_s(sciFi, fantasy) = S_s(fantasy, sciFi) = 0.9
25  S_s(adventure, fantasy) = S_s(fantasy, adventure) = 0.7
26  S_s(essay, philosophy) = S_s(philosophy, essay) = 0.8
27  S_s(essay, biography) = S_s(biography, essay) = 0.7
```

Fig. 1. SQCLP($\mathcal{S}_s, \mathcal{U}, \mathcal{R}$)-program $\mathcal{P}_s$ (*Library with books in different languages*)

compute reader levels on the basis of vocabulary level and other book features. The other predicates compute book features in the natural way, and the proximity relation $\mathcal{S}_s$ allows flexibility in any unification (i.e. solving of equality constraints) arising during the invocation of the program predicates.

The declarative semantics of a given SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-program $\mathcal{P}$ relies on *qualified constrained atoms* (briefly *qc-atoms*) of the form $A\sharp d \Leftarrow \Pi$, intended to assert

that the validity of atom A with qualification degree $d \in D$ is entailed by the constraint set Π. A qc-atom is called *defined*, *primitive* or *equational* according to the syntactic form of A; and it is called *observable* iff $d \in D \setminus \{\mathbf{b}\}$ and Π is satisfiable.

Program interpretations are defined as sets of observable qc-atoms which obey a natural closure condition. The results proved in (Rodríguez-Artalejo and Romero-Díaz 2010a) show two equivalent ways to characterize declarative semantics, using a fix-point approach and a proof-theoretical approach, respectively. For the purposes of the present paper it suffices to consider the proof theoretical approach, that relies on a formal inference system called *Proximity-based Qualified Constrained Horn Logic*—in symbols, SQCHL($\mathcal{S},\mathcal{D},\mathcal{C}$)—intended to infer observable qc-atoms from $\mathcal{P}$ and consisting of the three inference rules displayed in Figure 2. Rule **SQEA** depends on a relation $\approx_{d,\Pi}$ between terms that is defined in the following way: $t \approx_{d,\Pi} s$ iff there exist two terms $\hat{t}$ and $\hat{s}$ such that $\Pi \models_{\mathcal{C}} t == \hat{t}$, $\Pi \models_{\mathcal{C}} s == \hat{s}$ and $\mathbf{b} \neq d \trianglelefteq \mathcal{S}(\hat{t},\hat{s})$. This allows to deduce equations from Π in a flexible way, taking the proximity relation $\mathcal{S}$ into account. The reader is referred to (Rodríguez-Artalejo and Romero-Díaz 2010b) for more motivating comments on SQCHL($\mathcal{S},\mathcal{D},\mathcal{C}$) and some technical properties of the $\approx_{d,\Pi}$ relation.

SQDA $$\frac{(\ (t'_i == t_i\theta)\sharp d_i \Leftarrow \Pi\ )_{i=1\ldots n} \quad (\ B_j\theta\sharp e_j \Leftarrow \Pi\ )_{j=1\ldots m}}{p'(\overline{t'}_n)\sharp d \Leftarrow \Pi}$$

if $(p(\overline{t}_n) \stackrel{\alpha}{\leftarrow} B_1\sharp w_1, \ldots, B_m\sharp w_m) \in \mathcal{P}$, θ subst., $\mathcal{S}(p',p) = d_0 \neq \mathbf{b}$, $e_j \trianglerighteq^{?} w_j$ $(1 \leq j \leq m)$ and $d \trianglelefteq \bigsqcap_{i=0}^{n} d_i \sqcap \alpha \circ \bigsqcap_{j=1}^{m} e_j$.

SQEA $$\frac{}{(t == s)\sharp d \Leftarrow \Pi}$$ if $t \approx_{d,\Pi} s$. **SQPA** $$\frac{}{\kappa\sharp d \Leftarrow \Pi}$$ if $\Pi \models_{\mathcal{C}} \kappa$.

Fig. 2. Proximity-based Qualified Constrained Horn Logic

We will write $\mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ to indicate that φ can be deduced from $\mathcal{P}$ in SQCHL($\mathcal{S}$, $\mathcal{D},\mathcal{C}$), and $\mathcal{P} \vdash^{k}_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi$ in the case that the deduction can be performed with exactly k **SQDA** inference steps. As usual in formal inference systems, SQCHL($\mathcal{S},\mathcal{D},\mathcal{C}$) proofs can be represented as *proof trees* whose nodes correspond to qc-atoms, each node being inferred from its children by means of some SQCHL($\mathcal{S},\mathcal{D},\mathcal{C}$) inference step. The following theorem, proved in (Rodríguez-Artalejo and Romero-Díaz 2010b), characterizes least program models in the scheme SQCLP. This result allows to use SQCHL($\mathcal{S},\mathcal{D},\mathcal{C}$)-derivability as a logical criterion for proving the semantic correctness of program transformations, as we will do in Section 4.

Theorem 2.1 (*Logical characterization of least program models in SQCHL*)
For any SQCLP($\mathcal{S},\mathcal{D},\mathcal{C}$)-program $\mathcal{P}$, its least model can be characterized as:

$$\mathcal{M}_{\mathcal{P}} = \{\varphi \mid \varphi \text{ is an observable defined qc-atom and } \mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \varphi\}. \quad \square$$

Let us now discuss goals and their solutions. Goals for a given SQCLP($\mathcal{S},\mathcal{D},\mathcal{C}$)-

program $\mathcal{P}$ have the form

$$G \ : \ A_1 \sharp W_1, \ \ldots, \ A_m \sharp W_m \ [\!] \ W_1 \trianglerighteq^? \beta_1, \ \ldots, \ W_m \trianglerighteq^? \beta_m$$

abbreviated as $(A_i \sharp W_i, \ W_i \trianglerighteq^? \beta_i)_{i=1\ldots m}$. The $A_i \sharp W_i$ are called *annotated atoms*. The pairwise different variables $W_i \in \mathcal{W}ar$ are called qualification variables; they are taken from a set $\mathcal{W}ar$ assumed to be disjoint from the set $\mathcal{V}ar$ of data variables used in terms. The conditions $W_i \trianglerighteq^? \beta_i$ (with $\beta_i \in (D \setminus \{\mathbf{b}\}) \uplus \{?\}$) are called *threshold conditions* and their intended meaning (relying on the notations '?' and '$\trianglerighteq^?$') is as already explained when introducing program clauses above. In the sequel, war(o) will denote the set of all qualification variables occurring in the syntactic object o. In particular, for a goal G as displayed above, war(G) denotes the set $\{W_i \mid 1 \leq i \leq m\}$. In the case $m = 1$ the goal is called *atomic*. The following definition relies on SQCHL($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-derivability to provide a natural declarative notion of goal solution:

Definition 2.2 (*Goal Solutions*)
Assume a given SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-program $\mathcal{P}$ and a goal G for $\mathcal{P}$ with the syntax displayed above. Then:

1. A *solution* for G is any triple $\langle \sigma, \mu, \Pi \rangle$ such that σ is a $\mathcal{C}$-substitution, $W\mu \in D \setminus \{\mathbf{b}\}$ for all $W \in \mathrm{dom}(\mu)$, Π is a satisfiable and finite set of atomic $\mathcal{C}$-constraints and the following two conditions hold for all $i = 1 \ldots m$: $W_i\mu = d_i \trianglerighteq^? \beta_i$ and $\mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} A_i\sigma \sharp W_i\mu \Leftarrow \Pi$. The set of all solutions for G w.r.t. $\mathcal{P}$ is noted $\mathrm{Sol}_{\mathcal{P}}(G)$.
2. A solution $\langle \eta, \rho, \Pi \rangle$ for G is called *ground* iff $\Pi = \emptyset$ and $\eta \in \mathrm{Val}_{\mathcal{C}}$ is a variable valuation such that $A_i\eta$ is a ground atom for all $i = 1 \ldots m$. The set of all ground solutions for G w.r.t. $\mathcal{P}$ is noted $\mathrm{GSol}_{\mathcal{P}}(G) \subseteq \mathrm{Sol}_{\mathcal{P}}(G)$.
3. A ground solution $\langle \eta, \rho, \emptyset \rangle \in \mathrm{GSol}_{\mathcal{P}}(G)$ is *subsumed* by $\langle \sigma, \mu, \Pi \rangle$ iff there is some $\nu \in \mathrm{Sol}_{\mathcal{C}}(\Pi)$ s.t. $\eta =_{\mathrm{var}(G)} \sigma\nu$ and $W_i\rho \trianglelefteq W_i\mu$ for $i = 1 \ldots m$. □

A possible goal G_s for the library program displayed in Figure 1 is

$$G_s \ : \ \mathit{search(german, essay, intermediate, ID)} \sharp W \ [\!] \ W \geq 0.65$$

and one solution for G_s is $\langle \{\mathit{ID} \mapsto 4\}, \{W \mapsto 0.7\}, \emptyset \rangle$. In this simple case, the constraint set Π within the solution is empty. Other examples of goal solutions can be found in (Rodríguez-Artalejo and Romero-Díaz 2010b) and Sections 4 and 5 below.

In practice, users of SQCLP languages will rely on some available *goal solving system* for computing goal solutions. The following definition specifies two important abstract properties of goal solving systems which will be taken as a reference for the implementation presented in this paper.

Definition 2.3 (*Correct Abstract Goal Solving Systems*)
An *abstract goal solving system* for SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$) is any device that takes a program $\mathcal{P}$ and a goal G as input and yields various triples $\langle \sigma, \mu, \Pi \rangle$, called *computed answers*, as outputs. Such a goal solving system is called:

1. *Sound* iff every computed answer is a solution $\langle \sigma, \mu, \Pi \rangle \in \mathrm{Sol}_{\mathcal{P}}(G)$.

2. *Weakly complete* iff every ground solution $\langle \eta, \rho, \emptyset \rangle \in \mathrm{GSol}_{\mathcal{P}}(G)$ is subsumed by some computed answer.
3. *Correct* iff it is both sound and weakly complete. □

Every goal solving system for a SQCLP instance should be sound and ideally also weakly complete. In principle, goal solving systems with these properties for extensions of the classical LP paradigm can be formalized as extensions of the well-known SLD-resolution method (Lloyd 1987; Apt 1990). A sound and complete extensions of SLD-resolution for the CLP scheme can be found e.g. in (Jaffar et al. 1998), and several extensions of SLD resolution for LP languages aiming at uncertain reasoning SQCLP scheme have been mentioned in Section 1.

Our aim in this paper is to present an implementation based on a semantically correct program transformation from SQCLP into CLP, rather than developing a sound and complete extension of SLD resolution. Nevertheless, both our implementation and SLD-based approaches for SLP languages in the line of (Sessa 2002) must share the ability to solve unification problems w.r.t. to a proximity relation $\mathcal{S} : S \times S \to [0, 1]$ over signature symbols, which is assumed to be transitive in (Sessa 2002) but not in our setting. The lack of transitivity makes a crucial difference w.r.t. the behavior of unification algorithms. In the rest of this section we briefly discuss the problem by means of a simple example.

(Sessa 2002) presents a flexible unification algorithm for solving unification problems represented as systems of the form $S \,[\!]\, \alpha$, where S is a set of equations between terms and α is a certainty degree. A solution of such a system is any substitution θ which verifies $\mathcal{S}(s\theta, t\theta) \geq \alpha$ for all equations $s == t$ belonging to S. This notion of solution is consistent with the declarative semantics of the SQCLP scheme (more specifically, with Definition 2.2), even in the case that $\mathcal{S}$ is a non-transitive proximity relation. Following a traditional approach, Sessa presents the flexible unification algorithm as set of transformation rules which convert systems $S \,[\!]\, \alpha$ into solved form systems which represent unifiers. The transformations are similar to those presented in e.g. Section 4.6 of (Baader and Nipkow 1998) for the case of classical syntactic unification, extended with suitable computations to update α during the process, taking the given similarity relation $\mathcal{S}$ into account. One of the transformations allows to transform a system of the form $X == t, S \,[\!]\, \alpha$ into $S\{X \mapsto t\} \,[\!]\, \alpha$ (provided that X is not identical to t and does not occur in t, the so-called occurs check). Unfortunately, this transformation can lose solutions in case that $\mathcal{S}$ is not transitive. Consider for instance the following example:

Example 2.1
Assume constants a, b, c and a non-transitive proximity relation $\mathcal{S}$ such that $\mathcal{S}(a, b) = \mathcal{S}(b, a) = 0.7$; $\mathcal{S}(a, c) = \mathcal{S}(c, a) = 0.8$; $\mathcal{S}(b, c) = \mathcal{S}(c, b) = 0$. Then, the substitution $\theta = \{X \mapsto a\}$ is obviously a solution of the unification problem $X == b,\ X == c \,[\!]\, 0.7$. Nevertheless, the unification algorithm presented in (Sessa 2002) and related papers fails without computing any solution:

$$X == b,\ X == c \,[\!]\, 0.7 \implies X == c\ \{X \mapsto b\} \,[\!]\, 0.7 \implies \mathit{fail}$$

The second transformation step leads to *fail* because $X == c\ \{X \mapsto b\} \,[\!]\, 0.7$ is

the same as $b == c \,[\!]\, 0.7$ and $\mathcal{S}(b,c) = 0 < 0.7$. Should $\mathcal{S}$ satisfy transitivity, then $\mathcal{S}(b,c) = \mathcal{S}(c,b) \geq 0.7$, and Sessa's unification algorithm would compute the unifier $\sigma = \{X \mapsto b\}$ as follows:

$$X == b,\, X == c \,[\!]\, 0.7 \implies X == c\,\{X \mapsto b\} \,[\!]\, 0.7 \implies \{X \mapsto b\} \,[\!]\, 0.7$$

Note that σ is more general than θ in the sense that $\mathcal{S}(\theta, \sigma\theta) = \mathcal{S}(\theta, \sigma) \geq 0.7$. Therefore this example does not contradict the completeness of Sessa's unification algorithm for the case of (transitive) similarity relations. □

Even in the case that $\mathcal{S}$ is transitive, we have found examples showing that a goal solving system based on Sessa's unification algorithm can fail to compute some valid solutions for SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-programs whose clauses use attenuation factors other than $\mathbf{t}$. The unification algorithm underlaying the implementations presented in Section 5—based on the program transformations from Section 4—avoids the problematic transformation step $X == t, S \,[\!]\, \alpha \Longrightarrow S\{X \mapsto t\} \,[\!]\, \alpha$, that might cause incompleteness; instead, Prolog's backtracking is used to implement the effect of a non-deterministic choice between several transformation steps $X == c(\overline{t}_n), S \,[\!]\, \alpha \Longrightarrow X_1 == t_1, \ldots, X_n == t_n, S\mu \,[\!]\, \alpha$, where $X_1, \ldots, X_n$ are fresh variables and $\mu = \{X \mapsto c'(\overline{X}_n)\}$ for some possible choice of c' such that $\mathcal{S}(c, c') \geq \alpha$.

As an optimization, our prototype system allows the user to use a directive whose effect is that the system avoids the backtracking search just discussed and implements just the effect of the transformation $X == t, S \,[\!]\, \alpha \Longrightarrow S\{X \mapsto t\} \,[\!]\, \alpha$. When including this directive, the user runs the risk of losing some valid solutions. We conjecture that no incompleteness occurs in the case of SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-programs based on a transitive $\mathcal{S}$ and whose clauses do not use attenuation factors other than $\mathbf{t}$; i.e. SLP programs enriched with constraint solving.

3 The Schemes QCLP & CLP as Specializations of SQCLP

As discussed in the concluding section of (Rodríguez-Artalejo and Romero-Díaz 2010a), several specializations of the SQCLP scheme can be obtained by partial instantiation of its parameters. In particular, QCLP and CLP can be defined as schemes with instances:

$$\begin{aligned} \mathrm{QCLP}(\mathcal{D}, \mathcal{C}) &=_{\mathrm{def}} \mathrm{SQCLP}(\mathcal{S}_{\mathrm{id}}, \mathcal{D}, \mathcal{C}) \\ \mathrm{CLP}(\mathcal{C}) &=_{\mathrm{def}} \mathrm{SQCLP}(\mathcal{S}_{\mathrm{id}}, \mathcal{B}, \mathcal{C}) = \mathrm{QCLP}(\mathcal{B}, \mathcal{C}) \end{aligned}$$

where $\mathcal{S}_{\mathrm{id}}$ is the *identity* proximity relation and $\mathcal{B}$ is the qualification domain including just the two classical boolean values. As explained in the introduction, QCLP and CLP are the targets of the two program transformations to be developed in Section 4. In this brief section we provide an explicit description of the syntax and semantics of these two schemes, derived from their behavior as specializations of SQCLP.

3.1 Presentation of the QCLP Scheme

As already explained, the instances of QCLP can be defined by the equation QCLP($\mathcal{D}$,$\mathcal{C}$) = SQCLP($\mathcal{S}_{\mathrm{id}}$,$\mathcal{D}$,$\mathcal{C}$). Due to the admissibility of the parameter triple

$\langle \mathcal{S}_{\mathrm{id}}, \mathcal{D}, \mathcal{C} \rangle$, the qualification domain $\mathcal{D}$ must be (existentially) expressible in the constraint domain $\mathcal{C}$. Technically, the QCLP scheme can be seen as a common extension of the classical CLP scheme for Constraint Logic Programming (Jaffar and Lassez 1987; Jaffar et al. 1998) and the QLP scheme for Qualified Logic Programming originally introduced in (Rodríguez-Artalejo and Romero-Díaz 2008). Intuitively, QCLP programming behaves like SQCLP programming, except that proximity information other than the identity is not available for proving equalities.

Program clauses and observable qc-atoms in QCLP are defined in the same way as in SQCLP. The library program $\mathcal{P}_s$ in Figure 1 becomes a QCLP$(\mathcal{U}, \mathcal{R})$-program $\mathcal{P}'_s$ just by replacing $\mathcal{S}_{\mathrm{id}}$ for $\mathcal{S}$. Of course, $\mathcal{P}'_s$ does not support flexible unification as it was the case with $\mathcal{P}_s$.

As explained in Section 2, the proof system consisting of the three displayed in Figure 2 characterizes the declarative semantics of a given SQCLP$(\mathcal{S}, \mathcal{D}, \mathcal{C})$-program $\mathcal{P}$. In the particular case $\mathcal{S} = \mathcal{S}_{\mathrm{id}}$, the inference rules specialize to those displayed in Figure 3, yielding a formal proof system called *Qualified Constrained Horn Logic* – in symbols, QCHL$(\mathcal{D}, \mathcal{C})$ – which characterizes the declarative semantics of a given QCLP$(\mathcal{D}, \mathcal{C})$-program $\mathcal{P}$. Note that rule **SQEA** depends on a relation $\approx_\Pi$ between terms that is defined to behave the same as the specialization of $\approx_{d,\Pi}$ to the case $\mathcal{S} = \mathcal{S}_{\mathrm{id}}$. It is easily checked that $t \approx_\Pi s$ does not depend on d and holds iff $\Pi \models_{\mathcal{C}} t == s$. Both $\approx_{d,\Pi}$ and $\approx_\Pi$ allow to use the constraints within Π when deducing equations. However, $c(\overline{t}_n) \approx_\Pi c'(\overline{s}_n)$ never holds in the case that c and c' are not syntactically identical.

QDA $$\frac{(\ (t'_i == t_i\theta) \sharp d_i \Leftarrow \Pi\ )_{i=1\ldots n} \quad (\ B_j\theta \sharp e_j \Leftarrow \Pi\ )_{j=1\ldots m}}{p(\overline{t'}_n) \sharp d \Leftarrow \Pi}$$

if $(p(\overline{t}_n) \xleftarrow{\alpha} B_1 \sharp w_1, \ldots, B_m \sharp w_m) \in \mathcal{P}$, θ subst.,
$e_j \trianglerighteq^? w_j$ $(1 \leq j \leq m)$ and $d \trianglelefteq \bigsqcap_{i=1}^n d_i \sqcap \alpha \circ \bigsqcap_{j=1}^m e_j$.

QEA $$\frac{}{(t == s) \sharp d \Leftarrow \Pi}$$ if $t \approx_\Pi s$. **QPA** $$\frac{}{\kappa \sharp d \Leftarrow \Pi}$$ if $\Pi \models_{\mathcal{C}} \kappa$.

Fig. 3. Qualified Constrained Horn Logic

SQCHL$(\mathcal{S}, \mathcal{D}, \mathcal{C})$ proof trees and the notations related to them can be naturally specialized to QCHL$(\mathcal{D}, \mathcal{C})$. In particular, we will use the notation $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} \varphi$ (resp. $\mathcal{P} \vdash^k_{\mathcal{D},\mathcal{C}} \varphi$) to indicate that the qc-atom φ can be inferred in QCHL$(\mathcal{D}, \mathcal{C})$ from the program $\mathcal{P}$ (resp. it can be inferred by using exactly k **QDA** inference steps). Theorem 2.1 also specializes to QCHL, yielding the following result:

Theorem 3.1 (*Logical characterization of least program models in QCHL*)
For any QCLP$(\mathcal{D}, \mathcal{C})$-program $\mathcal{P}$, its least model can be characterized as:

$$\mathcal{M}_{\mathcal{P}} = \{\varphi \mid \varphi \text{ is an observable defined qc-atom and } \mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} \varphi\}. \quad \square$$

Concerning goals and their solutions, their specialization to the particular case $\mathcal{S} = \mathcal{S}_{\mathrm{id}}$ leaves the syntax of goals G unaffected and leads to the following definition, almost identical to Definition 2.2:

Definition 3.1 (*Goal Solutions in QCLP*)
Assume a given QCLP($\mathcal{S}, \mathcal{D}$)$\mathcal{C}$-program $\mathcal{P}$ and a goal $G : (\ A_i \sharp W_i, W_i \trianglerighteq^? \beta_i\ )_{i=1\ldots m}$. Then:

1. A *solution* for G is any triple $\langle \sigma, \mu, \Pi \rangle$ such that σ is a $\mathcal{C}$-substitution, $W\mu \in D \setminus \{\mathbf{b}\}$ for all $W \in \mathrm{dom}(\mu)$, Π is a satisfiable and finite set of atomic $\mathcal{C}$-constraints, and the following two conditions hold for all $i = 1 \ldots m$: $W_i\mu = d_i \trianglerighteq^? \beta_i$ and $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} A_i\sigma \sharp W_i\mu \Leftarrow \Pi$. The set of all solutions for G is noted $\mathrm{Sol}_{\mathcal{P}}(G)$.
2. A solution $\langle \eta, \rho, \Pi \rangle$ for G is called *ground* iff $\Pi = \emptyset$ and $\eta \in \mathrm{Val}_{\mathcal{C}}$ is a variable valuation such that $A_i\eta$ is a ground atom for all $i = 1 \ldots m$. The set of all ground solutions for G is noted $\mathrm{GSol}_{\mathcal{P}}(G) \subseteq \mathrm{Sol}_{\mathcal{P}}(G)$.
3. A ground solution $\langle \eta, \rho, \emptyset \rangle \in \mathrm{GSol}_{\mathcal{P}}(G)$ is *subsumed* by $\langle \sigma, \mu, \Pi \rangle$ iff there is some $\nu \in \mathrm{Sol}_{\mathcal{C}}(\Pi)$ s.t. $\eta =_{\mathrm{var}(G)} \sigma\nu$ and $W_i\rho \trianglelefteq W_i\mu$ for $i = 1 \ldots m$. $\square$

Finally, the notion of correct abstract goal solving system for SQCLP given in Definition 2.3 specializes to QCLP without any formal change. Therefore, we state no new definition at this point.

3.2 Presentation of the CLP Scheme

As already explained, the instances of CLP can be defined by the equation CLP($\mathcal{C}$) = SQCLP($\mathcal{S}_{\mathrm{id}}, \mathcal{B}, \mathcal{C}$), or equivalently, CLP($\mathcal{C}$) = QCLP($\mathcal{B}, \mathcal{C}$). Due to the fixed choice $\mathcal{D} = \mathcal{B}$, the only qualification value $d \in D \setminus \{\mathbf{b}\}$ available for use as attenuation factor or threshold value is $d = \mathbf{t}$. Therefore, CLP can only include threshold values equal to '?' and attenuation values equal to the top element $\mathbf{t} = \mathit{true}$ of $\mathcal{B}$. As explained in Section 2, such trivial threshold and attenuation values can be omitted, and CLP clauses can be written with the simplified syntax $A \leftarrow B_1, \ldots, B_m$.

Since $\mathbf{t} = \mathit{true}$ is the only non-trivial qualification value available in CLP, qc-atoms $A\sharp d \Leftarrow \Pi$ are always of the form $A\sharp \mathit{true} \Leftarrow \Pi$ and can be written as $A \Leftarrow \Pi$. Moreover, all the side conditions for the inference rule **QDA** in Figure 3 become trivial when specialized to the case $\mathcal{D} = \mathcal{B}$. Therefore, the specialization of QCHL($\mathcal{D}, \mathcal{C}$) to the case $\mathcal{D} = \mathcal{B}$ leads to the formal proof system called *Constrained Horn Logic* – in symbols, CHL($\mathcal{C}$) – consisting of the three inference rules displayed in Figure 4, which characterizes the declarative semantics of a given CLP($\mathcal{C}$)-program $\mathcal{P}$.

QCHL($\mathcal{D}, \mathcal{C}$) proof trees and the notations related to them can be naturally specialized to CHL($\mathcal{C}$). In particular, we will use the notation $\mathcal{P} \vdash_{\varphi}$ (resp. $\mathcal{P} \vdash_k \varphi$) to indicate that the qc-atom φ can be inferred in CHL($\mathcal{C}$) from the program $\mathcal{P}$ (resp. it can be inferred by using exactly k **DA** inference steps). Theorem 3.1 also specializes to CHL, yielding the following result:

$$\textbf{DA}\quad \frac{(\ (t'_i == t_i\theta) \Leftarrow \Pi\ )_{i=1\ldots n}\quad (\ B_j\theta \Leftarrow \Pi\ )_{j=1\ldots m}}{p(\overline{t'}_n) \Leftarrow \Pi}$$

if $(p(\overline{t}_n) \leftarrow B_1, \ldots, B_m) \in \mathcal{P}$ and θ subst.

$$\textbf{EA}\quad \frac{}{(t == s) \Leftarrow \Pi}\ \text{if } t \approx_\Pi s. \qquad \textbf{PA}\quad \frac{}{\kappa \Leftarrow \Pi}\ \text{if } \Pi \models_{\mathcal{C}} \kappa.$$

Fig. 4. Constrained Horn Logic

Theorem 3.2 (*Logical characterization of least program models in CHL*)
For any CLP($\mathcal{C}$)-program $\mathcal{P}$, its least model can be characterized as:

$$\mathcal{M}_{\mathcal{P}} = \{\varphi \mid \varphi \text{ is an observable defined qc-atom and } \mathcal{P} \vdash_{\varphi} \}. \quad \square$$

Concerning goals and their solutions, their specialization to the scheme CLP leads to the following definition:

Definition 3.2 (*Goals and their Solutions in CLP*)
Assume a given CLP($\mathcal{C}$)-program $\mathcal{P}$. Then:

1. Goals for $\mathcal{P}$ have the form $G : A_1, \ldots, A_m$, abbreviated as $(\ A_i\ )_{i=1\ldots m}$, where A_i $(1 \leq i \leq m)$ are atoms.
2. A *solution* for a goal G is any pair $\langle \sigma, \Pi \rangle$ such that σ is a $\mathcal{C}$-substitution, Π is a satisfiable and finite set of atomic $\mathcal{C}$-constraints, and $\mathcal{P} \vdash_{\mathcal{C}} A_i\sigma \Leftarrow \Pi$ holds for all $i = 1 \ldots m$. The set of all solutions for G is noted $\mathrm{Sol}_{\mathcal{P}}(G)$.
3. A solution $\langle \eta, \Pi \rangle$ for G is called *ground* iff $\Pi = \emptyset$ and $\eta \in \mathrm{Val}_{\mathcal{C}}$ is a variable valuation such that $A_i\eta$ is a ground atom for all $i = 1 \ldots m$. The set of all ground solutions for G is noted $\mathrm{GSol}_{\mathcal{P}}(G)$. Obviously, $\mathrm{GSol}_{\mathcal{P}}(G) \subseteq \mathrm{Sol}_{\mathcal{P}}(G)$.
4. A ground solution $\langle \eta, \emptyset \rangle \in \mathrm{GSol}_{\mathcal{P}}(G)$ is *subsumed* by $\langle \sigma, \Pi \rangle$ iff there is some $\nu \in \mathrm{Sol}_{\mathcal{C}}(\Pi)$ s.t. $\eta =_{\mathrm{var}(G)} \sigma\nu$. $\square$

The notion of correct abstract goal solving system for SQCFLP given in Definition 2.3 specializes to CLP with only minor formal changes, as follows:

Definition 3.3 (*Correct Abstract Goal Solving Systems for CLP*)
A *goal solving system* for CLP($\mathcal{C}$) is any effective procedure which takes a program $\mathcal{P}$ and a goal G as input and yields various pairs $\langle \sigma, \Pi \rangle$, called *computed answers*, as outputs. Such a goal solving system is called:

1. *Sound* iff every computed answer is a solution $\langle \sigma, \Pi \rangle \in \mathrm{Sol}_{\mathcal{P}}(G)$.
2. *Weakly complete* iff every ground solution $\langle \eta, \emptyset \rangle \in \mathrm{GSol}_{\mathcal{P}}(G)$ is subsumed by some computed answer.
3. *Correct* iff it is both sound and weakly complete. $\square$

We close this Subsection with a technical lemma that will be useful for proving some results in Subsection 4.2:

Lemma 3.1
Assume an existential $\mathcal{C}$-constraint $\pi(\overline{X}_n) = \exists Y_1 \ldots \exists Y_k (B_1 \wedge \ldots \wedge B_m)$ with free variables $\overline{X}_n$ and a given CLP($\mathcal{C}$)-program $\mathcal{P}$ including the clause $C : p(\overline{X}_n) \leftarrow B_1, \ldots, B_m$, where $p \in DP^n$ does not occur at the head of any other clause of $\mathcal{P}$. Then, for any n-tuple $\bar{t}_n$ of $\mathcal{C}$-terms and any finite and satisfiable $\Pi \subseteq \mathrm{Con}_{\mathcal{C}}$, one has:

1. $\mathcal{P} \vdash_{\mathcal{C}} (p(\bar{t}_n) \Leftarrow \Pi) \Longrightarrow \Pi \models_{\mathcal{C}} \pi(\bar{t}_n)$, where $\pi(\bar{t}_n)$ stands for the result of applying the substitution $\{\overline{X}_n \mapsto \bar{t}_n\}$ to $\pi(\overline{X}_n)$.
2. The opposite implication $\Pi \models_{\mathcal{C}} \pi(\bar{t}_n) \Longrightarrow \mathcal{P} \vdash_{\mathcal{C}} (p(\bar{t}_n) \Leftarrow \Pi)$ holds if $\bar{t}_n$ is a ground term tuple. Note that for ground $\bar{t}_n$ the constraint entailment $\Pi \models_{\mathcal{C}} \pi(\bar{t}_n)$ simply means that $\pi(\bar{t}_n)$ is true in $\mathcal{C}$.
3. $\Pi \models_{\mathcal{C}} \pi(\bar{t}_n) \Longrightarrow \mathcal{P} \vdash_{\mathcal{C}} (p(\bar{t}_n) \Leftarrow \Pi)$ may fail if $\bar{t}_n$ is not a ground term tuple.

Proof
We prove each item separately:

1. Assume $\mathcal{P} \vdash_{\mathcal{C}} (p(\bar{t}_n) \Leftarrow \Pi)$. Note that C is the only clause for p in $\mathcal{P}$ and that each atom B_j in C's body is an atomic constraint. Therefore, the CHL($\mathcal{C}$) proof must use a **DA** step based on an instance $C\theta$ of clause C such that $\Pi \models_{\mathcal{C}} t_i == X_i\theta$ holds for all $1 \leq i \leq n$ and $\Pi \models B_j\theta$ holds for all $1 \leq j \leq m$. These conditions and the syntactic form of $\pi(\overline{X}_n)$ obviously imply $\Pi \models_{\mathcal{C}} \pi(\bar{t}_n)$.
2. Assume now $\Pi \models_{\mathcal{C}} \pi(\bar{t}_n)$ and $\bar{t}_n$ ground. Then $\pi(\bar{t}_n)$ is true in $\mathcal{C}$, and due to the syntactic form of $\pi(\overline{X}_n)$, there must be some substitution θ such that $X_i\theta = t_i$ (syntactic identity) for all $1 \leq i \leq n$ and $B_j\theta$ is ground and true in $\mathcal{C}$ for all $1 \leq j \leq m$. Trivially, $\Pi \models_{\mathcal{C}} t_i == X_i\theta$ holds for all $1 \leq i \leq n$ and $\Pi \models_{\mathcal{C}} B_j\theta$ also holds for all $1 \leq j \leq m$. Then, it is obvious that $\mathcal{P} \vdash_{\mathcal{C}} (p(\bar{t}_n) \Leftarrow \Pi)$ can be proved by using a **DA** step based on the instance $C\theta$ of clause C.
3. We prove that $\Pi \models_{\mathcal{C}} \pi(\bar{t}_n) \Longrightarrow \mathcal{P} \vdash_{\mathcal{C}} (p(\bar{t}_n) \Leftarrow \Pi)$ can fail if $\bar{t}_n$ is not ground by presenting a counterexample based on the constraint domain $\mathcal{R}$, using the syntax for $\mathcal{R}$-constraints explained in (Rodríguez-Artalejo and Romero-Díaz 2010b). Consider the existential $\mathcal{R}$-constraint $\pi(X) = \exists Y(op_+(Y, Y, X))$, and a CLP($\mathcal{R}$)-program $\mathcal{P}$ including the clause $C : p(X) \leftarrow op_+(Y, Y, X)$ and no other occurrence of the defined predicate symbol p. Consider also $\Pi = \{cp_{\geq}(X, 0.0)\}$ and $t = X$. Then $\Pi \models_{\mathcal{R}} \pi(X)$ is obviously true, because any real number $x \geq 0.0$ satisfies $\exists Y(op_+(Y, Y, x))$ in $\mathcal{R}$. However, there is no $\mathcal{R}$-term s such that $\Pi \models_{\mathcal{R}} op_+(s, s, X)$, and therefore there is no instance $C\theta$ of clause C that can be used to prove $\mathcal{P} \vdash_{\mathcal{C}} (p(X) \Leftarrow \Pi)$ by applying a **DA** step. □

4 Implementation by Program Transformation

The purpose of this section is to introduce a program transformation that transforms SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$) programs and goals into semantically equivalent CLP($\mathcal{C}$) programs and goals. This transformation is performed as the composition of the two following specific transformations:

1. $\text{elim}_{\mathcal{S}}$ — Eliminates the proximity relation $\mathcal{S}$ of arbitrary SQCLP($\mathcal{S},\mathcal{D},\mathcal{C}$) programs and goals, producing equivalent QCLP($\mathcal{D},\mathcal{C}$) programs and goals.
2. $\text{elim}_{\mathcal{D}}$ — Eliminates the qualification domain $\mathcal{D}$ of arbitrary QCLP($\mathcal{D},\mathcal{C}$) programs and goals, producing equivalent CLP($\mathcal{C}$) programs and goals.

Thus, given a SQCLP($\mathcal{S},\mathcal{D},\mathcal{C}$)-program $\mathcal{P}$—resp. SQCLP($\mathcal{S},\mathcal{D},\mathcal{C}$)-goal G—, the composition of the two transformations will produce an equivalent CLP($\mathcal{C}$)-program $\text{elim}_{\mathcal{D}}(\text{elim}_{\mathcal{S}}(\mathcal{P}))$—resp. CLP($\mathcal{C}$)-goal $\text{elim}_{\mathcal{D}}(\text{elim}_{\mathcal{S}}(G))$—.

Example 4.1 (*Running example: SQCLP($\mathcal{S}_r$, $\mathcal{U}\otimes\mathcal{W}$, $\mathcal{R}$)-program $\mathcal{P}_r$*)
As a running example for this section, consider the SQCLP($\mathcal{S}_r$, $\mathcal{U}\otimes\mathcal{W}$, $\mathcal{R}$)-program $\mathcal{P}_r$ as follows:

$$
\begin{array}{ll}
_{R_1} & \textit{famous(sha)} \xleftarrow{(0.9,1)} \\
_{R_2} & \textit{wrote(sha, kle)} \xleftarrow{(1,1)} \\
_{R_3} & \textit{wrote(sha, hamlet)} \xleftarrow{(1,1)} \\
_{R_4} & \textit{good\_work(G)} \xleftarrow{(0.75,3)} \textit{famous(A)\#(0.5,100), authored(A, G)} \\
 & \\
_{S_1} & \mathcal{S}_r\textit{(wrote, authored)} = \mathcal{S}_r\textit{(authored, wrote)} = \textit{(0.9,0)} \\
_{S_2} & \mathcal{S}_r\textit{(kle, kli)} = \mathcal{S}_r\textit{(kli, kle)} = \textit{(0.8,2)}
\end{array}
$$

where the constants *shakespeare*, *king_lear* and *king_liar* have been respectively replaced, for clarity purposes in the subsequent examples, by *sha*, *kle* and *kli*.

In addition, consider the SQCLP($\mathcal{S}_r$, $\mathcal{U}\otimes\mathcal{W}$, $\mathcal{R}$)-goal G_r as follows:

$$\textit{good\_work(X)\#W} \;[\!]\; W \trianglerighteq^{?} \textit{(0.5,10)}$$

We will illustrate the two transformation by showing, in subsequent examples, the program clauses of $\text{elim}_{\mathcal{S}}(\mathcal{P}_r)$ and $\text{elim}_{\mathcal{D}}(\text{elim}_{\mathcal{S}}(\mathcal{P}_r))$ and the goals $\text{elim}_{\mathcal{S}}(G_r)$ and $\text{elim}_{\mathcal{D}}(\text{elim}_{\mathcal{S}}(G_r))$. □

The next two subsections explain each transformation in detail.

4.1 Transforming SQCLP into QCLP

In this subsection we assume that the triple $\langle\mathcal{S},\mathcal{D},\mathcal{C}\rangle$ is admissible. In the sequel we say that a defined predicate symbol $p \in DP^n$ is *affected* by a SQCLP($\mathcal{S},\mathcal{D},\mathcal{C}$)-program $\mathcal{P}$ iff $\mathcal{S}(p,p') \neq \mathbf{b}$ for some p' occurring in $\mathcal{P}$. We also say that an atom A is *relevant* for $\mathcal{P}$ iff some of the three following cases hold: a) A is an equation $t == s$; b) A is a primitive atom κ; or c) A is a defined atom $p(\bar{t}_n)$ such that p is affected by $\mathcal{P}$.

As a first step towards the definition of the first program transformation $\text{elim}_{\mathcal{S}}$, we define a set $EQ_{\mathcal{S}}$ of QCLP($\mathcal{D},\mathcal{C}$) program clauses that emulates the behavior of equations in SQCLP($\mathcal{S},\mathcal{D},\mathcal{C}$). The following definition assumes that the binary predicate symbol $\sim \in DP^2$ (used in infix notation) and the nullary predicate symbols $\text{pay}_\lambda \in DP^0$ are not affected by $\mathcal{P}$.

Definition 4.1
We define $EQ_{\mathcal{S}}$ as the following QCLP($\mathcal{D},\mathcal{C}$)-program:

$$\begin{aligned} EQ_{\mathcal{S}} =_{\text{def}} & \{\ X \sim Y \overset{\mathbf{t}}{\leftarrow} (X == Y)\sharp?\ \} \\ & \bigcup \{\ u \sim u' \overset{\mathbf{t}}{\leftarrow} \text{pay}_\lambda\sharp? \mid u, u' \in B_{\mathcal{C}} \text{ and } \mathcal{S}(u,u') = \lambda \neq \mathbf{b}\ \} \\ & \bigcup \{\ c(\overline{X}_n) \sim c'(\overline{Y}_n) \overset{\mathbf{t}}{\leftarrow} \text{pay}_\lambda\sharp?, (\ (X_i \sim Y_i)\sharp?\ )_{i=1\ldots n} \mid c, c' \in DC^n \\ & \quad \text{and } \mathcal{S}(c,c') = \lambda \neq \mathbf{b}\ \} \\ & \bigcup \{\ \text{pay}_\lambda \overset{\lambda}{\leftarrow} \mid \text{ for each } \lambda \in D \setminus \{\mathbf{b}\}\ \}. \quad \square \end{aligned}$$

The following lemma shows the relation between the semantics of equations in SQCHL($\mathcal{S},\mathcal{D},\mathcal{C}$) and the behavior of the binary predicate symbol '$\sim$' defined by $EQ_{\mathcal{S}}$ in QCHL($\mathcal{D},\mathcal{C}$).

Lemma 4.1
Consider any two arbitrary terms t and s; $EQ_{\mathcal{S}}$ defined as in Definition 4.1; and a satisfiable finite set Π of $\mathcal{C}$-constraints. Then, for every $d \in D \setminus \{\mathbf{b}\}$:

$$t \approx_{d,\Pi} s \Longleftrightarrow EQ_{\mathcal{S}} \vdash_{\mathcal{D},\mathcal{C}} (t \sim s)\sharp d \Leftarrow \Pi\ .$$

Proof
We separately prove each implication.

[$\Longrightarrow$] Assume $t \approx_{d,\Pi} s$. Then, there are two terms $\hat{t}$, $\hat{s}$ such that:

$$(1)\ t \approx_\Pi \hat{t} \qquad (2)\ s \approx_\Pi \hat{s} \qquad (3)\ \hat{t} \approx_d \hat{s}$$

We use structural induction on the form of the term $\hat{t}$.

- $\hat{t} = Z$, $Z \in \mathcal{V}ar$. From (3) we have $\hat{s} = Z$. Then (1) and (2) become $t \approx_\Pi Z$ and $s \approx_\Pi Z$, therefore $t \approx_\Pi s$. Now $EQ_{\mathcal{S}} \vdash_{\mathcal{D},\mathcal{C}} (t \sim s)\sharp d \Leftarrow \Pi$ can be proved with a proof tree rooted by a **QDA** step of the form:

$$\frac{(t == X\theta)\sharp\mathbf{t} \Leftarrow \Pi \qquad (s == Y\theta)\sharp\mathbf{t} \Leftarrow \Pi \qquad (X == Y)\theta\sharp\mathbf{t} \Leftarrow \Pi}{(t \sim s)\sharp d \Leftarrow \Pi}$$

  using the clause $X \sim Y \overset{\mathbf{t}}{\leftarrow} (X == Y)\sharp? \in EQ_{\mathcal{S}}$ instantiated by the substitution $\theta = \{X \mapsto t,\ Y \mapsto s\}$. Therefore the three premises can be derived from $EQ_{\mathcal{S}}$ with **QEA** steps since $t \approx_\Pi t$, $s \approx_\Pi s$ and $t \approx_\Pi s$, respectively. Checking the side conditions of all inference steps is straightforward.
- $\hat{t} = u$, $u \in B_{\mathcal{C}}$. From (3) we have $\hat{s} = u'$ for some $u' \in B_{\mathcal{C}}$ such that $d \trianglelefteq \lambda = \mathcal{S}(u,u')$. Then (1) and (2) become $t \approx_\Pi u$ and $s \approx_\Pi u'$, which allow to build a proof of $EQ_{\mathcal{S}} \vdash_{\mathcal{D},\mathcal{C}} (t \sim s)\sharp d \Leftarrow \Pi$ by means of a **QDA** step using the clause $u \sim u' \overset{\mathbf{t}}{\leftarrow} \text{pay}_\lambda\sharp?$.
- $\hat{t} = c$, $c \in DC^0$. From (3) we have $\hat{s} = c'$ for some $c' \in DC^0$ such that $d \trianglelefteq \lambda = \mathcal{S}(c,c')$. Then (1) and (2) become $t \approx_\Pi c$ and $s \approx_\Pi c'$, which allow us to build a proof of $EQ_{\mathcal{S}} \vdash_{\mathcal{D},\mathcal{C}} (t \sim s)\sharp d \Leftarrow \Pi$ by means of a **QDA** step using the clause $c \sim c' \overset{\mathbf{t}}{\leftarrow} \text{pay}_\lambda\sharp?$.
- $\hat{t} = c(\overline{t}_n)$, $c \in DC^n$ with $n > 0$. In this case, and because of (3), we can assume $\hat{s} = c'(\overline{s}_n)$ for some $c' \in DC^n$ satisfying $d \trianglelefteq d_0 =_{\text{def}} \mathcal{S}(c,c')$ and $d \trianglelefteq d_i =_{\text{def}} \mathcal{S}(t_i,s_i)$

for $i = 1 \ldots n$. Then $EQ_{\mathcal{S}} \vdash_{\mathcal{D},\mathcal{C}} (t \sim s)\sharp d \Leftarrow \Pi$ with a proof tree rooted by a **QDA** step of the form:

$$\frac{\begin{array}{ll}(t == c(\overline{t}_n))\sharp\mathbf{t} \Leftarrow \Pi & \mathrm{pay}_{d_0}\sharp d_0 \Leftarrow \Pi \\ (s == c'(\overline{s}_n))\sharp\mathbf{t} \Leftarrow \Pi & (\ (t_i \sim s_i)\sharp d_i \Leftarrow \Pi\ )_{i=1\ldots n}\end{array}}{(t \sim s)\sharp d \Leftarrow \Pi}$$

using the $EQ_{\mathcal{S}}$ clause $C : c(\overline{X}_n) \sim c'(\overline{Y}_n) \overset{\mathbf{t}}{\leftarrow} \mathrm{pay}_{d_0}\sharp ?, ((X_i \sim Y_i)\sharp ?)_{i=1\ldots n}$ instantiated by the substitution $\theta = \{X_1 \mapsto t_1,\ Y_1 \mapsto s_1,\ \ldots,\ X_n \mapsto t_n,\ Y_n \mapsto s_n\}$. Note that C has attenuation factor $\mathbf{t}$ and threshold values ? at the body. Therefore, the side conditions of the **QDA** step boil down to $d \trianglelefteqslant d_i$ $(1 \leq i \leq n)$ which are true by assumption. It remains to prove that each premise of the **QDA** step can be derived from $EQ_{\mathcal{S}}$ in QCHL$(\mathcal{D},\mathcal{C})$:

- $EQ_{\mathcal{S}} \vdash_{\mathcal{D},\mathcal{C}} (t == c(\overline{t}_n))\sharp\mathbf{t} \Leftarrow \Pi$ and $EQ_{\mathcal{S}} \vdash_{\mathcal{D},\mathcal{C}} (s == c'(\overline{s}_n))\sharp\mathbf{t} \Leftarrow \Pi$ are trivial consequences of $t \approx_\Pi c(\overline{t}_n)$ and $s \approx_\Pi c'(\overline{s}_n)$, respectively. In both cases, the QCHL$(\mathcal{D},\mathcal{C})$ proofs consist of one single **QEA** step.
- $EQ_{\mathcal{S}} \vdash_{\mathcal{D},\mathcal{C}} \mathrm{pay}_{d_0}\sharp d_0 \Leftarrow \Pi$ can be proved using the clause $\mathrm{pay}_{d_0} \overset{d_0}{\leftarrow} \in EQ_{\mathcal{S}}$ in one single **QDA** step.
- $EQ_{\mathcal{S}} \vdash_{\mathcal{D},\mathcal{C}} (t_i \sim s_i)\sharp d_i \Leftarrow \Pi$ for $i = 1 \ldots n$. For each i, we observe that $t_i \approx_{d_i,\Pi} s_i$ holds because of $\hat{t}_i = t_i$, $\hat{s}_i = s_i$ which satisfy $t_i \approx_\Pi \hat{t}_i$, $s_i \approx_\Pi \hat{s}_i$ and $\hat{t}_i \approx_{d_i} \hat{s}_i$. Since $\hat{t}_i = t_i$ is a subterm of $\hat{t} = c(\overline{t}_n)$, the inductive hypothesis can be applied.

[$\Longleftarrow$] Let T be a QCHL$(\mathcal{D},\mathcal{C})$-proof tree witnessing $EQ_{\mathcal{S}} \vdash_{\mathcal{D},\mathcal{C}} (t \sim s)\sharp d \Leftarrow \Pi$. We prove $t \approx_{d,\Pi} s$ reasoning by induction on the number $n = \|T\|$ of nodes in T that represent conclusions of **QDA** inference steps. Note that all the program clauses belonging to $EQ_{\mathcal{S}}$ define either the binary predicate symbol '$\sim$' or the nullary predicates pay_λ.

Basis ($n = 1$).
In this case we have for the **QDA** inference step that there can be used three possible $EQ_{\mathcal{S}}$ clauses:

1. The program clause is $X \sim Y \overset{\mathbf{t}}{\leftarrow} (X == Y)\sharp ?$. Then the **QDA** inference step must be of the form:
$$\frac{(t == t')\sharp d_1 \Leftarrow \Pi \quad (s == s')\sharp d_2 \Leftarrow \Pi \quad (t' == s')\sharp e_1 \Leftarrow \Pi}{(t \sim s)\sharp d \Leftarrow \Pi}$$
with $d \trianglelefteqslant d_1 \sqcap d_2 \sqcap e_1$. The proof of the three premises must use the **QEA** inference rule. Because of the conditions of this inference rule we have $t \approx_\Pi t'$, $s \approx_\Pi s'$ and $t' \approx_\Pi s'$. Therefore $t \approx_\Pi s$ is clear. Then $t \approx_{d,\Pi} s$ holds by taking $\hat{t} = \hat{s} = t$ because, trivially, $t \approx_\Pi \hat{t}$, $s \approx_\Pi \hat{s}$ and $\hat{t} \approx_d \hat{s}$.
2. The program clause is $u \sim u' \overset{\mathbf{t}}{\leftarrow} \mathrm{pay}_\lambda\sharp ?$ with $u, u' \in B_{\mathcal{C}}$ such that $\mathcal{S}(u, u') = \lambda \neq \mathbf{b}$. The **QDA** inference step must be of the form:
$$\frac{(t == u)\sharp d_1 \Leftarrow \Pi \quad (s == u')\sharp d_2 \Leftarrow \Pi \quad \mathrm{pay}_\lambda\sharp e_1 \Leftarrow \Pi}{(t \sim s)\sharp d \Leftarrow \Pi}$$

with $d \trianglelefteqslant d_1 \sqcap d_2 \sqcap e_1$. Due to the forms of the **QEA** inference rule and the $EQ_{\mathcal{S}}$ clause $\text{pay}_\lambda \overset{\lambda}{\leftarrow}$, we can assume without loss of generality that $d_1 = d_2 = \mathbf{t}$ and $e_1 = \lambda$. Therefore $d \trianglelefteqslant \lambda$. Moreover, the QCHL($\mathcal{D}$,$\mathcal{C}$) proofs of the first two premises must use **QEA** inferences. Consequently we have $t \approx_\Pi u$ and $s \approx_\Pi u'$. These facts and $u \approx_d u'$ imply $t \approx_{d,\Pi} s$.

3. The program clause is $c \sim c' \overset{\mathbf{t}}{\leftarrow} \text{pay}_\lambda \sharp ?$ with $c, c' \in DC^0$ such that $\mathcal{S}(c, c') = \lambda \neq \mathbf{b}$. The **QDA** inference step must be of the form:

$$\frac{(t == c)\sharp d_1 \Leftarrow \Pi \quad (s == c')\sharp d_2 \Leftarrow \Pi \quad \text{pay}_\lambda \sharp e_1 \Leftarrow \Pi}{(t \sim s)\sharp d \Leftarrow \Pi}$$

with $d \trianglelefteqslant d_1 \sqcap d_2 \sqcap e_1$. Due to the forms of the **QEA** inference rule and the $EQ_{\mathcal{S}}$ clause $\text{pay}_\lambda \overset{\lambda}{\leftarrow}$, we can assume without loss of generality that $d_1 = d_2 = \mathbf{t}$ and $e_1 = \lambda$. Therefore $d \trianglelefteqslant \lambda$. Moreover, the QCHL($\mathcal{D}$,$\mathcal{C}$) proofs of the first two premises must use **QEA** inferences. Consequently we have $t \approx_\Pi c$ and $s \approx_\Pi c'$. These facts and $c \approx_d c'$ imply $t \approx_{d,\Pi} s$.

Inductive step ($n > 1$).

In this case t and s must be of the form $t = c(\overline{t}_n)$ and $s = c'(\overline{s}_n)$. The $EQ_{\mathcal{S}}$ clause used in the **QDA** inference step at the root must be of the form:

$$c(\overline{X}_n) \sim c'(\overline{Y}_n) \overset{\mathbf{t}}{\leftarrow} \text{pay}_{d_0} \sharp ?,\ ((X_i \sim Y_i)\sharp ?)_{i=1\ldots n}$$

with $\mathcal{S}(c, c') = d_0 \neq \mathbf{b}$. The inference step at the root will be:

$$\frac{\begin{array}{ll}(t == c(\overline{t}_n))\sharp d_1 \Leftarrow \Pi & pay_{d_0}\sharp e_0 \Leftarrow \Pi \\ (s == c'(\overline{s}_n))\sharp d_2 \Leftarrow \Pi & (\ (t_i \sim s_i)\sharp e_i \Leftarrow \Pi\ )_{i=1\ldots n}\end{array}}{(t \sim s)\sharp d \Leftarrow \Pi}$$

with $d \trianglelefteqslant d_1 \sqcap d_2 \sqcap \bigsqcap_{i=0}^{n} e_i$. Due to the forms of the $EQ_{\mathcal{S}}$ clause $\text{pay}_{d_0} \overset{d_0}{\leftarrow}$ and the **QEA** inference rule there is no loss of generality in assuming $d_1 = d_2 = \mathbf{t}$ and $e_0 = d_0$, therefore we have $d \trianglelefteqslant d_0 \sqcap \bigsqcap_{i=1}^{n} e_i$. By the inductive hypothesis $t_i \approx_{e_i,\Pi} s_i$ $(1 \leq i \leq n)$, i.e. there are constructor terms $\hat{t}_i$, $\hat{s}_i$ such that $t_i \approx_\Pi \hat{t}_i$, $s_i \approx_\Pi \hat{s}_i$ and $\hat{t}_i \approx_{e_i} \hat{s}_i$ for $i = 1 \ldots n$. Thus, we can build $\hat{t} = c(\hat{t}_1, \ldots, \hat{t}_n)$ and $\hat{s} = c'(\hat{s}_1, \ldots, \hat{s}_n)$ having $t \approx_{d,\Pi} s$ because:

- $t \approx_\Pi \hat{t}$, i.e. $c(\overline{t}_n) \approx_\Pi c(\overline{\hat{t}}_n)$, by decomposition since $t_i \approx_\Pi \hat{t}_i$.
- $s \approx_\Pi \hat{s}$, i.e. $c'(\overline{s}_n) \approx_\Pi c'(\overline{\hat{s}}_n)$, again by decomposition since $s_i \approx_\Pi \hat{s}_i$.
- $\hat{t} \approx_d \hat{s}$, since $d \trianglelefteqslant d_0 \sqcap \bigsqcap_{i=1}^{n} e_i \trianglelefteqslant \mathcal{S}(c, c') \sqcap \bigsqcap_{i=1}^{n} \mathcal{S}(\hat{t}_i, \hat{s}_i) = \mathcal{S}(\hat{t}, \hat{s})$. $\square$

We are now ready to define $\text{elim}_{\mathcal{S}}$ acting over programs and goals.

Definition 4.2

Assume a SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-program $\mathcal{P}$ and a SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-goal G for $\mathcal{P}$ whose atoms are all relevant for $\mathcal{P}$. Then we define:

1. For each atom A, let $A_\sim$ be $t \sim s$ if $A : t == s$; otherwise let $A_\sim$ be A.
2. For each clause $C : (p(\overline{t}_n) \overset{\alpha}{\leftarrow} \overline{B}) \in \mathcal{P}$ let $\hat{\mathcal{C}}_{\mathcal{S}}$ be the set of QCLP($\mathcal{D}, \mathcal{C}$) clauses consisting of:

— The clause $\hat{C} : (\widehat{p}_C(\overline{t}_n) \stackrel{\alpha}{\leftarrow} \overline{B}_\sim)$, where $\widehat{p}_C \in DP^n$ is not affected by $\mathcal{P}$ (chosen in a different way for each C) and $\overline{B}_\sim$ is obtained from $\overline{B}$ by replacing each atom A occurring in $\overline{B}$ by $A_\sim$.
— A clause $p'(\overline{X}_n) \stackrel{\mathbf{t}}{\leftarrow} \mathrm{pay}_\lambda \sharp ?,\ ((X_i \sim t_i)\sharp ?)_{i=1\ldots n},\ \widehat{p}_C(\overline{t}_n)\sharp ?$ for each $p' \in DP^n$ such that $\mathcal{S}(p,p') = \lambda \neq \mathbf{b}$. Here, $\overline{X}_n$ must be chosen as n pairwise different variables not occurring in the clause C.

3. $\mathrm{elim}_\mathcal{S}(\mathcal{P})$ is the QCLP$(\mathcal{D},\mathcal{C})$-program $EQ_\mathcal{S} \cup \hat{\mathcal{P}}_\mathcal{S}$ where $\hat{\mathcal{P}}_\mathcal{S} =_{\mathrm{def}} \bigcup_{C\in\mathcal{P}} \hat{\mathcal{C}}_\mathcal{S}$.
4. $\mathrm{elim}_\mathcal{S}(G)$ is the QCLP$(\mathcal{D},\mathcal{C})$-goal $G_\sim$ obtained from G by replacing each atom A occurring in G by $A_\sim$. □

The following example illustrates the transformation $\mathrm{elim}_\mathcal{S}$.

Example 4.2 (Running example: QCLP($\mathcal{U}\otimes\mathcal{W}$, $\mathcal{R}$)-program $\mathrm{elim}_\mathcal{S}(\mathcal{P}_r)$)
Consider the SQCLP$(\mathcal{S}_r, \mathcal{U}\otimes\mathcal{W}, \mathcal{R})$-program $\mathcal{P}_r$ and the goal G_r for $\mathcal{P}_r$ as presented in Example 4.1. The transformed QCLP$(\mathcal{U}\otimes\mathcal{W}, \mathcal{R})$-program $\mathrm{elim}_\mathcal{S}(\mathcal{P}_r)$ is as follows:

$\hat{R}_1$ $\quad \hat{famous}_{R_1}(sha) \xleftarrow{(0.9,1)}$
$R_{1.1}$ $\quad famous(X) \leftarrow pay_\mathbf{t},\ X{\sim}sha,\ \hat{famous}_{R_1}(sha)$
$\hat{R}_2$ $\quad \hat{wrote}_{R_2}(sha,\ kle) \xleftarrow{(1,1)}$
$R_{2.1}$ $\quad wrote(X,\ Y) \leftarrow pay_\mathbf{t},\ X{\sim}sha,\ Y{\sim}kle,\ \hat{wrote}_{R_2}(sha,\ kle)$
$R_{2.2}$ $\quad authored(X,\ Y) \leftarrow pay_{(0.9,0)},\ X{\sim}sha,\ Y{\sim}kle,\ \hat{wrote}_{R_2}(sha,\ kle)$
$\hat{R}_3$ $\quad \hat{wrote}_{R_3}(sha,\ hamlet) \xleftarrow{(1,1)}$
$R_{3.1}$ $\quad wrote(X,\ Y) \leftarrow pay_\mathbf{t},\ X{\sim}sha,\ Y{\sim}hamlet,\ \hat{wrote}_{R_3}(sha,\ hamlet)$
$R_{3.2}$ $\quad authored(X,\ Y) \leftarrow pay_{(0.9,0)},\ X{\sim}sha,\ Y{\sim}hamlet,\ \hat{wrote}_{R_3}(sha,\ hamlet)$
$\hat{R}_4$ $\quad \hat{good\_work}_{R_4}(G) \xleftarrow{(0.75,3)} famous(A)\#(0.5{,}100),\ authored(A,\ G)$
$R_{4.1}$ $\quad good\_work(X) \leftarrow pay_\mathbf{t},\ X{\sim}G,\ \hat{good\_work}_{R_4}(G)$

% Program clauses for ∼:	*% Program clauses for pay:*
$X{\sim}Y \leftarrow X{==}Y$	$pay_\mathbf{t} \leftarrow$
$kle{\sim}kli \leftarrow pay_{(0.8,2)}$	$pay_{(0.9,0)} \xleftarrow{(0.9,0)}$
$[\ldots]$	$pay_{(0.8,2)} \xleftarrow{(0.8,2)}$

Finally, the goal $\mathrm{elim}_\mathcal{S}(G_r)$ for $\mathrm{elim}_\mathcal{S}(\mathcal{P}_r)$ is as follows:

$$good\_work(X)\#W \ [\!]\ W \trianglerighteq^? (0.5{,}10) \quad \square$$

The next theorem proves the semantic correctness of the program transformation.

Theorem 4.1
Consider a SQCLP$(\mathcal{S},\mathcal{D},\mathcal{C})$-program $\mathcal{P}$, an atom A relevant for $\mathcal{P}$, a qualification value $d \in D \setminus \{\mathbf{b}\}$ and a satisfiable finite set of $\mathcal{C}$-constraints Π. Then, the following two statements are equivalent:

1. $\mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} A\sharp d \Leftarrow \Pi$
2. $\mathrm{elim}_\mathcal{S}(\mathcal{P}) \vdash_{\mathcal{D},\mathcal{C}} A_\sim\sharp d \Leftarrow \Pi$

where $A_\sim$ is understood as in Definition 4.2(1).

Proof
We separately prove each implication.

[1. $\Rightarrow$ 2.] *(the transformation is complete).* Assume that T is a SQCHL($\mathcal{S},\mathcal{D},\mathcal{C}$) proof tree witnessing $\mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} A\sharp d \Leftarrow \Pi$. We want to show the existence of a QCHL($\mathcal{D},\mathcal{C}$) proof tree T' witnessing $\mathrm{elim}_{\mathcal{S}}(\mathcal{P}) \vdash_{\mathcal{D},\mathcal{C}} A_{\sim}\sharp d \Leftarrow \Pi$. We reason by complete induction on $\|T\|$. There are three possible cases according to the syntactic form of the atom A. In each case we argue how to build the desired proof tree T'.
— A is a primitive atom κ. In this case $A_{\sim}$ is also κ and T contains only one **SQPA** inference node. Because of the inference rules **SQPA** and **QPA**, both $\mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \kappa\sharp d \Leftarrow \Pi$ and $\mathrm{elim}_{\mathcal{S}}(\mathcal{P}) \vdash_{\mathcal{D},\mathcal{C}} \kappa\sharp d \Leftarrow \Pi$ are equivalent to $\Pi \models_{\mathcal{C}} \kappa$, therefore T' trivially contains just one **QPA** inference node.
— A is an equation $t == s$. In this case $A_{\sim}$ is $t \sim s$ and T contains just one **SQEA** inference node. We know $\mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} (t == s)\sharp d \Leftarrow \Pi$ is equivalent to $t \approx_{d,\Pi} s$ because of the inference rule **SQEA**. From this equivalence follows $EQ_{\mathcal{S}} \vdash_{\mathcal{D},\mathcal{C}} (t \sim s)\sharp d \Leftarrow \Pi$ due to Lemma 4.1 and hence $\mathrm{elim}_{\mathcal{S}}(\mathcal{P}) \vdash_{\mathcal{D},\mathcal{C}} (t \sim s)\sharp d \Leftarrow \Pi$ by construction of $\mathrm{elim}_{\mathcal{S}}(\mathcal{P})$. In this case, T' will be a proof tree rooted by a **QDA** inference step.
— A is a defined atom $p'(\overline{t'}_n)$ with $p' \in DP^n$. In this case $A_{\sim}$ is $p'(\overline{t'}_n)$ and the root inference of T must be a **SQDA** inference step of the form:

$$\frac{(\ (t'_i == t_i\theta)\sharp d_i \Leftarrow \Pi\ )_{i=1\ldots n} \quad (\ B_j\theta\sharp e_j \Leftarrow \Pi\ )_{j=1\ldots m}}{p'(\overline{t'}_n)\sharp d \Leftarrow \Pi}\ (\clubsuit)$$

with $C : (p(\overline{t}_n) \overset{\alpha}{\leftarrow} B_1\sharp w_1, \ldots, B_m\sharp w_m) \in \mathcal{P}$, θ substitution, $\mathcal{S}(p',p) = d_0 \neq \mathbf{b}$, $e_j \trianglerighteq^{?} w_j$ $(1 \leq j \leq m)$, $d \trianglelefteq d_i$ $(0 \leq i \leq n)$ and $d \trianglelefteq \alpha \circ e_j$ $(1 \leq j \leq m)$—which means $d \trianglelefteq \alpha$ in the case $m = 0$. We can assume that the first n premises at ($\clubsuit$) are proved in SQCLP($\mathcal{S},\mathcal{D},\mathcal{C}$) w.r.t. $\mathcal{P}$ by proof trees T_{1i} $(1 \leq i \leq n)$ satisfying $\|T_{1i}\| < \|T\|$ $(1 \leq i \leq n)$, and the last m premises at ($\clubsuit$) are proved in SQCLP($\mathcal{S},\mathcal{D},\mathcal{C}$) w.r.t. $\mathcal{P}$ by proof trees T_{2j} $(1 \leq j \leq m)$ satisfying $\|T_{2j}\| < \|T\|$ $(1 \leq j \leq m)$.

By Definition 4.2, we know that the transformed program $\mathrm{elim}_{\mathcal{S}}(\mathcal{P})$ contains two clauses of the following form:

$$\begin{array}{lll} \hat{C} & : & \hat{p}_C(\overline{t}_n) \overset{\alpha}{\leftarrow} B^1_{\sim}\sharp w_1,\ \ldots,\ B^m_{\sim}\sharp w_m \\ \hat{C}_{p'} & : & p'(\overline{X}_n) \overset{\mathbf{t}}{\leftarrow} \mathrm{pay}_{d_0}\sharp ?,\ (\ (X_i \sim t_i)\sharp ?\ )_{i=1\ldots n},\ \hat{p}_C(\overline{t}_n)\sharp ? \end{array}$$

where X_i $(1 \leq i \leq n)$ are fresh variables not occurring in C and $B^j_{\sim}$ $(1 \leq j \leq m)$ is the result of replacing '$\sim$' for '$==$' if B_j is equation; and B_j itself otherwise. Given that the n variables X_i do not occur in C, we can assume that $\sigma =_{\mathrm{def}} \theta' \uplus \theta$ with $\theta' =_{\mathrm{def}} \{X_1 \mapsto t'_1,\ \ldots,\ X_n \mapsto t'_n\}$ is a well-defined substitution. We claim that $\mathrm{elim}_{\mathcal{S}}(\mathcal{P}) \vdash_{\mathcal{D},\mathcal{C}} A_{\sim}\sharp d \Leftarrow \Pi$ can be proved with a proof tree T' rooted by the **QDA** inference step (♠.1), which uses the clause $\hat{C}_{p'}$ instantiated by σ and having $d_{n+1} = d$.

$$\frac{\begin{array}{l}(\ (t'_i == X_i\sigma)\sharp \mathbf{t} \Leftarrow \Pi\ )_{i=1\ldots n} \\ \mathrm{pay}_{d_0}\sigma\sharp d_0 \Leftarrow \Pi \\ (\ (X_i \sim t_i)\sigma\sharp d_i \Leftarrow \Pi\ )_{i=1\ldots n} \\ \hat{p}_C(\overline{t}_n)\sigma\sharp d_{n+1} \Leftarrow \Pi\end{array}}{p'(\overline{t'}_n)\sharp d \Leftarrow \Pi}\ (♠.1) \qquad \frac{\begin{array}{l}(\ (t'_i == X_i\theta')\sharp \mathbf{t} \Leftarrow \Pi\ )_{i=1\ldots n} \\ \mathrm{pay}_{d_0}\sharp d_0 \Leftarrow \Pi \\ (\ (X_i\theta' \sim t_i\theta)\sharp d_i \Leftarrow \Pi\ )_{i=1\ldots n} \\ \hat{p}_C(\overline{t}_n\theta)\sharp d_{n+1} \Leftarrow \Pi\end{array}}{p'(\overline{t'}_n)\sharp d \Leftarrow \Pi}\ (♠.2)$$

By construction of σ, ($\spadesuit$.1) can be rewritten as ($\spadesuit$.2), and in order to build the rest of T', we show that each premise of ($\spadesuit$.2) admits a proof in QCHL($\mathcal{D},\mathcal{C}$) w.r.t. the transformed program $\mathrm{elim}_{\mathcal{S}}(\mathcal{P})$:

- $\mathrm{elim}_{\mathcal{S}}(\mathcal{P}) \vdash_{\mathcal{D},\mathcal{C}} (t'_i == X_i\theta')\sharp\mathbf{t} \Leftarrow \Pi$ for $i = 1 \ldots n$. Straightforward using a single **QEA** inference step since $X_i\theta' = t'_i$ and $t'_i \approx_\Pi t'_i$ is trivially true.
- $\mathrm{elim}_{\mathcal{S}}(\mathcal{P}) \vdash_{\mathcal{D},\mathcal{C}} \mathrm{pay}_{d_0}\sharp d_0 \Leftarrow \Pi$. Immediate using the clause $(\mathrm{pay}_{d_0} \xleftarrow{d_0}) \in \mathrm{elim}_{\mathcal{S}}(\mathcal{P})$ with a single **QDA** inference step.
- $\mathrm{elim}_{\mathcal{S}}(\mathcal{P}) \vdash_{\mathcal{D},\mathcal{C}} (X_i\theta' \sim t_i\theta)\sharp d_i \Leftarrow \Pi$ for $i = 1 \ldots n$. From the first n premises of ($\clubsuit$) we know $\mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} (t'_i == t_i\theta)\sharp d_i \Leftarrow \Pi$ with a proof tree T_{1i} satisfying $\|T_{1i}\| < \|T\|$ for $i = 1 \ldots n$. Therefore, for $i = 1 \ldots n$, $\mathrm{elim}_{\mathcal{S}}(\mathcal{P}) \vdash_{\mathcal{D},\mathcal{C}} (t'_i \sim t_i\theta)\sharp d_i \Leftarrow \Pi$ with some QCHL($\mathcal{D}$,$\mathcal{C}$) proof tree T'_{1i} by inductive hypothesis. Since $(X_i\theta' \sim t_i\theta) = (t'_i \sim t_i\theta)$ for $i = 1 \ldots n$, we are done.
- $\mathrm{elim}_{\mathcal{S}}(\mathcal{P}) \vdash_{\mathcal{D},\mathcal{C}} \hat{p}_C(\overline{t}_n\theta)\sharp d \Leftarrow \Pi$. This is proved by a QCHL($\mathcal{D},\mathcal{C}$) proof tree with a **QDA** inference step node at its root of the following form:

$$\frac{(\ (t_i\theta == t_i\theta)\sharp d_i \Leftarrow \Pi\ )_{i=1\ldots n} \quad (\ B^j_{\sim}\theta\sharp e_j \Leftarrow \Pi\ )_{j=1\ldots m}}{\hat{p}_C(\overline{t}_n\theta)\sharp d \Leftarrow \Pi}\ (\heartsuit)$$

which uses the program clause $\hat{C}$ instantiated by the substitution θ. Once more, we have to check that the premises can be derived in QCHL($\mathcal{D},\mathcal{C}$) from the transformed program $\mathrm{elim}_{\mathcal{S}}(\mathcal{P})$ and that the side conditions of ($\heartsuit$) are satisfied:

— The first n premises can be trivially proved using **QEA** inference steps.
— The last m premises can be proved w.r.t. $\mathrm{elim}_{\mathcal{S}}(\mathcal{P})$ with some QCHL($\mathcal{D},\mathcal{C}$) proof trees T'_{2j} ($1 \le j \le m$) by the inductive hypothesis, since we have premises $(\ B_j\theta\sharp e_j \Leftarrow \Pi\ )_{j=1\ldots m}$ at ($\clubsuit$) that can be proved in SQCLP($\mathcal{S},\mathcal{D},\mathcal{C}$) w.r.t. $\mathcal{P}$ with proof trees T_{2j} of size $\|T_{2j}\| < \|T\|$ ($1 \le j \le m$).
— The side conditions—namely: $e_j \trianglerighteq^? w_j$ ($1 \le j \le m$), $d \trianglelefteq d_i$ ($1 \le i \le n$) and $d \trianglelefteq \alpha \circ e_j$ ($1 \le j \le m$)—trivially hold because they are also satisfied by ($\clubsuit$).

Finally, we complete the construction of T' by checking that ($\spadesuit$.2) satisfies the side conditions of the inference rule **QDA**:

- All threshold values at the body of $\hat{C}_{p'}$ are '?', therefore the first group of side conditions becomes $d_i \trianglerighteq^?$? ($0 \le i \le n+1$), which are trivially true.
- The second side condition reduces to $d \trianglelefteq \mathbf{t}$, which is also trivially true.
- The third, and last, side condition is $d \trianglelefteq \mathbf{t} \circ d_i$ ($0 \le i \le n+1$), or equivalently $d \trianglelefteq d_i$ ($0 \le i \le n+1$). In fact, $d \trianglelefteq d_i$ ($0 \le i \le n$) holds due to the side conditions in ($\clubsuit$), and $d \trianglelefteq d_{n+1}$ holds because $d_{n+1} = d$ by construction of ($\spadesuit$.1) and ($\spadesuit$.2).

[2. $\Rightarrow$ 1.] *(the transformation is sound).* Assume that T' is a QCHL($\mathcal{D},\mathcal{C}$) proof tree witnessing $\mathrm{elim}_{\mathcal{S}}(\mathcal{P}) \vdash_{\mathcal{D},\mathcal{C}} A_{\sim}\sharp d \Leftarrow \Pi$. We want to show the existence of a SQCHL($\mathcal{S},\mathcal{D},\mathcal{C}$) proof tree T witnessing $\mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} A\sharp d \Leftarrow \Pi$. We reason by complete induction of $\|T'\|$. There are three possible cases according to the syntactic form of the atom $A_{\sim}$. In each case we argue how to build the desired proof tree T.
— $A_{\sim}$ is a primitive atom κ. In this case A is also κ and T' contains only one **QPA** inference node. Both $\mathrm{elim}_{\mathcal{S}}(\mathcal{P}) \vdash_{\mathcal{D},\mathcal{C}} \kappa\sharp d \Leftarrow \Pi$ and $\mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} \kappa\sharp d \Leftarrow \Pi$ are equivalent

to $\Pi \models_{\mathcal{C}} \kappa$ because of the inference rules **QPA** and **SQPA**, therefore T trivially contains just one **SQPA** inference node.
— $A_\sim$ is of the form $t \sim s$. In this case A is $t == s$ and T' is rooted by a **QDA** inference step. From $\text{elim}_{\mathcal{S}}(\mathcal{P}) \vdash_{\mathcal{D},\mathcal{C}} (t \sim s)\sharp d \Leftarrow \Pi$ and by construction of $\text{elim}_{\mathcal{S}}(\mathcal{P})$ we have $EQ_{\mathcal{S}} \vdash_{\mathcal{D},\mathcal{C}} (t \sim s)\sharp d \Leftarrow \Pi$. By Lemma 4.1 we get $t \approx_{d,\Pi} s$ and, by the definition of the **SQEA** inference step, we can build T as a proof tree with only one **SQEA** inference node proving $\mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} (t == s)\sharp d \Leftarrow \Pi$.
— $A_\sim$ is a defined atom $p'(\overline{t}_n)$ with $p' \in DP^n$ and $p' \neq \sim$. In this case $A = A_\sim$ and the step at the root of T' must be a **QDA** inference step using a clause $C' \in \text{elim}_{\mathcal{S}}(\mathcal{P})$ with head predicate p' and a substitution θ. Because of Definition 4.2 and the fact that p' is relevant for $\mathcal{P}$, there must be some clause $C : (p(\overline{t}_n) \stackrel{\alpha}{\leftarrow} \overline{B}) \in \mathcal{P}$ such that $\mathcal{S}(p, p') = d_0 \neq \mathbf{b}$, and C' must be of the form:

$$C' : p'(\overline{X}_n) \stackrel{\mathbf{t}}{\leftarrow} \text{pay}_{d_0}\sharp?,\ ((X_i \sim t_i)\sharp?)_{i=1\ldots n},\ \hat{p}_C(\overline{t}_n)\sharp?$$

where the variables $\overline{X}_n$ do not occur in C. Thus the **QDA** inference step at the root of T' must be of the form:

$$\frac{\begin{array}{l}(\ (t'_i == X_i\theta)\sharp d_{1i} \Leftarrow \Pi\ )_{i=1\ldots n}\\ \text{pay}_{d_0}\theta\sharp e_{10} \Leftarrow \Pi\\ (\ (X_i \sim t_i)\theta\sharp e_{1i} \Leftarrow \Pi\ )_{i=1\ldots n}\\ \hat{p}_C(\overline{t}_n)\theta\sharp e_{1(n+1)} \Leftarrow \Pi\end{array}}{p'(\overline{t'}_n)\sharp d \Leftarrow \Pi}\ (\spadesuit)$$

and the proof of the last premise must use the only clause for $\hat{p}_C$ introduced in $\text{elim}_{\mathcal{S}}(\mathcal{P})$ according to Definition 4.2, i.e.:

$$\hat{C} : \hat{p}_C(\overline{t}_n) \stackrel{\alpha}{\leftarrow} B^1_\sim\sharp w_1,\ \ldots,\ B^m_\sim\sharp w_m\ .$$

Therefore, the proof of this premise must be of the form:

$$\frac{(\ (t_i\theta == t_i\theta')\sharp d_{2i} \Leftarrow \Pi\ )_{i=1\ldots n}\quad (\ B^j_\sim\theta'\sharp e_{2j} \Leftarrow \Pi\ )_{j=1\ldots m}}{\hat{p}_C(\overline{t}_n)\theta\sharp e_{1(n+1)} \Leftarrow \Pi}\ (\heartsuit)$$

for some substitution θ' not affecting $\overline{X}_n$. We can assume that the last m premises in ($\heartsuit$) are proved in $\text{QCHL}(\mathcal{D},\mathcal{C})$ w.r.t. $\text{elim}_{\mathcal{S}}(\mathcal{P})$ by proof trees T'_j satisfying $\|T'_j\| < \|T'\|$ $(1 \leq j \leq m)$. Then we use the substitution θ' and clause C to build a $\text{SQCHL}(\mathcal{S},\mathcal{D},\mathcal{C})$ proof tree T with a **SQDA** inference step at the root of the form:

$$\frac{(\ (t'_i == t_i\theta')\sharp e_{1i} \Leftarrow \Pi\ )_{i=1\ldots n}\quad (\ B_j\theta'\sharp e_{2j} \Leftarrow \Pi\ )_{j=1\ldots m}}{p'(\overline{t'}_n)\sharp d \Leftarrow \Pi}\ (\clubsuit)$$

Next we check that the premises of this inference step admit proofs in $\text{SQCHL}(\mathcal{S},\mathcal{D},\mathcal{C})$ and that ($\clubsuit$) satisfies the side conditions of a valid **SQDA** inference step.

- $\mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} (t'_i == t_i\theta')\sharp e_{1i} \Leftarrow \Pi$ for $i = 1 \ldots n$.
  - — From the premises $((X_i \sim t_i)\theta\sharp e_{1i} \Leftarrow \Pi)_{i=1\ldots n}$ of ($\spadesuit$) and by construction of $\text{elim}_{\mathcal{S}}(\mathcal{P})$ we know $EQ_{\mathcal{S}} \vdash_{\mathcal{D},\mathcal{C}} (X_i \sim t_i)\theta\sharp e_{1i} \Leftarrow \Pi$ $(1 \leq i \leq n)$. Therefore by Lemma 4.1 we have $X_i\theta \approx_{e_{1i},\Pi} t_i\theta$ for $i = 1 \ldots n$.
  - — Consider now the premises $((t'_i == X_i\theta)\sharp d_{1i} \Leftarrow \Pi)_{i=1\ldots n}$ of ($\spadesuit$). Their proofs

must rely on **QEA** inference steps, and therefore $t'_i \approx_\Pi X_i\theta$ holds for $i = 1 \ldots n$.

— Analogously, from the proofs of the premises $((t_i\theta == t_i\theta')\sharp d_{2i} \Leftarrow \Pi)_{i=1\ldots n}$ we have $t_i\theta \approx_\Pi t_i\theta'$ (or equivalently $t_i\theta' \approx_\Pi t_i\theta$) for $i = 1 \ldots n$.

From the previous points we have $X_i\theta \approx_{e_{1i},\Pi} t_i\theta$, $t'_i \approx_\Pi X_i\theta$ and $t_i\theta' \approx_\Pi t_i\theta$, which by Lemma 2.7(1) of (Rodríguez-Artalejo and Romero-Díaz 2010b) imply $t'_i \approx_{e_{1i},\Pi} t_i\theta'$ $(1 \leq i \leq n)$. Therefore the premises $((t'_i == t_i\theta')\sharp e_{1i} \Leftarrow \Pi)_{i=1\ldots n}$ can be proven in SQCHL$(\mathcal{S},\mathcal{D},\mathcal{C})$ using a **SQEA** inference step.

- $\mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} B_j\theta'\sharp e_{2j} \Leftarrow \Pi$ for $j = 1 \ldots m$. We know $\mathrm{elim}_{\mathcal{S}}(\mathcal{P}) \vdash_{\mathcal{D},\mathcal{C}} B^j_\sim\theta'\sharp e_{2j} \Leftarrow \Pi$ with a proof tree T'_j satisfying $\|T'_j\| < \|T'\|$ $(1 \leq j \leq m)$ because of (♡). Therefore we have, by inductive hypothesis, $\mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} B_j\theta'\sharp e_{2j} \Leftarrow \Pi$ for some SQCHL$(\mathcal{S},\mathcal{D},\mathcal{C})$ proof tree T_j $(1 \leq j \leq m)$.
- $\mathcal{S}(p,p') = d_0 \neq \mathbf{b}$. As seen above.
- $e_{2j} \mathrel{\mathbb{B}}^? w_j$ for $j = 1 \ldots m$. This is a side condition of the **QDA** step in (♡).
- $d \trianglelefteq e_{1i}$ for $i = 1 \ldots n$. Straightforward from the side conditions of (♠), which include $d \trianglelefteq \mathbf{t} \circ e_{1i}$ for $(0 \leq i \leq n+1)$.
- $d \trianglelefteq \alpha \circ e_{2j}$ for $j = 1 \ldots m$. This follows from the side conditions of (♠) and (♡), since we have $d \trianglelefteq \mathbf{t} \circ e_{1i}$ for $i = 0 \ldots n+1$ (in particular $d \trianglelefteq e_{1(n+1)}$) and $e_{1(n+1)} \trianglelefteq \alpha \circ e_{2j}$ for $j = 1 \ldots m$. □

Finally, the next theorem extends the previous result to goals.

Theorem 4.2
Let G be a goal for a SQCLP$(\mathcal{S},\mathcal{D},\mathcal{C})$-program $\mathcal{P}$ whose atoms are all relevant for $\mathcal{P}$. Assume $\mathcal{P}' = \mathrm{elim}_{\mathcal{S}}(\mathcal{P})$ and $G' = \mathrm{elim}_{\mathcal{S}}(G)$. Then, $\mathrm{Sol}_{\mathcal{P}}(G) = \mathrm{Sol}_{\mathcal{P}'}(G')$.

Proof
According to the definition of goals in Section 2, and Definition 4.2, G and G' must be of the form $(A_i\sharp W_i, W_i \mathrel{\mathbb{B}}^? \beta_i)_{i=1\ldots m}$ and $(A^i_\sim\sharp W_i, W_i \mathrel{\mathbb{B}}^? \beta_i)_{i=1\ldots m}$, respectively. By Definitions 2.2 and 3.1, both $\mathrm{Sol}_{\mathcal{P}}(G)$ and $\mathrm{Sol}_{\mathcal{P}'}(G')$ are sets of triples $\langle\sigma,\mu,\Pi\rangle$ where σ is a $\mathcal{C}$-substitution, $\mu : \mathrm{war}(G) \to D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ (note that $\mathrm{war}(G) = \mathrm{war}(G')$) and Π is a satisfiable finite set of $\mathcal{C}$-constraints. Moreover:

1. $\langle\sigma,\mu,\Pi\rangle \in \mathrm{Sol}_{\mathcal{P}}(G)$ iff $W_i\mu = d_i \mathrel{\mathbb{B}}^? \beta_i$ and $\mathcal{P} \vdash_{\mathcal{S},\mathcal{D},\mathcal{C}} A_i\sigma\sharp W_i\mu \Leftarrow \Pi$ $(1 \leq i \leq m)$.
2. $\langle\sigma,\mu,\Pi\rangle \in \mathrm{Sol}_{\mathcal{P}'}(G')$ iff $W_i\mu = d_i \mathrel{\mathbb{B}}^? \beta_i$ and $\mathcal{P}' \vdash_{\mathcal{D},\mathcal{C}} A^i_\sim\sigma\sharp W_i\mu \Leftarrow \Pi$ $(1 \leq i \leq m)$.

Because of Theorem 4.1, conditions (1) and (2) are equivalent. □

4.2 Transforming QCLP into CLP

The results presented in this subsection are dependant on the assumption that the qualification domain $\mathcal{D}$ is existentially expressible in the constraint domain $\mathcal{C}$ via an injective mapping $\imath : D_{\mathcal{D}} \setminus \{\mathbf{b}\} \to C_{\mathcal{C}}$ and two existential $\mathcal{C}$-constraints of the following form:

$$\mathsf{qVal}(X) = \exists U_1 \ldots \exists U_k (B_1 \wedge \ldots \wedge B_m)$$
$$\mathsf{qBound}(X,Y,Z) = \exists V_1 \ldots \exists V_l (C_1 \wedge \ldots \wedge C_q)$$

Our aim is to present semantically correct transformations from QCLP($\mathcal{D},\mathcal{C}$) into CLP($\mathcal{C}$), working both for programs and goals. In order to compute with the encodings of $\mathcal{D}$ values in $\mathcal{C}$, we will use the CLP($\mathcal{C}$)-program $E_\mathcal{D}$ consisting of the following two clauses:

$$qVal(X) \leftarrow B_1, \ldots, B_m$$
$$qBound(X,Y,Z) \leftarrow C_1, \ldots, C_q$$

where $qVal \in DP^1$ and $qBound \in DP^3$ do not occur in the QCLP($\mathcal{D},\mathcal{C}$) programs and goals to be transformed.

The lemma stated below is an immediate consequence of Lemma 3.1 and Definition 2.1.

Lemma 4.2
For any satisfiable finite set Π of $\mathcal{C}$-constraints one has:

1. For any ground term $t \in C_\mathcal{C}$:
$$t \in \mathrm{ran}(\imath) \iff \mathsf{qVal}(t) \text{ true in } \mathcal{C} \iff E_\mathcal{D} \vdash_\mathcal{C} qVal(t) \Leftarrow \Pi$$
2. For any ground terms $r = \imath(x)$, $s = \imath(y)$, $t = \imath(z)$ with $x, y, z \in D_\mathcal{D} \setminus \{\mathbf{b}\}$:
$$x \preccurlyeq y \circ z \iff \mathsf{qBound}(r,s,t) \text{ true in } \mathcal{C} \iff E_\mathcal{D} \vdash_\mathcal{C} qBound(r,s,t) \Leftarrow \Pi$$

The two items above are also valid if $E_\mathcal{D}$ is replaced by any CLP($\mathcal{C}$)-program including the two clauses in $E_\mathcal{D}$ and having no additional occurrences of *qVal* and *qBound* at the head of clauses. □

Now we are ready to define the transformations from QCLP($\mathcal{D},\mathcal{C}$) into CLP($\mathcal{C}$).

Definition 4.3
Assume that $\mathcal{D}$ is existentially expressible in $\mathcal{C}$, and let $\mathsf{qVal}(X)$, $\mathsf{qBound}(X,Y,Z)$ and $E_\mathcal{D}$ be as explained above. Assume also a QCLP($\mathcal{D},\mathcal{C}$)-program $\mathcal{P}$ and a QCLP($\mathcal{D},\mathcal{C}$)-goal G for $\mathcal{P}$ without occurrences of the defined predicate symbols *qVal* and *qBound*. Then:

1. $\mathcal{P}$ is transformed into the CLP($\mathcal{C}$)-program $\mathrm{elim}_\mathcal{D}(\mathcal{P})$ consisting of the two clauses in $E_\mathcal{D}$ and the transformed $C^\mathcal{T}$ of each clause $C \in \mathcal{P}$, built as specified in Figure 5. The transformation rules of this figure assume a different choice of $p' \in DP^{n+1}$ for each $p \in DP^n$.
2. G is transformed into the CLP($\mathcal{C}$)-goal $\mathrm{elim}_\mathcal{D}(G)$ built as specified in Figure 5. Note that the qualification variables $\overline{W}_n$ occurring in G become normal CLP variables in the transformed goal. □

The following example illustrates the transformation $\mathrm{elim}_\mathcal{D}$.

Example 4.3 (*Running example: CLP($\mathcal{R}$)-program* $\mathrm{elim}_\mathcal{D}(\mathrm{elim}_\mathcal{S}(\mathcal{P}_r))$)
Consider the QCLP($\mathcal{U}\otimes\mathcal{W},\mathcal{R}$)-program $\mathrm{elim}_\mathcal{S}(\mathcal{P}_r)$ and the goal $\mathrm{elim}_\mathcal{S}(G_r)$ for the same program as presented in Example 4.2. The transformed CLP($\mathcal{R}$)-program $\mathrm{elim}_\mathcal{D}(\mathrm{elim}_\mathcal{S}(\mathcal{P}_r))$ is as follows:

Transforming Atoms

TEA $(t == s)^{\mathcal{T}} = (t == s,\ \imath(\mathbf{t}))$.

TPA $(\kappa)^{\mathcal{T}} = (\kappa,\ \imath(\mathbf{t}))$ with κ primitive atom.

TDA $(p(\bar{t}_n))^{\mathcal{T}} = (p'(\bar{t}_n, W),\ W)$ with $p \in DP^n$ and W a fresh CLP variable.

Transforming qc-Atoms

TQCA
$$\frac{A^{\mathcal{T}} = (A', w)}{(A \sharp d \Leftarrow \Pi)^{\mathcal{T}} = (A' \Leftarrow \Pi,\ \{\mathsf{qVal}(w),\ \mathsf{qBound}(\imath(d), \imath(\mathbf{t}), w)\})}$$

Transforming Program Clauses

TPC
$$\frac{(\ B_j^{\mathcal{T}} = (B_j', w_j')\ )_{j=1\ldots m}}{C^{\mathcal{T}} = p'(\bar{t}_n, W)\ \leftarrow\ qVal(W),\ \left(\begin{array}{l} qVal(w_j'),\ \ulcorner w_j' \trianglerighteq^{?} \imath(w_j)\urcorner, \\ qBound(W, \imath(\alpha), w_j'),\ B_j' \end{array}\right)_{j=1\ldots m}}$$

where $C : p(\bar{t}_n) \overset{\alpha}{\leftarrow} B_1 \sharp w_1, \ldots, B_m \sharp w_m$, W is a fresh CLP variable and $\ulcorner w_j' \trianglerighteq^{?} \imath(w_j)\urcorner$ is omitted if $w_j = ?$, i.o.c. abbreviates $qBound(\imath(w_j), \imath(\mathbf{t}), w_j')$.

Transforming Goals

TG
$$\frac{(\ B_j^{\mathcal{T}} = (B_j', w_j')\ )_{j=1\ldots m}}{\mathrm{elim}_{\mathcal{D}}(G) = \left(\begin{array}{l} qVal(W_j),\ \ulcorner W_j \trianglerighteq^{?} \imath(\beta_j)\urcorner, \\ qVal(w_j'),\ qBound(W_j, \imath(\mathbf{t}), w_j'),\ B_j' \end{array}\right)_{j=1\ldots m}}$$

where $G : (B_j \sharp W_j, W_j \trianglerighteq^{?} \beta_j)_{j=1\ldots m}$ and $\ulcorner W_j \trianglerighteq^{?} \imath(\beta_i)\urcorner$ as in **TPC** above.

Fig. 5. Transformation rules

$\hat{R}_1$ $\hat{famous}_{R_1}(sha,\ W) \leftarrow qVal(W),\ qBound(W, \mathbf{t},\ (0.9,1))$

$R_{1.1}$ $famous(X,\ W) \leftarrow qVal(W),\ qVal(W_1),\ qBound(W, \mathbf{t}, W_1),\ pay_{\mathbf{t}}(W_1),$
$qVal(W_2),\ qBound(W, \mathbf{t}, W_2),\ {\sim}(X,\ sha,\ W_2),$
$qVal(W_3),\ qBound(W, \mathbf{t}, W_3),\ \hat{famous}_{R_1}(sha,\ W_3)$

$\hat{R}_2$ $\hat{wrote}_{R_2}(sha,\ kle,\ W) \leftarrow qVal(W),\ qBound(W, \mathbf{t},\ (1,1))$

$R_{2.1}$ $wrote(X,\ Y,\ W) \leftarrow qVal(W),\ qVal(W_1),\ qBound(W, \mathbf{t}, W_1),\ pay_{\mathbf{t}}(W_1),$
$qVal(W_2),\ qBound(W, \mathbf{t}, W_2),\ {\sim}(X,\ sha,\ W_2),$
$qVal(W_3),\ qBound(W, \mathbf{t}, W_3),\ {\sim}(Y,\ kle,\ W_3),$
$qVal(W_4),\ qBound(W, \mathbf{t}, W_4),\ \hat{wrote}_{R_2}(sha,\ kle,\ W_4)$

$R_{2.2}$ $authored(X,\ Y,\ W) \leftarrow qVal(W),\ qVal(W_1),\ qBound(W, \mathbf{t}, W_1),\ pay_{(0.9,0)}(W_1),$
$qVal(W_2),\ qBound(W, \mathbf{t}, W_2),\ {\sim}(X,\ sha,\ W_2),$
$qVal(W_3),\ qBound(W, \mathbf{t}, W_3),\ {\sim}(Y,\ kle,\ W_3),$
$qVal(W_4),\ qBound(W, \mathbf{t}, W_4),\ \hat{wrote}_{R_2}(sha,\ kle,\ W_4)$

$\hat{R}_3$ $\hat{wrote}_{R_3}(sha,\ hamlet,\ W) \leftarrow qVal(W),\ qBound(W, \mathbf{t},\ (1,1))$

$R_{3.1}$ $wrote(X,\ Y,\ W) \leftarrow qVal(W),\ qVal(W_1),\ qBound(W, \mathbf{t}, W_1),\ pay_{\mathbf{t}}(W_1),$
$qVal(W_2),\ qBound(W, \mathbf{t}, W_2),\ {\sim}(X,\ sha,\ W_2),$
$qVal(W_3),\ qBound(W, \mathbf{t}, W_3),\ {\sim}(Y,\ hamlet,\ W_3),$
$qVal(W_4),\ qBound(W, \mathbf{t}, W_4),\ \hat{wrote}_{R_3}(sha,\ hamlet,\ W_4)$

$R_{3.2}$ $\mathit{authored}(X,\ Y,\ W) \leftarrow \mathit{qVal}(W),\ \mathit{qVal}(W_1),\ \mathit{qBound}(W,\ \mathbf{t},\ W_1),\ \mathit{pay}_{(0.9,0)}(W_1),$
$\mathit{qVal}(W_2),\ \mathit{qBound}(W,\ \mathbf{t},\ W_2),\ {\sim}(X,\ \mathit{sha},\ W_2),$
$\mathit{qVal}(W_3),\ \mathit{qBound}(W,\ \mathbf{t},\ W_3),\ {\sim}(Y,\ \mathit{hamlet},\ W_3),$
$\mathit{qVal}(W_4),\ \mathit{qBound}(W,\ \mathbf{t},\ W_4),\ \hat{w}\mathit{rote}_{R_3}(\mathit{sha},\ \mathit{hamlet},\ W_4)$

$\hat{R}_4$ $\hat{g}\mathit{ood\_work}_{R_4}(X,\ W) \leftarrow \mathit{qVal}(W),$
$\mathit{qVal}(W_1),\ \mathit{qBound}((0.5{,}100),\ \mathbf{t},\ W_1),\ \mathit{qBound}(W,\ (0.75{,}3),\ W_1),\ \mathit{famous}(Y,\ W_1),$
$\mathit{qVal}(W_2),\ \mathit{qBound}(W,\ (0.75{,}3),\ W_2),\ \mathit{authored}(Y,\ X,\ W_2)$

$R_{4.1}$ $\mathit{good\_work}(G,\ W) \leftarrow \mathit{qVal}(W),\ \mathit{qVal}(W_1),\ \mathit{qBound}(W,\ \mathbf{t},\ W_1),\ \mathit{pay}_{\mathbf{t}}(W_1),$
$\mathit{qVal}(W_2),\ \mathit{qBound}(W,\ \mathbf{t},\ W_2),\ {\sim}(G,\ X,\ W_2),$
$\mathit{qVal}(W_3),\ \mathit{qBound}(W,\ \mathbf{t},\ W_3),\ \hat{g}\mathit{ood\_work}_{R_4}(X,\ W_3)$

% Program clauses for ∼:
${\sim}(X,\ Y,\ W) \leftarrow \mathit{qVal}(W),\ \mathit{qVal}(\mathbf{t}),\ \mathit{qBound}(W,\ \mathbf{t},\ \mathbf{t}),\ X{=}{=}Y$
${\sim}(\mathit{kle},\ \mathit{kli},\ W) \leftarrow \mathit{qVal}(W),\ \mathit{qVal}(W_1),\ \mathit{qBound}(W,\ \mathbf{t},\ W_1),\ \mathit{pay}_{(0.8,2)}(W_1)$
[...]

% Program clauses for pay:
$\mathit{pay}_{\mathbf{t}}(W) \leftarrow \mathit{qVal}(W),\ \mathit{qBound}(W,\ \mathbf{t},\ \mathbf{t})$
$\mathit{pay}_{(0.9,0)}(W) \leftarrow \mathit{qVal}(W),\ \mathit{qBound}(W,\ \mathbf{t},\ (0.9{,}0))$
$\mathit{pay}_{(0.8,2)}(W) \leftarrow \mathit{qVal}(W),\ \mathit{qBound}(W,\ \mathbf{t},\ (0.8{,}2))$

% Program clauses for qVal & qBound:
$\mathit{qVal}((X_1{,}X_2)) \leftarrow X_1 > 0,\ X_1 \leq 1,\ X_2 \geq 0$
$\mathit{qBound}((W_1{,}W_2),\ (Y_1{,}Y_2),\ (Z_1{,}Z_2)) \leftarrow W_1 \leq Y_1 \times Z_1,\ W_2 \geq Y_2 + Z_2$

Finally, the goal $\mathrm{elim}_{\mathcal{D}}(\mathrm{elim}_{\mathcal{S}}(G_r))$ for $\mathrm{elim}_{\mathcal{D}}(\mathrm{elim}_{\mathcal{S}}(\mathcal{P}_r))$ is as follows:

$\mathit{qVal}(W),\ \mathit{qBound}((0.5{,}10),\ \mathbf{t},\ W),\ \mathit{qVal}(W'),\ \mathit{qBound}(W,\ \mathbf{t},\ W'),\ \mathit{good\_work}(X,\ W')$

Note that, in order to improve the clarity of the program clauses of this example, the qualification value (1,0)—top value in $\mathcal{U}\otimes\mathcal{W}$—has been replaced by $\mathbf{t}$. □

The next theorem proves the semantic correctness of the program transformation.

Theorem 4.3
Let A be an atom such that *qVal* and *qBound* do not occur in A. Assume $d \in D\backslash\{\mathbf{b}\}$ such that $(A\sharp d \Leftarrow \Pi)^{\mathcal{T}} = (A' \Leftarrow \Pi, \Omega)$. Then, the two following statements are equivalent:

1. $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} A\sharp d \Leftarrow \Pi$
2. $\mathrm{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} A'\rho \Leftarrow \Pi$ for some $\rho \in \mathrm{Sol}_{\mathcal{C}}(\Omega)$ such that $\mathrm{dom}(\rho) = \mathrm{var}(\Omega)$.

Proof
We separately prove each implication.

[1. ⇒ 2.] *(the transformation is complete).* We assume that T is a QCHL($\mathcal{D},\mathcal{C}$) proof tree witnessing $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} A\sharp d \Leftarrow \Pi$. We want to show the existence of a CLP($\mathcal{C}$) proof tree T' witnessing $\mathrm{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} A'\rho \Leftarrow \Pi$ for some $\rho \in \mathrm{Sol}_{\mathcal{C}}(\Omega)$ such that $\mathrm{dom}(\rho) = \mathrm{var}(\Omega)$. We reason by complete induction on $\|T\|$. There are three possible

cases, according to the the syntactic form of the atom A. In each case we argue how to build the desired proof tree T'.

— A is a primitive atom κ. In this case **TQCA** and **TPA** compute $A' = \kappa$ and $\Omega = \{\mathsf{qVal}(\imath(\mathbf{t})),\ \mathsf{qBound}(\imath(d), \imath(\mathbf{t}), \imath(\mathbf{t}))\}$. Now, from $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} \kappa\sharp d \Leftarrow \Pi$ follows $\Pi \models_{\mathcal{C}} \kappa$ due to the **QPA** inference, and therefore taking $\rho = \varepsilon$ we can prove $\mathrm{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} \kappa\varepsilon \Leftarrow \Pi$ with a proof tree T' containing only one **PA** node. Moreover, $\varepsilon \in \mathrm{Sol}_{\mathcal{C}}(\Omega)$ is trivially true because the two constraints belonging to Ω are obviously true in $\mathcal{C}$.

— A is an equation $t == s$. In this case **TQCA** and **TEA** compute $A' = (t == s)$ and $\Omega = \{\mathsf{qVal}(\imath(\mathbf{t})),\ \mathsf{qBound}(\imath(d), \imath(\mathbf{t}), \imath(\mathbf{t}))\}$. Now, from $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} (t == s)\sharp d \Leftarrow \Pi$ follows $t \approx_\Pi s$ due to the **QEA** inference, and therefore taking $\rho = \varepsilon$ we can prove $\mathrm{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} (t == s)\varepsilon \Leftarrow \Pi$ with a proof tree T' containing only one **EA** node. Moreover, $\varepsilon \in \mathrm{Sol}_{\mathcal{C}}(\Omega)$ is trivially true because the two constraints belonging to Ω are obviously true in $\mathcal{C}$.

— A is a defined atom $p(\overline{t'}_n)$ with $p \in DP^n$. In this case **TQCA** and **TDA** compute $A' = p'(\overline{t'}_n, W)$ and $\Omega = \{\mathsf{qVal}(W),\ \mathsf{qBound}(\imath(d), \imath(\mathbf{t}), W)\}$ where W is a fresh CLP variable. On the other hand, T must be rooted by a **QDA** step of the form:

$$\frac{(\ (t'_i == t_i\theta)\sharp d_i \Leftarrow \Pi\ )_{i=1\ldots n} \quad (\ B_j\theta\sharp e_j \Leftarrow \Pi\ )_{j=1\ldots m}}{p(\overline{t'}_n)\sharp d \Leftarrow \Pi} \quad (\clubsuit)$$

using a clause $C : (p(\overline{t}_n) \overset{\alpha}{\Leftarrow} B_1\sharp w_1, \ldots, B_m\sharp w_m) \in \mathcal{P}$ instantiated by a substitution θ and such that the side conditions $e_j \rhd^? w_j$ $(1 \le j \le m)$, $d \lhd d_i$ $(1 \le i \le n)$ and $d \lhd \alpha \circ e_j$ $(1 \le j \le m)$ are fulfilled.

For $j = 1 \ldots m$ we can assume $B_j^{\mathcal{T}} = (B'_j, w'_j)$ and thus $(B_j\theta\sharp e_j \Leftarrow \Pi)^{\mathcal{T}} = (B'_j\theta \Leftarrow \Pi, \Omega_j)$ where $\Omega_j = \{\mathsf{qVal}(w'_j),\ \mathsf{qBound}(\imath(e_j), \imath(\mathbf{t}), w'_j)\}$. The proof trees T_j of the last m premises of $(\clubsuit)$ will have less than $\|T\|$ nodes, and hence the induction hypothesis can be applied to each $(B_j\theta\sharp e_j \Leftarrow \Pi)$ with $1 \le j \le m$, obtaining CHL($\mathcal{C}$) proof trees T'_j proving $\mathrm{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} B'_j\theta\rho_j \Leftarrow \Pi$ for some $\rho_j \in \mathrm{Sol}_{\mathcal{C}}(\Omega_j)$ with $\mathrm{dom}(\rho_j) = \mathrm{var}(\Omega_j)$.

Consider $\rho = \{W \mapsto \imath(d)\}$ and $C^{\mathcal{T}} \in \mathrm{elim}_{\mathcal{D}}(\mathcal{P})$ of the form:

$$C^{\mathcal{T}} :\ p'(\overline{t}_n, W') \ \leftarrow\ qVal(W'),\ \left(\begin{array}{l} qVal(w'_j),\ \ulcorner w'_j \rhd^? \imath(w_j)\urcorner, \\ qBound(W', \imath(\alpha), w'_j),\ B'_j \end{array}\right)_{j=1\ldots m.}$$

Obviously, $\rho \in \mathrm{Sol}_{\mathcal{C}}(\Omega)$ and $\mathrm{dom}(\rho) = \mathrm{var}(\Omega)$. To finish the proof we must prove $\mathrm{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} A'\rho \Leftarrow \Pi$. We claim that this can be done with a CHL($\mathcal{C}$) proof tree T' whose root inference is a **DA** step of the form:

$$\frac{\begin{array}{l}(\ (t'_i\rho == t_i\theta') \Leftarrow \Pi\ )_{i=1\ldots n} \\ (W\rho == W'\theta') \Leftarrow \Pi \\ qVal(W')\theta' \Leftarrow \Pi \\ \left(\begin{array}{l} qVal(w'_j)\theta' \Leftarrow \Pi \\ \ulcorner w'_j \rhd^? \imath(w_j)\urcorner\theta' \Leftarrow \Pi \\ qBound(W', \imath(\alpha), w'_j)\theta' \Leftarrow \Pi \\ B'_j\theta' \Leftarrow \Pi \end{array}\right)_{j=1\ldots m}\end{array}}{p'(\overline{t'}_n, W)\rho \Leftarrow \Pi} \quad (\spadesuit)$$

using $C^{\mathcal{T}}$ instantiated by the substitution $\theta' = \theta \uplus \rho_1 \uplus \cdots \uplus \rho_m \uplus \{W' \mapsto \imath(d)\}$. We check that the premises of (♠) can be derived from $\mathrm{elim}_{\mathcal{D}}(\mathcal{P})$ in CHL($\mathcal{C}$):

- $\mathrm{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} (t'_i\rho == t_i\theta') \Leftarrow \Pi$ for $i = 1 \ldots n$. By construction of ρ and θ', these are equivalent to prove $\mathrm{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} (t'_i == t_i\theta) \Leftarrow \Pi$ for $i = 1 \ldots n$ and these hold with CHL($\mathcal{C}$) proof trees of only one **EA** node because of $t'_i \approx_\Pi t_i\theta$, which is a consequence of the first n premises of (♣).
- $\mathrm{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} (W\rho == W'\theta') \Leftarrow \Pi$. By construction of ρ and θ', this is equivalent to prove $\mathrm{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} (\imath(d) == \imath(d)) \Leftarrow \Pi$ which results trivial.
- $\mathrm{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} qVal(W')\theta' \Leftarrow \Pi$. By construction of θ', this is equivalent to prove $\mathrm{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} qVal(\imath(d)) \Leftarrow \Pi$. We trivially have that $\imath(d) \in \mathrm{ran}(\imath)$. Then, by Lemma 4.2, this premise holds.
- $\mathrm{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} qVal(w'_j)\theta' \Leftarrow \Pi$ for $j = 1 \ldots m$. By construction of θ' and Lemma 4.2 we must prove, for any fixed j, that $\mathsf{qVal}(w'_j\rho_j)$ is true in $\mathcal{C}$. As $\rho_j \in \mathrm{Sol}_{\mathcal{C}}(\Omega_j)$ we know $\rho_j \in \mathrm{Sol}_{\mathcal{C}}(\mathsf{qVal}(w'_j))$, therefore $\mathsf{qVal}(w'_j\rho_j)$ is trivially true in $\mathcal{C}$.
- $\mathrm{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} \ulcorner w'_j \trianglerighteq^{?} \imath(w_j) \urcorner\theta' \Leftarrow \Pi$ for $j = 1 \ldots m$. We reason for any fixed j. If $w_j = ?$ this results trivial. Otherwise, it amounts to $\mathsf{qBound}(\imath(w_j), \imath(\mathbf{t}), w'_j\rho_j)$ being true in $\mathcal{C}$, by construction of θ' and Lemma 4.2. As seen before, $\mathsf{qVal}(w'_j\rho_j)$ is true in $\mathcal{C}$, therefore $w'_j\rho_j = \imath(e'_j)$ for some $e'_j \in D \setminus \{\mathbf{b}\}$. From the side conditions of (♣) we have $w_j \trianglelefteq e_j$. On the other hand, $\rho_j \in \mathrm{Sol}_{\mathcal{C}}(\Omega_j)$ and, in particular, $\rho_j \in \mathrm{Sol}_{\mathcal{C}}(\mathsf{qBound}(\imath(e_j), \imath(\mathbf{t}), w'_j))$. This, together with $w'_j\rho_j = \imath(e'_j)$, means $e_j \trianglelefteq e'_j$, which with $w_j \trianglelefteq e_j$ implies $w_j \trianglelefteq e'_j$, i.e. $\mathsf{qBound}(\imath(w_j), \imath(\mathbf{t}), w'_j\rho_j)$ is true in $\mathcal{C}$.
- $\mathrm{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} qBound(W', \imath(\alpha), w'_j)\theta' \Leftarrow \Pi$ for $j = 1 \ldots m$. We reason for any fixed j. By construction of θ' and Lemma 4.2, we must prove that $\mathsf{qBound}(\imath(d), \imath(\alpha), w'_j\rho_j)$ is true in $\mathcal{C}$. As seen before, $\mathsf{qVal}(w'_j\rho_j)$ is true in $\mathcal{C}$, therefore $w'_j\rho_j = \imath(e'_j)$ for some $e'_j \in D \setminus \{\mathbf{b}\}$. From the side conditions of (♣) we have $d \trianglelefteq \alpha \circ e_j$. On the other hand, $\rho_j \in \mathrm{Sol}_{\mathcal{C}}(\Omega_j)$ and, in particular, $\rho_j \in \mathrm{Sol}_{\mathcal{C}}(\mathsf{qBound}(\imath(e_j), \imath(\mathbf{t}), w'_j))$. This, together with $w'_j\rho_j = \imath(e'_j)$, means $e_j \trianglelefteq e'_j$. Now, $d \trianglelefteq \alpha \circ e_j$ and $e_j \trianglelefteq e'_j$ implies $d \trianglelefteq \alpha \circ e'_j$, i.e. $\mathsf{qBound}(\imath(d), \imath(\alpha), w'_j\rho_j)$ is true in $\mathcal{C}$.
- $\mathrm{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} B'_j\theta' \Leftarrow \Pi$ for $j = 1 \ldots m$. In this case, it is easy to see that $B'_j\theta' = B'_j\theta\rho_j$ by construction of θ' and because of the program transformation rules. On the other hand, proof trees T'_j proving $\mathrm{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} B'_j\theta\rho_j \Leftarrow \Pi$ can be obtained by inductive hypothesis as seen before.

[2. ⇒ 1.] *(the transformation is sound).* We assume that T' is a a CHL($\mathcal{C}$) proof tree witnessing $\mathrm{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} A'\rho \Leftarrow \Pi$ for some $\rho \in \mathrm{Sol}_{\mathcal{C}}(\Omega)$ such that $\mathrm{dom}(\rho) = \mathrm{var}(\Omega)$. We want to to show the existence of a QCHL($\mathcal{D}, \mathcal{C}$) proof tree T witnessing $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} A\sharp d \Leftarrow \Pi$. We reason by complete induction on $\|T'\|$. There are three possible cases according to the the syntactic form of the atom A'. In each case we argue how to build the desired proof tree T.

— A' is a primitive atom κ. In this case due to **TQCA** and **TPA** we can assume $A = \kappa$ and $\Omega = \{\mathsf{qVal}(\imath(\mathbf{t})),\ \mathsf{qBound}(\imath(d), \imath(\mathbf{t}), \imath(\mathbf{t}))\}$. Note that $\mathrm{dom}(\rho) = \mathrm{var}(\Omega) = \emptyset$ implies $\rho = \varepsilon$. Now, from $\mathrm{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} \kappa\varepsilon \Leftarrow \Pi$ follows $\Pi \models_{\mathcal{C}} \kappa$ due to the **PA** inference, and therefore we can prove $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} \kappa\sharp d \Leftarrow \Pi$ with a proof tree T containing only one **QPA** node.

— A' is an equation $t == s$. In this case due to **TQCA** and **TEA** we can assume

$A = (t == s)$ and $\Omega = \{\mathsf{qVal}(\imath(\mathbf{t})),\ \mathsf{qBound}(\imath(d), \imath(\mathbf{t}), \imath(\mathbf{t}))\}$. Note that $\mathrm{dom}(\rho) = \mathrm{var}(\Omega) = \emptyset$ implies $\rho = \varepsilon$. Now, from $\mathrm{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} (t == s)\varepsilon \Leftarrow \Pi$ follows $t \approx_\Pi s$ due to the **EA** inference, and therefore we can prove $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} (t == s)\sharp d \Leftarrow \Pi$ with a proof tree T containing only one **QEA** node.

— A' is a defined atom $p'(\overline{t'}_n, W)$ with $p' \in DP^{n+1}$. In this case due to **TQCA** and **TDA** we can assume $A = p(\overline{t'}_n)$ and $\Omega = \{\mathsf{qVal}(W),\ \mathsf{qBound}(\imath(d), \imath(\mathbf{t}), W)\}$. On the other hand, T' must be rooted by a **DA** step (♠) using a clause $C^{\mathcal{T}} \in \mathrm{elim}_{\mathcal{D}}(\mathcal{P})$ instantiated by a substitution θ'. We can assume that (♠), $C^{\mathcal{T}}$ and the corresponding clause $C \in \mathcal{P}$ have the form already displayed in [1. ⇒ 2.].

By construction of $C^{\mathcal{T}}$, we can assume $B'^{\mathcal{T}}_j = (B'_j,\ w'_j)$. Let $\theta = \theta' \restriction \mathrm{var}(C)$ and $\rho_j = \theta' \restriction \mathrm{var}(w'_j)$ $(1 \geq j \geq m)$. Then, due to the premises $qVal(w'_j)\theta' \Leftarrow \Pi$ of (♠) and Lemma 4.2 we can assume $e'_j \in D \setminus \{\mathbf{b}\}$ $(1 \leq j \leq m)$ such that $w'_j\rho_j = \imath(e'_j)$.

To finish the proof, we must prove $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} A\sharp d \Leftarrow \Pi$. We claim that this can be done with a QCHL$(\mathcal{D},\mathcal{C})$ proof tree T whose root inference is a **QDA** step of the form of (♣), as displayed in [1. ⇒ 2.], using clause C instantiated by θ. In the premises of this inference we choose $d_i = \mathbf{t}$ $(1 \leq i \leq n)$ and $e_j = e'_j$ $(1 \leq j \leq m)$. Next we check that these premises can be derived from $\mathcal{P}$ in QCHL$(\mathcal{D},\mathcal{C})$ and that the side conditions are fulfilled:

- $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} (t'_i == t_i\theta)\sharp d_i \Leftarrow \Pi$ for $i = 1 \ldots n$. This amounts to $t'_i \approx_\Pi t_i\theta$ which follows from the first n premises of (♠) given that $t'_i\rho = t'_i$ and $t_i\theta' = t_i\theta$.
- $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} B_j\theta\sharp e_j \Leftarrow \Pi$ for $j = 1 \ldots m$. From $B'^{\mathcal{T}}_j = (B'_j, w'_j)$ and due to rule **TQCA**, we have $((B_j\theta)\sharp e_j \Leftarrow \Pi)^{\mathcal{T}} = (B_j\theta \Leftarrow \Pi, \Omega_j)$ where $\Omega_j = \{\mathsf{qVal}(w'_j),\ \mathsf{qBound}(\imath(e_j), \imath(\mathbf{t}), w'_j)\}$. From the premises of (♠) and the fact that $B'_j\theta' = B'_j\theta\rho_j$ we know that $\mathrm{elim}_{\mathcal{D}}(\mathcal{P}) \vdash_{\mathcal{C}} B'_j\theta\rho_j \Leftarrow \Pi$ with a CHL$(\mathcal{C})$ proof tree T'_j such that $\|T'_j\| < \|T'\|$. Therefore $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} B_j\theta\sharp e_j \Leftarrow \Pi$ follows by inductive hypothesis provided that $\rho_j \in \mathrm{Sol}_{\mathcal{C}}(\Omega_j)$. In fact, due to the form of Ω_j, $\rho_j \in \mathrm{Sol}_{\mathcal{C}}(\Omega_j)$ holds iff $w'_j\rho_j = \imath(e'_j)$ for some e'_j such that $e_j \trianglelefteqslant e'_j$, which is the case because of the choice of e_j.
- $e_j \trianglerighteqslant^? w_j$ for $j = 1 \ldots m$. Trivial in the case that $w_j = ?$. Otherwise they are equivalent to $w_j \trianglelefteqslant e'_j$ which follow from premises $\ulcorner w'_j \trianglerighteqslant^? \imath(w_j)\urcorner\theta' \Leftarrow \Pi$ (i.e. $\ulcorner w'_j\rho_j \trianglerighteqslant^? \imath(w_j)\urcorner \Leftarrow \Pi$) of (♠) and Lemma 4.2.
- $d \trianglelefteqslant d_i$ for $i = 1 \ldots n$. Trivially hold due to the choice of $d_i = \mathbf{t}$.
- $d \trianglelefteqslant \alpha \circ e_j$ for $j = 1 \ldots m$. Note that $\rho \in \mathrm{Sol}_{\mathcal{C}}(\Omega)$ implies the existence of $d' \in D \setminus \{\mathbf{b}\}$ such that $\imath(d') = W\rho$ and $d \trianglelefteqslant d'$. On the other hand, $e_j = e'_j$ by choice. It suffices to prove $d' \trianglelefteqslant \alpha \circ e'_j$ for $j = 1 \ldots m$. Premises of (♠) and Lemma 4.2 imply that $\mathsf{qBound}(W'\theta', \imath(\alpha), w'_j\theta')$ is true in $\mathcal{C}$. Moreover, $W'\theta' = W\rho = \imath(d')$ because of another premise of (♠) and $w'_j\theta' = \imath(e'_j)$ as explained above. Therefore $\mathsf{qBound}(W'\theta', \imath(\alpha), w'_j\theta')$ amounts to $\mathsf{qBound}(\imath(d'), \imath(\alpha), \imath(e'_j))$ which guarantees $d' \trianglelefteqslant \alpha \circ e'_j$ $(1 \leq j \leq m)$. □

The goal transformation correctness is established by the next theorem, which will rely on the previous result:

Theorem 4.4
Let G be a goal for a QCLP$(\mathcal{D},\mathcal{C})$-program $\mathcal{P}$ such that *qVal* and *qBound* do not occur in G. Let $\mathcal{P}' = \mathrm{elim}_{\mathcal{D}}(\mathcal{P})$ and $G' = \mathrm{elim}_{\mathcal{D}}(G)$. Assume a $\mathcal{C}$-subtitution σ,

a mapping $\mu : \mathrm{war}(G) \to D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ and a satisfiable finite set of $\mathcal{C}$-constraints Π. Then, the following two statements are equivalent:

1. $\langle \sigma, \mu, \Pi \rangle \in \mathrm{Sol}_{\mathcal{P}}(G)$.
2. $\langle \theta, \Pi \rangle \in \mathrm{Sol}_{\mathcal{P}'}(G')$ for some θ that verifies the following requirements:
   (a) $\theta =_{\mathrm{var}(G)} \sigma$,
   (b) $\theta =_{\mathrm{war}(G)} \mu\imath$ and
   (c) $W\theta \in \mathrm{ran}(\imath)$ for each $W \in \mathrm{var}(G') \setminus (\mathrm{var}(G) \cup \mathrm{war}(G))$.

Proof
As explained in Subsection 3.1 the syntax of goals in QCLP($\mathcal{D}, \mathcal{C}$)-programs is the same as that of goals for SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-programs, which is described in Section 2. Therefore G, and G' due to rule **TG**, must have the following form:

$G : (\ B_j \sharp W_j,\ W_j \Rrightarrow^? \beta_j\ )_{j=1\ldots m}$
$G' : (\ qVal(W_j),\ \ulcorner W_j \Rrightarrow^? \imath(\beta_j) \urcorner,\ qVal(w'_j),\ qBound(W_j, \imath(\mathbf{t}), w'_j),\ B'_j\ )_{j=1\ldots m}$

with $B_j^{\mathcal{T}} = (B'_j, w'_j)$ $(1 \le j \le m)$. Note that, because of rule **TQCA**, we have $(B_j\sigma \sharp W_j\mu \Leftarrow \Pi)^{\mathcal{T}} = (B'_j\sigma \Leftarrow \Pi, \Omega_j)$ with $\Omega_j = \{\mathsf{qVal}(w'_j),\ \mathsf{qBound}(\imath(W_j\mu), \imath(\mathbf{t}), w'_j)\}$ for $j = 1 \ldots m$. We now prove each implication.

[1. $\Rightarrow$ 2.] Let $\langle \sigma, \mu, \Pi \rangle \in \mathrm{Sol}_{\mathcal{P}}(G)$. This means, by Definition 3.1, $W_j\mu \Rrightarrow^? \beta_j$ and $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} B_j\sigma \sharp W_j\mu \Leftarrow \Pi$ for $j = 1 \ldots m$. In these conditions, Theorem 4.3 guarantees $\mathcal{P}' \vdash_{\mathcal{C}} B'_j\sigma\rho_j \Leftarrow \Pi$ $(1 \le j \le m)$ for some $\rho_j \in \mathrm{Sol}_{\mathcal{C}}(\Omega_j)$ such that $\mathrm{dom}(\rho_j) = \mathrm{var}(\Omega_j)$. It is easy to see that $\mathrm{var}(G') \setminus (\mathrm{var}(G) \cup \mathrm{war}(G)) = \mathrm{var}(\Omega_1) \uplus \cdots \uplus \mathrm{var}(\Omega_m)$. Therefore it is possible to define a substitution θ verifying $\theta =_{\mathrm{var}(G)} \sigma$, $\theta =_{\mathrm{war}(G)} \mu\imath$ and $\theta =_{\mathrm{dom}(\rho_j)} \rho_j$ $(1 \le j \le m)$. Trivially, θ satisfies conditions 2.(a) and 2.(b). It also satisfies condition 2.(c) because for any j and any variable X such that $X \in \mathrm{var}(\Omega_j)$, we have a constraint $\mathsf{qVal}(X) \in \Omega_j$ implying, due to Lemma 4.2, $X\rho_j \in \mathrm{ran}(\imath)$ (because $\rho_j \in \mathrm{Sol}_{\mathcal{C}}(\Omega_j)$).

In order to prove $\langle \theta, \Pi \rangle \in \mathrm{Sol}_{\mathcal{P}'}(G')$ in the sense of Definition 3.2 we check the following items:

- By construction, θ is a $\mathcal{C}$-substitution.
- By the theorem's assumptions, Π is a satisfiable and finite set of $\mathcal{C}$-constraints.
- $\mathcal{P}' \vdash_{\mathcal{C}} A\theta \Leftarrow \Pi$ for every atom A in G'. Because of the form of G' we have to prove the following for any fixed j:
  - — $\mathcal{P}' \vdash_{\mathcal{C}} qVal(W_j)\theta \Leftarrow \Pi$. By construction of θ and Lemma 4.2, this amounts to $\mathsf{qVal}(\imath(W_j\mu))$ being true in $\mathcal{C}$, which is trivial consequence of $W_j\mu \in D \setminus \{\mathbf{b}\}$.
  - — $\mathcal{P}' \vdash_{\mathcal{C}} \ulcorner W_j \Rrightarrow^? \imath(\beta_j) \urcorner\theta \Leftarrow \Pi$. If $\beta_j = ?$ this becomes trivial. Otherwise, $W_j\theta = \imath(W_j\mu)$ by construction of θ, and by Lemma 4.2 it suffices to prove $\mathsf{qBound}(\imath(\beta_j), \imath(\mathbf{t}), \imath(W_j\mu))$ is true in $\mathcal{C}$. This follows from $W_j\mu \Rrightarrow^? \beta_j$, that is ensured by $\langle \sigma, \mu, \Pi \rangle \in \mathrm{Sol}_{\mathcal{P}}(G)$.
  - — $\mathcal{P}' \vdash_{\mathcal{C}} qVal(w'_j)\theta \Leftarrow \Pi$. By construction of θ and Lemma 4.2, this amounts to $\mathsf{qVal}(w'_j\rho_j)$ being true in $\mathcal{C}$, that is guaranteed by $\rho_j \in \mathrm{Sol}_{\mathcal{C}}(\Omega_j)$.
  - — $\mathcal{P}' \vdash_{\mathcal{C}} qBound(W_j, \imath(\mathbf{t}), w'_j)\theta \Leftarrow \Pi$. By construction of θ and Lemma 4.2, this amounts to $\mathsf{qBound}(\imath(W_j\mu), \imath(\mathbf{t}), w'_j\rho_j)$ being true in $\mathcal{C}$, that is also guaranteed by $\rho_j \in \mathrm{Sol}_{\mathcal{C}}(\Omega_j)$.

— $\mathcal{P}' \vdash_{\mathcal{C}} B'_j\theta \Leftarrow \Pi$. Note that, by construction of θ, $B'_j\theta = B'_j\sigma\rho_j$. On the other hand, ρ_j has been chosen above to verify $\mathcal{P}' \vdash_{\mathcal{C}} B'_j\sigma\rho_j \Leftarrow \Pi$.

[2. $\Rightarrow$ 1.] Let $\langle\theta, \Pi\rangle \in \mathrm{Sol}_{\mathcal{P}'}(G')$ and assume that θ verifies 2.(a), 2.(b) and 2.(c). In order to prove $\langle\sigma, \mu, \Pi\rangle \in \mathrm{Sol}_{\mathcal{P}}(G)$ in the sense of Definition 3.1 we must prove the following items:

- By the theorem's assumptions, σ is a $\mathcal{C}$-substitution, $\mu : \mathrm{war}(G) \to D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ and Π is a satisfiable finite set of $\mathcal{C}$-constraints.
- $W_j\mu \trianglerighteq^? \beta_j$. We reason for any fixed j. If $\beta_j = ?$ this results trivial. Otherwise, we have $\mathcal{P}' \vdash_{\mathcal{C}} \ulcorner W_j \trianglerighteq^? \imath(\beta_j)\urcorner\theta \Leftarrow \Pi$ which, by condition 2.(b) and Lemma 4.2 amounts to $\mathsf{qBound}(\imath(\beta_j), \imath(\mathbf{t}), \imath(W_j\mu))$ is true $\mathcal{C}$, i.e. $W_j\mu \trianglerighteq \beta_j$.
- $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} B_j\sigma\sharp W_j\mu \Leftarrow \Pi$ for $j = 1 \ldots m$. We reason for any fixed j. Let ρ_j be the restriction of θ to $\mathrm{var}(\Omega_j)$. Then, $\mathcal{P}' \vdash_{\mathcal{C}} B'_j\sigma\rho_j \Leftarrow \Pi$ follows from $\langle\theta, \Pi\rangle \in \mathrm{Sol}_{\mathcal{P}'}(G')$ and $B'_j\theta = B'_j\sigma\rho_j$. Therefore, $\mathcal{P} \vdash_{\mathcal{D},\mathcal{C}} B_j\sigma\sharp W_j\mu \Leftarrow \Pi$ follows from Theorem 5.3 provided that $\rho_j \in \mathrm{Sol}_{\mathcal{C}}(\Omega_j)$. By Lemma 4.2 and the form of Ω_j, $\rho_j \in \mathrm{Sol}_{\mathcal{C}}(\Omega_j)$ holds iff $\mathcal{P}' \vdash_{\mathcal{C}} qVal(w'_j\rho_j) \Leftarrow \Pi$ and $\mathcal{P}' \vdash_{\mathcal{C}} qBound(\imath(W_j\mu), \imath(\mathbf{t}), w'_j\rho_j) \Leftarrow \Pi$, which is true because $\langle\theta, \Pi\rangle \in \mathrm{Sol}_{\mathcal{P}'}(G')$ and construction of ρ_j. □

4.3 Solving SQCLP Goals

In this subsection we show that the transformations from the two previous subsections can be used to define abstract goal solving systems for SQCLP and arguing about their correctness. In the sequel we consider a given SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-program $\mathcal{P}$ and a goal G for $\mathcal{P}$ whose atoms are all relevant for $\mathcal{P}$. We also consider $\mathcal{P}' = \mathrm{elim}_{\mathcal{S}}(\mathcal{P})$, $G' = \mathrm{elim}_{\mathcal{S}}(G)$, $\mathcal{P}'' = \mathrm{elim}_{\mathcal{D}}(\mathcal{P}')$ and $G'' = \mathrm{elim}_{\mathcal{D}}(G')$. Due to the definition of both $\mathrm{elim}_{\mathcal{S}}$ and $\mathrm{elim}_{\mathcal{D}}$, we can assume:

$$
\begin{array}{ll}
G & : (\ A_i\sharp W_i,\ W_i \trianglerighteq^? \beta_i\ )_{i=1\ldots m} \\
G' & : (\ A^i_{\sim}\sharp W_i,\ W_i \trianglerighteq^? \beta_i\ )_{i=1\ldots m} \\
G'' & : (\ qVal(W_i),\ \ulcorner W_i \trianglerighteq^? \imath(\beta_i)\urcorner,\ qVal(w'_i),\ qBound(W_i, \imath(\mathbf{t}), w'_i),\ A'_i\ )_{i=1\ldots m} \\
 & \quad \text{where } A_i^{\mathcal{T}} = (A'_i, w'_i).
\end{array}
$$

We start by presenting an auxiliary result.

Lemma 4.3
Assume $\mathcal{P}$, G, $\mathcal{P}'$, G', $\mathcal{P}''$ and G'' as above. Let $\langle\sigma', \Pi\rangle \in \mathrm{Sol}_{\mathcal{P}''}(G'')$, $\nu \in \mathrm{Sol}_{\mathcal{C}}(\Pi)$ and $\theta = \sigma'\nu$. Then $\langle\theta, \Pi\rangle \in \mathrm{Sol}_{\mathcal{P}''}(G'')$. Moreover, $W\theta \in \mathrm{ran}(\imath)$ for every $W \in \mathrm{var}(G'') \setminus \mathrm{var}(G)$.[1]

Proof
Consider an arbitrary atom A'' occurring in G''. Because of $\langle\sigma', \Pi\rangle \in \mathrm{Sol}_{\mathcal{P}''}(G'')$ we have $\mathcal{P} \vdash_{\mathcal{C}} A''\sigma' \Leftarrow \Pi$. On the other hand, because of $\nu \in \mathrm{Sol}_{\mathcal{C}}(\Pi)$ we have $\emptyset \models_{\mathcal{C}} \Pi\nu$ and therefore also $\Pi \models_{\mathcal{C}} \Pi\nu$. This and Definition 3.1(4) of (Rodríguez-Artalejo and Romero-Díaz 2010b) ensure $A''\sigma' \Leftarrow \Pi \succcurlyeq_{\mathcal{C}} A''\sigma'\nu \Leftarrow \Pi$, i.e. $A''\sigma' \Leftarrow \Pi \succcurlyeq_{\mathcal{C}} A''\theta \Leftarrow \Pi$.

[1] Note that $\mathrm{war}(G) \subseteq \mathrm{var}(G'') \setminus \mathrm{var}(G)$.

This fact, $\mathcal{P}'' \vdash_{\mathcal{C}} A''\sigma' \Leftarrow \Pi$ and the Entailment Property for Programs in CLP($\mathcal{C}$) imply $\mathcal{P}'' \vdash_{\mathcal{C}} A''\theta \Leftarrow \Pi$. Therefore, $\langle \theta, \Pi \rangle \in \mathrm{Sol}_{\mathcal{P}''}(G'')$.

Consider now any $W \in \mathrm{var}(G'') \setminus \mathrm{var}(G)$. By construction of G'', one of the atoms occurring in G'' is $qVal(W)$. Then, due to $\langle \sigma'\Pi \rangle \in \mathrm{Sol}_{\mathcal{P}''}(G'')$ we have $\mathcal{P}'' \vdash_{\mathcal{C}} qVal(W\sigma') \Leftarrow \Pi$. Because of Lemma 3.1(1) this implies $\Pi \models_{\mathcal{C}} \mathsf{qVal}(W\sigma')$, i.e. $\mathrm{Sol}_{\mathcal{C}}(\Pi) \subseteq \mathrm{Sol}_{\mathcal{C}}(\mathsf{qVal}(W\sigma'))$. Since $\nu \in \mathrm{Sol}_{\mathcal{C}}(\Pi)$ we get $\nu \in \mathrm{Sol}_{\mathcal{C}}(\mathsf{qVal}(W\sigma'))$, i.e. $W\sigma'\nu \in \mathrm{ran}(\imath)$. Since $W\sigma'\nu = W\theta$, we are done. □

Next, we explain how to define an abstract goal solving system for SQCLP from a given abstract goal solving system for CLP.

Definition 4.4
Let CLP-AGSS be an abstract goal solving system for CLP($\mathcal{C}$) (in the sense of Definition 3.3). Then we define *SQCLP-AGSS* as an abstract goal solving system for SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$) that works as follows:

1. Given a goal G for the SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$)-program $\mathcal{P}$, consider $\mathcal{P}'$, G', $\mathcal{P}''$ and G'' as explained at the beginning of the subsection.
2. For each solution $\langle \sigma', \Pi \rangle$ computed by CLP-AGSS for G'', $\mathcal{P}''$ and for any $\nu \in \mathrm{Sol}_{\mathcal{C}}(\Pi)$, SQCLP-AGSS computes $\langle \sigma, \mu, \Pi \rangle$ where $\theta = \sigma'\nu$, $\sigma = \theta{\restriction}\mathrm{var}(G)$ and $\mu = \theta\imath^{-1}{\restriction}\mathrm{war}(G)$. Note that μ is well-defined thanks to Lemma 4.3. □

The next theorem ensures that SQCLP-AGSS is correct provided that CLP-AGSS is also correct. The proof relies on the semantic results of the two previous subsections.

Theorem 4.5
Assume that CLP-AGSS is correct (in the sense of Definition 3.3). Let SQCLP-AGSS be as in the previous definition. Then SQCLP-AGSS is correct in the sense of Definition 2.3.

Proof
We separately prove that SQCLP-AGSS is *sound* and *weakly complete*.

— *SQCLP-AGSS is sound.* Let $\langle \sigma, \mu, \Pi \rangle$ be an answer computed by SQCLP-AGSS for $G, \mathcal{P}$. We must prove that $\langle \sigma, \mu, \Pi \rangle \in \mathrm{Sol}_{\mathcal{P}}(G)$. By Definition 4.4 we can assume $\langle \sigma', \Pi \rangle \in \mathrm{Sol}_{\mathcal{P}''}(G'')$ and $\nu \in \mathrm{Sol}_{\mathcal{C}}(\Pi)$ such that $\sigma = \theta{\restriction}\mathrm{var}(G)$ and $\mu = \theta\imath^{-1}{\restriction}\mathrm{war}(G)$ with $\theta = \sigma'\nu$. Because of Lemma 4.3 we have $\langle \theta, \Pi \rangle \in \mathrm{Sol}_{\mathcal{P}''}(G'')$ and $W\theta \in \mathrm{ran}(\imath)$ for every $W \in \mathrm{var}(G'') \setminus \mathrm{var}(G)$. Note that:

- $\theta =_{\mathrm{var}(G')} \sigma$. This follows from $\mathrm{var}(G') = \mathrm{var}(G)$ and the construction of σ.
- $\theta =_{\mathrm{war}(G')} \mu\imath$. This follows from $\mathrm{war}(G') = \mathrm{war}(G)$ and $\theta =_{\mathrm{war}(G)} \mu\imath$, that is obvious from the construction of μ.
- $W\theta \in \mathrm{ran}(\imath)$ for each $W \in \mathrm{var}(G'') \setminus (\mathrm{var}(G') \cup \mathrm{war}(G'))$. This is a consequence of Lemma 4.3 since $\mathrm{var}(G'') \setminus (\mathrm{var}(G') \cup \mathrm{war}(G')) \subseteq \mathrm{var}(G'') \setminus \mathrm{var}(G')$ and $\mathrm{var}(G') = \mathrm{var}(G)$.

From the previous items and Theorem 4.4 we get $\langle \sigma, \mu, \Pi \rangle \in \mathrm{Sol}_{\mathcal{P}'}(G')$, which trivially implies $\langle \sigma, \mu, \Pi \rangle \in \mathrm{Sol}_{\mathcal{P}}(G)$ because of Theorem 4.2.

— *SQCLP-AGSS is weakly complete.* Let $\langle \eta, \rho, \emptyset \rangle \in \mathrm{GSol}_{\mathcal{P}}(G)$ be a ground solution for G w.r.t. $\mathcal{P}$. We must prove that it is subsumed—in the sense of Definition 2.2(3)—by some answer $\langle \sigma, \mu, \Pi \rangle$ computed by SQCLP-AGSS for $G, \mathcal{P}$.

By Theorem 4.2 we have that $\langle \eta, \rho, \emptyset \rangle$ is also a ground solution for G' w.r.t. $\mathcal{P}'$. In addition, by Theorem 4.4 $\langle \eta', \emptyset \rangle \in \mathrm{Sol}_{\mathcal{P}''}(G'')$ for some η' such that

- (1) $\eta' =_{\mathrm{var}(G')} \eta$,
- (2) $\eta' =_{\mathrm{war}(G')} \rho\imath$ and hence $\eta'(\imath^{-1}) =_{\mathrm{war}(G')} \rho$, and
- $W\eta' \in \mathrm{ran}(\imath)$ for each $W \in \mathrm{var}(G'') \setminus (\mathrm{var}(G') \cup \mathrm{war}(G'))$ (i.e. $w'_i\eta' \in \mathrm{ran}(\imath)$ for each $i = 1 \ldots m$ such that w'_i is a variable).

By construction of η', it is clear that $\langle \eta', \emptyset \rangle$ is ground. Now, by the weak completeness of CLP-AGSS, there is some computed answer $\langle \sigma', \Pi \rangle$ subsuming $\langle \eta', \emptyset \rangle$, therefore satisfying

- (3) there is some $\nu \in \mathrm{Sol}_{\mathcal{C}}(\Pi)$, and
- (4) $\eta' =_{\mathrm{var}(G'')} \sigma'\nu$.

Because of Definition 4.4 one can build a SQCLP-AGSS computed answer $\langle \sigma, \mu, \Pi \rangle$ as follows:

- (5) $\sigma = \sigma'\nu{\restriction}\mathrm{var}(G)$
- (6) $\mu = \sigma'\nu\imath^{-1}{\restriction}\mathrm{war}(G)$

We now check that $\langle \sigma, \mu, \Pi \rangle$ subsumes $\langle \eta, \rho, \emptyset \rangle$:

- $W_i\rho \trianglelefteq W_i\mu$ and even $W_i\rho = W_i\mu$ because:
$$W_i\rho =_{(2)} W_i\eta'(\imath^{-1}) =_{(4)} W_i\sigma'\nu(\imath^{-1}) =_{(6)} W_i\mu \ .$$
- $\nu \in \mathrm{Sol}_{\mathcal{C}}(\Pi)$ by (3) and, moreover, for any $X \in \mathrm{var}(G)$:
$$X\eta =_{(1)} X\eta' =_{(4)} X\sigma'\nu =_{(\dagger)} X\sigma'\nu\nu =_{(5)} X\sigma\nu$$
therefore $\eta =_{\mathrm{var}(G)} \sigma\nu$.
The step (†) is justified because $\nu \in \mathrm{Val}_{\mathcal{C}}$ implies $\nu = \nu\nu$. □

5 A Practical Implementation

This section is devoted to the more practical aspects of the SQCLP programming scheme and it is developed in three subsections: Subsection 5.1 explains what steps must be given when implementing a programming scheme like this and why the theoretic results presented in the previous sections—with special emphasis in those in Subsection 4.3—become useful for implementation. Subsection 5.2 introduces a prototype implementation and explains how to write programs and how to solve goals. Finally, in Subsection 5.3 we study the unavoidable overload introduced in the system by qualifications and proximity relations when comparing the execution of programs without any explicit use of such resources.

5.1 SQCLP over a CLP Prolog System

Assume an available CLP Prolog System, a SQCLP($\mathcal{S},\mathcal{D},\mathcal{C}$)-program $\mathcal{P}$ and a goal G for $\mathcal{P}$. Our purpose is to implement a goal solving system for SQCLP following Definition 4.4. We will examine each step in this schema, discussing the necessary implementation details for putting theory into practice.

The first step is to obtain the transformed programs $\mathcal{P}' = \text{elim}_{\mathcal{S}}(\mathcal{P})$ and $\mathcal{P}'' = \text{elim}_{\mathcal{D}}(\mathcal{P}')$; and the transformed goals $G' = \text{elim}_{\mathcal{S}}(G)$ and $G'' = \text{elim}_{\mathcal{D}}(G')$. According to Definition 4.2(3), $\mathcal{P}' = \text{elim}_{\mathcal{S}}(\mathcal{P})$ is of the form $EQ_{\mathcal{S}} \cup \hat{\mathcal{P}}_{\mathcal{S}}$, where $EQ_{\mathcal{S}}$ is obtained following Definition 4.1 and $\hat{\mathcal{P}}_{\mathcal{S}}$ is obtained following Definition 4.2(3,2). When implementing $EQ_{\mathcal{S}}$ a first difficulty arises, namely the implementation of $\sim \in DP^2$, which apparently requires one clause of the form:

$$u \sim u' \overset{\mathbf{t}}{\leftarrow} \text{pay}_{\lambda}\sharp?$$

for each pair $u, u' \in B_{\mathcal{C}}$ such that $\mathcal{S}(u,u') = \lambda \neq \mathbf{b}$, and one clause of the form:

$$c(\overline{X}_n) \sim c'(\overline{Y}_n) \overset{\mathbf{t}}{\leftarrow} \text{pay}_{\lambda}\sharp?,\ ((X_i \sim Y_i)\sharp?)_{i=1\ldots n}$$

for each pair $c, c' \in DC^n$ such that $\mathcal{S}(c,c') = \lambda \neq \mathbf{b}$. While this should obviously require an infinite number of clauses (because DC^n is infinite and $\mathcal{S}(c,c) = \mathbf{t} \neq \mathbf{b}$ for all $c \in DC^n$; and also $B_{\mathcal{C}}$ is infinite—in general—and $\mathcal{S}(u,u) = \mathbf{t} \neq \mathbf{b}$ for every $u \in B_{\mathcal{C}}$), in practice, it is enough to limit the number of clauses to the finite number of different basic values $u \in B_{\mathcal{C}}$ and constructors $c \in DC^n$ that can be found either in $\mathcal{P}$, G or $\mathcal{S}$.

A similar difficulty arises when codifying the clauses for predicates $\text{pay}_{\lambda} \in DP^0$, which according to Definition 4.1 there should be a clause of the form:

$$\text{pay}_{\lambda} \overset{\lambda}{\leftarrow}$$

in $EQ_{\mathcal{S}}$ for each $\lambda \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$. In this case, the solution is also similar because it suffices to generate enough pay_{λ} clauses for the finite $\lambda \in D_{\mathcal{D}} \setminus \{\mathbf{b}\}$ that can be found occurring either in the clauses of $\hat{\mathcal{P}}_{\mathcal{S}}$ or in the clauses implementing the predicate $\sim \in DP^2$.

The construction of $\hat{\mathcal{P}}_{\mathcal{S}}$, following Definition 4.2, presents no particular difficulties. For each clause $C : (p(\bar{t}_n) \overset{\alpha}{\leftarrow} \overline{B}) \in \mathcal{P}$ we will generate a finite set $\hat{\mathcal{C}}_{\mathcal{S}}$ of clauses, because the number of symbols p' such that $\mathcal{S}(p,p') = \lambda \neq \mathbf{b}$ will be also finite in practice. Finally, the construction of G' is merely the straightforward replacement of all the occurrences of '==' in G by '$\sim$'.

The transformation $\text{elim}_{\mathcal{D}}$ from QCLP($\mathcal{D},\mathcal{C}$) into CLP($\mathcal{C}$), is defined in Definition 4.3. $\mathcal{P}'' = \text{elim}_{\mathcal{D}}(\mathcal{P}')$ is obtained by incorporating the two clauses of the program $E_{\mathcal{D}}$ to the result of applying the transformation rules in Figure 5 to the QCLP($\mathcal{D},\mathcal{C}$)-program $\mathcal{P}'$. Applying the transformation rules is straightforward, but the codification of constraints $\mathsf{qVal}(X)$ and $\mathsf{qBound}(X,Y,Z)$ in $E_{\mathcal{D}}$ requires some clarification. In our implementation we have considered the constraint domain $\mathcal{R}$, as well as any qualification domain that can be built from $\mathcal{B}$, $\mathcal{U}$ and $\mathcal{W}$ by means of the strict cartesian product operation $\otimes$ including, in particular, $\mathcal{U}\otimes\mathcal{W}$. These qualification domains are existentially expressible in $\mathcal{R}$, therefore the constraints

can be implemented by defined predicates as explained in Section 4.2. In particular in our prototype implementation these predicates are:

```
% qval( +QDom, ?W ):
qval(b, 1).
qval(u, W) :- {W > 0, W =< 1}.
qval(w, W) :- {W >= 0}.
qval((D1,D2), (W1,W2)) :- qval(D1, W1), qval(D2, W2).

% qbound( +QDom, ?X, ?Y, ?Z ):
qbound(b, 1, 1, 1).
qbound(u, X, Y, Z) :- {X =< Y * Z}.
qbound(w, X, Y, Z) :- {X >= Y + Z}.
qbound((D1,D2), (X1,X2), (Y1,Y2), (Z1,Z2)) :- qbound(D1, X1, Y1, Z1),
    qbound(D2, X2, Y2, Z2).
```

Instead of using different *qVal* and *qBound* predicates for each allowable $\mathcal{D}$, our prototype implementation just uses two predicates *qVal* and *qBound* with an extra first argument, used to encode an identifier of some specific allowable $\mathcal{D}$. This parameter can take either the value `b` (for $\mathcal{B}$), `u` (for $\mathcal{U}$), `w` (for $\mathcal{W}$) or a pair `(`$\mathtt{D}_1$`,`$\mathtt{D}_2$`)` (for $\mathcal{D}_1 \otimes \mathcal{D}_2$), where each $\mathtt{D}_i$ can be either `b`, `u`, `w` or another pair representing a product. For instance `((u,w),w)` represents the qualification domain $(\mathcal{U}\otimes\mathcal{W})\otimes\mathcal{W}$. The compiler ensures that this argument takes the correct value for each transformed program and goal depending on the specific instance of the SQCLP scheme the program is written for.

After obtaining $\mathcal{P}''$ and G'', the CLP Prolog System is used to solve G'' w.r.t. $\mathcal{P}''$. This yields computed answers of the form $\langle \sigma', \Pi \rangle$. Now, instead of obtaining particular substitutions $\theta = \sigma'\nu$, $\sigma = \theta\restriction \mathrm{var}(G)$ and $\mu = \theta\imath^{-1}\restriction \mathrm{war}(G)$ for any $\nu \in \mathrm{Sol}_{\mathcal{C}}(\Pi)$ as explained in Definition 4.4(2), our prototype implementation limits itself to display $\langle \sigma', \Pi \rangle$ as the computed answer in SQCLP. The reason behind this behavior is that, in general (and particularly in $\mathcal{R}$), it is impossible to enumerate the possible solutions $\nu \in \mathrm{Sol}_{\mathcal{C}}(\Pi)$. Thus, it results impossible to implement a technique for obtaining all the possible triples $\langle \sigma, \mu, \Pi \rangle$. Note, however, that for a user it will not be difficult to distinguish, in the shown computed answers, what variable bindings correspond to the substitution σ of the triple and what to the substitution μ, even when the qualification variables are not bound but constrained, which is a common behavior in the context of CLP programming.

However, for the SQCLP-AGSS of Definition 4.4, it results mandatory to define the computed answers in terms of $\nu \in \mathrm{Sol}_{\mathcal{C}}(\Pi)$, because our SQCLP-semantics relies on proving instances of G for some specific ground values of the variables in $\mathrm{war}(G)$.

5.2 `(S)QCLP`*: A Prototype System for SQCLP Programming*

The prototype implementation object of this subsection is publicly available, and can be found at:

`http://gpd.sip.ucm.es/cromdia/qclp`

The system currently requires the user to have installed either *SICStus Prolog* or *SWI-Prolog*, and it has been tested to work under Windows, Linux and MacOSX platforms. The latest version available at the time of writing this paper is `0.6`. If a latter version is available some things might have changed but in any case the main aspects of the system should remain the same. Please consult the *changelog* provided within the system itself for specific changes between versions.

SQCLP is a very general programming scheme and, as such, it supports different proximity relations, different qualification domains and different constraint domains when building specific instances of the scheme for any specific purpose. As it would result impossible to provide an implementation for every admissible triple (or instance of the scheme), it becomes mandatory to decide in advance what specific instances will be available for writing programs in `(S)QCLP`. In essence:

1. In its current state, the only available constraint domain is $\mathcal{R}$. Thus, under both *SICStus Prolog* and *SWI-Prolog* the library `clpr` will provide all the available primitives in `(S)QCLP` programs.
2. The available qualification domains are: '`b`' for the domain $\mathcal{B}$; '`u`' for the domain $\mathcal{U}$; '`w`' for the domain $\mathcal{W}$; and any strict cartesian product of those, as e.g. '`(u,w)`' for the product domain $\mathcal{U}\otimes\mathcal{W}$.
3. With respect to proximity relations, the user will have to provide, in addition to the two symbols and their proximity value, their *kind* (either predicate or constructor) and their *arity*. Both kind and arity must be the same for each pair of symbols having a proximity value different of $\mathbf{b}$.

Note, however, that when no specific proximity relation $\mathcal{S}$ is provided for a given program, $\mathcal{S}_{\mathrm{id}}$ is then assumed. Under this circumstances, an obvious technical optimization consists on transforming the original program only with $\mathrm{elim}_{\mathcal{D}}$, thus reducing the overload introduced in this case by $\mathrm{elim}_{\mathcal{S}}$. The reason behind this optimization is that for any given SQCLP$(\mathcal{S}_{\mathrm{id}},\mathcal{D},\mathcal{C})$-program $\mathcal{P}$, it is also true that $\mathcal{P}$ is a QCLP$(\mathcal{D},\mathcal{C})$-program, therefore $\mathrm{elim}_{\mathcal{D}}(\mathrm{elim}_{\mathcal{S}}(\mathcal{P}))$ must semantically be equivalent to $\mathrm{elim}_{\mathcal{D}}(\mathcal{P})$. Nevertheless, $\mathrm{elim}_{\mathcal{D}}(\mathcal{P})$ behaves more efficiently than $\mathrm{elim}_{\mathcal{D}}(\mathrm{elim}_{\mathcal{S}}(\mathcal{P}))$ due to the reduced number of resulting clauses. Thus, in order to improve the efficiency, the system will avoid the use of $\mathrm{elim}_{\mathcal{S}}$ when no proximity relation is provided by the user.

The final available instances in the `(S)QCLP` system are: SQCLP$(\mathcal{S},\mathtt{b},\mathtt{clpr})$, SQCLP$(\mathcal{S},\mathtt{u},\mathtt{clpr})$, SQCLP$(\mathcal{S},\mathtt{w},\mathtt{clpr})$, SQCLP$(\mathcal{S},(\mathtt{u},\mathtt{w}),\mathtt{clpr})$, ... and their counterparts in the QCLP scheme when $\mathcal{S}=\mathcal{S}_{\mathrm{id}}$.

5.2.1 Programming in `(S)QCLP`

Programming in `(S)QCLP` is straightforward if the user is accustomed to the Prolog programming style. However, there are three syntactic differences with pure Prolog:

1. Clauses implications are replaced by "`<-`d`-`" where $d \in D \setminus \{\mathbf{b}\}$. If $d = \mathbf{t}$, then the implication can become just "`<--`". E.g. "`<-0.9-`" is a valid implication in the domains $\mathcal{U}$ and $\mathcal{W}$; and "`<-(0.9,2)-`" is a valid implication in the domain $\mathcal{U}\otimes\mathcal{W}$.

2. Clauses in (S)QCLP are not finished with a dot (.). They are separated by layout, therefore all clauses in a (S)QCLP program must start in the same column. Otherwise, the user will have to explicitly separate them by means of semicolons (;).
3. After every body atom (even constraints) the user can provide a threshold condition using '#'. The notation '?' can also be used instead of some particular qualification value, but in this case the threshold condition '#?' can be omitted.

Comments are as in Prolog:

```
% This is a line comment.
/* This is a multi-line comment, /* and they nest! */. */
```

and the basic structure of a (S)QCLP program is the following (line numbers are for reference):

File: *Peano.qclp*

```
1  % Directives...
2  # qdom w

3  % Program clauses...
4  % num( ?Num )
5  num(z) <--
6  num(s(X)) <-1- num(X)
```

In the previous small program, lines 1, 3 and 4 are line comments, line 2 is a program directive telling the compiler the specific qualification domain the program is written for, and lines 5 and 6 are program clauses defining the well-known Peano numbers. As usual, comments can be written anywhere in the program as they will be completely ignored (remember that a line comment must necessarily end in a new line character, therefore the very last line of a file cannot contain a line comment), and directives must be declared before any program clause. There are three program directives in (S)QCLP:

1. The first one is "#qdom *qdom*" where *qdom* is any system available qualification domain, i.e. b, u, w, (u,w)... See line 2 in the previous program sample as an example. This directive is mandatory because the user must tell the compiler for which particular qualification domain the program is written.
2. The second one is "#prox *file*" where *file* is the name of a file (with extension .prox containing a proximity relation. If the name of the file starts with a capital letter, or it contains spaces or any special character, *file* will have to be quoted with single quotes. For example, assume that with our program file we have another file called *Proximity.prox*. Then, we would have to write "#prox 'Proximity'" to link the program with such proximity relation. This directive is optional, and if omitted, the system assumes that the program is of an instance of the QCLP scheme.
3. The third one is "#optimized_unif". This directive tells the compiler that the program is intended to be used with the optimized version of the unification algorithm, what improves the general efficiency of the goal solving

process. However, as noted at the end of Section 2, this could have the effect of losing valid answers, although we conjecture that if the proximity relation is transitive and if the program clauses do not make use of attenuation factors other that **t**, this will not happen.

Proximity relations are defined in files of extension `.prox` with the following form:

File: *Work.prox*

```
1  % Predicates: pprox( S1, S2, Arity, Value ).
2  pprox(wrote, authored, 2, (0.9,0)).
3  % Constructors: cprox( S1, S2, Arity, Value ).
4  cprox(king_lear, king_liar, 0, (0.8,2)).
```

where the file can contain `pprox/4` Prolog facts, for defining proximity between predicate symbols of any arity; or `cprox/4` Prolog facts, for defining proximity between constructor symbols of any arity. The arguments of both `pprox/4` and `cprox/4` are: the two symbols, their arity and its proximity value. Note that, although it is not made explicit the qualification domain this proximity relation is written for, all values in it must be of the same specific qualification domain, and this qualification domain must be the same declared in every program using the proximity relation. Otherwise, the solving of equations may produce unexpected results or even fail.

Reflexive and symmetric closure is inferred by the system, therefore, there is no need for writing reflexive proximity facts, nor the symmetric variants of proximity facts already provided. You can notice this in the previous sample file in which neither reflexive proximity facts, nor the symmetric proximity facts to those at lines 2 and 4 are provided. In the case of being explicitly provided, additional (repeated) solutions might be computed for the same given goal, although soundness and weak completeness of the system should still be preserved. Transitivity is neither checked nor inferred so the user will be responsible for ensuring it if desired.

As the reader would have already guessed, the file `Work.prox` implements the proximity relation $\mathcal{S}_r$ of Example 4.1 in `(S)QCLP`. Finally, the program $\mathcal{P}_r$ of Example 4.1 can be represented in `(S)QCLP` as follows:

File: *Work.qclp*

```
1  # qdom (u,w)
2  # prox 'Work'
3  % famous( ?Author )
4  famous(shakespeare) <-(0.9,1)-
5  % wrote( ?Author, ?Book )
6  wrote(shakespeare, king_lear) <-(1,1)-
7  wrote(shakespeare, hamlet) <-(1,1)-
8  % good_work( ?Work )
9  good_work(X) <-(0.75,3)- famous(Y)#(0.5,100), authored(Y,X)
```

Note that, at line 1 the qualification domain $\mathcal{U}\otimes\mathcal{W}$ is declared, and at line 2 the proximity relation at `Work.prox` is linked to the program. In addition, observe

that one threshold constraint is imposed for a body atom in the program clause at line `9`, effectively requiring to prove `famous(Y)` for a qualification value of *at least* `(0.5,100)` to be able to use this program clause.

Finally, we explain how constraints are written in `(S)QCLP`. As it has already been said, only $\mathcal{R}$ is available, thus both in *SICStus Prolog* and *SWI-Prolog* the library `clpr` is the responsible for providing the available primitive predicates. Given that constraints are primitive atoms of the form $\mathtt{r}(\overline{\mathtt{t}}_n)$ where $\mathtt{r} \in PP^n$ and $\mathtt{t}_i$ are terms; primitive atoms share syntax with usual Prolog atoms. At this point, and having that many of the primitive predicates are syntactically operators (hence not valid identifiers), the syntax for predicate symbols has been extended to include operators, therefore predicate symbols like $op_+ \in PP^3$, which codifies the operation `+` in a 3-ary predicate, will let us to build constraints of the form `+(A,B,C)`, that must be understood as in $A + B = C$ or $C = A + B$. Similarly, predicate symbols like $cp_> \in PP^2$, which codifies the comparison operator `>` in a binary predicate, will let us to build constraints of the form `>(A,B)`, that must be understood as in $A > B$. Any other primitive predicate such as *maximize* $\in PP^1$, will let us to build constraints like `maximize(X)`. Valid primitive predicate symbols include `+`, `-`, `*`, `/`, `>`, `>=`, `=<`, `<`, `maximize`, `minimize`, etc.

Threshold constraints can also be provided for primitive atoms in the body of clauses with the usual notation. Note, however, that due the semantics of SQCLP, all primitive atoms can be trivially proved with `t` if they ever succeeds—so threshold constraints become, in this case, of no use.

The syntax for constraints explained above follows the standard syntax for atoms. Nonetheless, the system also allows to write these constraints in a more natural infix notation. More precisely, `+(A,B,C)` can be also written in the infix form `A+B=C` or `C=A+B`, and `>(X,Y)` in the infix form `X>Y`; and similarly for other *op* and *cp* constraints. When using infix notation, threshold conditions can be set by (optionally) enclosing the primitive atom between parentheses, therefore becoming `(A+B=C)#t`, `(C=A+B)#t` or `(X>Y)#t` (or any other valid qualification value or '?'). Using parentheses is recommended to avoid understanding that the threshold condition is set only for the last term in the constraint, which would not be the case. Note that even in infix notation, operators cannot be nested, that is, terms `A`, `B`, `C`, `X` and `Y` cannot have operators as main symbols (neither in prefix nor in infix notation), so the infix notation is just a syntactic sugar of its corresponding prefix notation.

As a final example for constraints, one could write the predicate `double/2` in `(S)QCLP`, for computing the double of any given number, with just the clause `double(N,D) <-- *(N,2,D)`, or `double(N,D) <-- N*2=D` for a clause with a more natural syntax.

5.2.2 The interpreter for `(S)QCLP`

The interpreter for `(S)QCLP` has been implemented on top of both *SICStus Prolog* and *SWI-Prolog*. To load it, one must first load her desired (and supported) Prolog system and then load the main file of the interpreter—i.e. `qclp.pl`—, that will be located in the main `(S)QCLP` folder among other folders. Once loaded, one will

see the welcome message and will be ready to compile and load programs, and to execute goals.

```
WELCOME TO (S)QCLP 0.6
(S)QCLP is free software and comes with absolutely no warranty.
Support & Updates: http://gpd.sip.ucm.es/cromdia/qclp.

Type ':help.' for help.
yes
| ?-
```

From the interpreter for `(S)QCLP` one can, in addition to making use of any standard Prolog goals, use the specific `(S)QCLP` commands required for both interacting with the `(S)QCLP` system, and for compiling/loading SQCLP programs. All these commands take the form:

`:command.`

if they do not require arguments, or:

`:command(`Arg_1`, ..., `Arg_n`).`

if they do; where each argument Arg_i must be a prolog atom unless stated otherwise. The most useful commands are:

- `:cd(`*Folder*`).`
  Changes the working directory to *Folder*. *Folder* can be an absolute or relative path.
- `:compile(`*Program*`).`
  Compiles the `(S)QCLP` program '*Program*`.qclp`' producing the equivalent Prolog program in the file '*Program*`.pl`'.
- `:load(`*Program*`).`
  Loads the already compiled `(S)QCLP` program '*Program*`.qclp`' (note that the file '*Program*`.pl`' must exist for the program to correctly load).
- `:run(`*Program*`).`
  Compiles the `(S)QCLP` program '*Program*`.qclp`' and loads it afterwards. This command is equivalent to executing: `:compile(`*Program*`), :load(`*Program*`).`

For illustration purposes, we will assume that you have the files `Work.prox` and `Work.qclp` (both as seen before) in the folder `~/examples`. Under these circumstances, after loading your preferred Prolog system and the interpreter for `(S)QCLP`, one would only have to change the working directory to that where the files are located:

```
| ?- :cd('~/examples').
```

and run the program:

```
| ?- :run('Work').
```

If no errors are encountered, one should see the output:

```
| ?- :run('Work').
<Work> Compiling...
<Work> QDom: 'u,w'.
<Work> Prox: 'Work'.
<Work> Translating to QCLP...
<Work> Translating to CLP...
<Work> Generating code...
<Work> Done.
<Work> Loaded.
yes
```

and now everything is ready to execute goals for the program loaded.

5.2.3 Executing SQCLP-Goals

Recall that goals have the form $A_1 \sharp W_1,\ \ldots,\ A_m \sharp W_m \ [] \ W_1 \trianglerighteq^? \beta_1,\ \ldots,\ W_m \trianglerighteq^? \beta_m$ which in actual `(S)QCLP` syntax becomes:

```
| ?- A1#W1, ..., Am#Wm :: W1 >= B1, ..., Wm >= Bm.
```

Note the following:

1. Goals must end in a dot (`.`).
2. The symbol '`[]`' is replaced by '`::`'.
3. The symbol '$\trianglerighteq^?$' is replaced by '`>=`' (and this is independent of the qualification domain in use, so that it may mean $\leq$ in $\mathcal{W}$).
4. Conditions of the form $W \trianglerighteq^?$? *must be omitted*, therefore $A_1 \sharp W_1, A_2 \sharp W_2 \ [] \ W_1 \trianglerighteq^?$ $?, W_2 \trianglerighteq^? \beta_2$ becomes "`A1#W1, A2#W2 :: W2 >= B2.`", and $A \sharp W \ [] \ W \trianglerighteq^?$? becomes just "`A#W.`".

Assuming now that we have loaded the program `Work.qclp` as explained before, we can execute the goal $good\_work(king\_liar) \sharp W \ [] \ W \trianglerighteq^? (0.5, 100)$:

```
| ?- good_work(king_liar)#W::W>=(0.5,10).
W = (0.6,5.0) ?
yes
```

5.2.4 Examples

To finish this subsection, we are now showing some additional goal executions using the interpreter for `(S)QCLP` and the programs displayed along the paper.

Peano. Consider the program `Peano.qclp` as displayed at the beginning of Subsection 5.2.1. Qualifications in this program are intended as a cost measure for obtaining a given number in the Peano representation, assuming that each use of the clause at line `6` requires to pay *at least* `1`. In essence, threshold conditions will impose an upper bound over the maximum number obtainable in goals containing the atom `num(X)`. Therefore if we ask for numbers *up to* a cost of `3` we get the following answers:

```
Goal   ?- num(X)#W::W>=3.

Sol1   W = 0.0, X = z ? ;
Sol2   W = 1.0, X = s(z) ? ;
Sol3   W = 2.0, X = s(s(z)) ? ;
Sol4   W = 3.0, X = s(s(s(z))) ? ;
       no
```

Work. Consider now the program `Work.qclp` and the proximity relation `Work.prox`, both as displayed in Subsection 5.2.1 above. In this program, qualifications behave as the conjunction of the certainty degree of the user confidence about some particular atom, and a measure of the minimum cost to pay for proving such atom. In these circumstances, we could ask—just for illustration purposes—for famous authors with a minimum certainty degree—for them being actually famous—of `0.5`, and with a proof cost of no more than `30` (think of an upper bound for possible searches in different databases). Such a goal would have, in this very limited example, only the following solution:

```
Goal   ?- famous(X)#W::W>=(0.5,30).

Sol1   W = (0.9,1.0), X = shakespeare ? ;
       no
```

meaning that we can have a confidence of `shakespeare` being famous of `0.9`, and that we can prove it with a cost of `1`.

Now, in a similar fashion we could try to obtain different works that can be considered as good works by using the last clause in the example. Limiting the search to those works that can be considered good with a qualification value better or equal to `(0.5,100)` produce the following result:

```
Goal   ?- good_work(X)#W::W>=(0.5,100).

Sol1   W = (0.675,4.0), X = king_lear ? ;
Sol2   W = (0.6,5.0), X = king_liar ? ;
       no
```

It is important to remark here that the qualification value obtained for a particular computed answer is not guaranteed to be the best possible one; rather, different computed answers may compute different qualification values which can be observed by the user. This is easy to see if we try to solve a more particular goal:

```
Goal   ?- good_work(king_liar)#W::W>=(0.675,4.0).

Sol1   W = (0.675,4.0) ? ;
       no
```

That is, not only `good_work(king_liar)` can be proved for for `W = (0.6,5.0)` as shown in Sol_2 above, but also with `W = (0.675,4.0)`, which results a better qualification value (i.e. greater certainty degree and lower proof cost).

Library. Finally, consider the program $\mathcal{P}_s$ and the proximity relation $\mathcal{S}_s$, both as displayed in Figure 1 of Section 2. As it has been said when this example was introduced, the predicate *guessRdrLvl* takes advantage of attenuation factors to

encode heuristic rules to compute reader levels on the basis of vocabulary level and other book features. As an illustration of use, consider the following goal:

```
Goal   ?- guessRdrLvl(book(2, 'Dune', 'F. P. Herbert', english, sciFi,
          medium, 345), Level)#W.
Sol1   W = 0.8, Level = intermediate ? ;
       ...
Sol6   W = 0.7, Level = upper ?
       yes
```

Here we ask for possible ways of classifying the second book in the library according to reader levels. We obtain as valid solutions, among others, `intermediate` with a certainty factor of `0.8`; and `upper` with a certainty factor of `0.7`. These valid solutions show that the predicate *guessRdrLvl* tries with different levels for any certain book based on the heuristic implemented by the qualified clauses.

To conclude, consider now the goal proposed in Section 2 for this program. For such goal we obtain:

```
Goal   ?- search(german, essay, intermediate, ID)#W::W>=0.65.
Sol1   W = 0.8, ID = 4 ?
       yes
```

What tells us that the forth book in the library is written in German, it can be considered to be an essay, and it is targeted for an intermediate reader level. All this with a certainty degree of *at least* `0.8`.

5.3 Efficiency

The minimum—and unavoidable—overload introduced by qualifications and proximity relations in the transformed programs manifests itself in the case of `(S)QCLP` programs which use the identity proximity relation and have **t** as the attenuation factor of all their clauses. In order to measure this overload we have made some experiments using some program samples, taken from the *SICStus Prolog Benchmark* that can be found in:

`http://www.sics.se/isl/sicstuswww/site/performance.html`

and we have compared the time it took to repeatedly execute a significant number of times each program in both `(S)QCLP` and *SICStus Prolog* making use of a *slightly* modified (to ensure a correct behavior in both systems) version of the harness also provided in the same site.

From all the programs available in the aforementioned site, we selected the following four:

- *naivrev:* naive implementation of the predicate that reverses the contents of a list.
- *deriv:* program for symbolic derivation.
- *qsort:* implementation of the well-known sorting algorithm *Quicksort*.
- *query:* obtaining the population density of different countries.

No other program could be used because they included impure features such as cuts which are not currently supported by our system. In order to adapt these Prolog programs to our setting the following modifications were required:

1. All the program clause are assumed to have `t` as attenuation factor. After including these attenuation factors, we obtain as results QCLP programs. More specifically we obtain two QCLP programs for each initial Prolog program, one using the qualification domain $\mathcal{B}$ (because this domain uses trivial constraints), and another using the qualification domain $\mathcal{U}$ (which uses $\mathcal{R}$-constraints).
2. We define an empty proximity relation, allowing us to obtain two additional SQCLP-programs.
3. By means of the program directive "`#optimized_unif`" defined in Subsection 5.2.1, each SQCLP program can be also executed in this optimized mode. Therefore each original Prolog Program produces six `(S)QCLP` programs, denoted as Q(b), Q(u), PQ(b), PQ(u), SQ(b) and SQ(u) in Table 1.

Additionally some minor modifications to the program samples have been introduced for compatibility reasons, i.e. additions using the predicate `is/2` were replaced, both in the Prolog version of the benchmark and in the multiple `(S)QCLP` versions, by `clpr` constraints. In any case, all the program samples used for this benchmarks in this subsection can be found in the folder `benchmarks/` of the `(S)QCLP` distribution.

Finally, we proceeded to solve the same goals for every version of the benchmark programs, both in *SICStus Prolog* and in `(S)QCLP`. The benchmark results can be found in Table 1. All the experiments were performed in a computer with a Intel(R) Core(TM)2 Duo CPU at 2.19GHz and with 3.5 GB RAM.

Table 1. Time overload factor with respect to Prolog

Program	Q(b)[a]	Q(u)[b]	PQ(b)[c]	PQ(u)[d]	SQ(b)[e]	SQ(u)[f]
naivrev	1.80	10.71	4289.79	4415.11	56.22	65.75
deriv	1.94	10.60	331.45	469.67	29.63	39.32
qsort	1.05	1.11	135.59	136.98	2.51	2.83
query	1.02	1.12	7.17	7.13	3.80	3.88

[a] QCLP($\mathcal{B},\mathcal{R}$) version (i.e. the program does not have the `#prox` directive).
[b] QCLP($\mathcal{U},\mathcal{R}$) version (i.e. the program does not have the `#prox` directive).
[c] SQCLP($\mathcal{S}_{\mathrm{id}},\mathcal{B},\mathcal{R}$) version.
[d] SQCLP($\mathcal{S}_{\mathrm{id}},\mathcal{U},\mathcal{R}$) version.
[e] SQCLP($\mathcal{S}_{\mathrm{id}},\mathcal{B},\mathcal{R}$) version with directive `#optimized_unif`.
[f] SQCLP($\mathcal{S}_{\mathrm{id}},\mathcal{U},\mathcal{R}$) version with directive `#optimized_unif`.

The results in the table indicate the slowdown factor obtained for each version of each program. For instance, the first column indicates that the time required for evaluating the goal corresponding to the sample program *naivrev* in QCLP($\mathcal{B},\mathcal{R}$) is about 1.80 times the required time for the evaluation of the same goal in Prolog. Next we discuss the results:

- *Influence of the qualification domain.* In general the difference between the slowdown factors obtained for the two considered qualification domains is not large. However, in the case of QCLP-programs *naivrev* and *deriv* the difference increases notably. This is due to the different ratios of the $\mathcal{B}$-constraints w.r.t. the program and $\mathcal{U}$-constraints w.r.t. the program. It must be noticed that the transformed programs are the same in both cases, but for the implementation of `qval` and `qbound` constraints, which is more complex for $\mathcal{U}$ as one can see in Section 5.1. In the case of *naivrev* and *deriv* this makes a big difference because the number of computation steps directly required by the programs is much smaller than in the other cases. Thus the slowdown factor becomes noticeable for the qualification domain $\mathcal{U}$ in computations that requires a large number of steps.
- *Influence of the proximity relation.* The introduction of a proximity relation, even of empty, is very significative. This is due to the introduction of the predicate $\sim$, which replaces Prolog unification. The situation even worsens when the computation introduces large constructor terms, as in the case of *naivrev* which deals with Prolog lists. The efficient Prolog unification is replaced by an explicit term decomposition.
- *Influence of the optimized unification.* As explained at the end of Section 2 this optimization can lead to the loss of solutions in general. However, this is not the case for the chosen examples. As seen in the table, the use of the program directive `#optimized_unif` causes a clear increase in the efficiency of goal solving for these examples.

6 Conclusions

In our recent work (Rodríguez-Artalejo and Romero-Díaz 2010a) we extended the classical CLP scheme to a new programming scheme SQCLP whose instances SQCLP($\mathcal{S}, \mathcal{D}, \mathcal{C}$) were parameterized by a proximity relation $\mathcal{S}$, a qualification domain $\mathcal{D}$ and a constraint domain $\mathcal{C}$. This new scheme offered extra facilities for dealing with expert knowledge representation and flexible query answering. In this paper we have contributed to the aforementioned scheme providing, in a more practical sense, both a semantically correct transformation technique, in two steps, for transforming SQCLP programs and goals intro equivalent CLP programs and goals; and a prototype implementation on top CLP($\mathcal{R}$) systems like *SICStus Prolog* and *SWI-Prolog* of some particularly interesting instances of the scheme.

The two-step transformation technique presented in Section 4 has provided us with the needed theoretical results for effectively showing how proximity relations can be reduced to qualifications and clause annotations by means of the transformation $\text{elim}_{\mathcal{S}}$; and how qualifications and clause annotations can be reduced to classical CLP programming by means of the transformation $\text{elim}_{\mathcal{D}}$. These two transformations altogether, ultimately enables the use of the classical mechanism of SLD resolution to obtain computed answers for SQCLP goals w.r.t SQCLP programs, via their equivalent CLP programs and goals and the computed answers obtained from them by any capable CLP goal solving procedure.

The prototype implementation presented in Section 5 has finally allowed us to execute all the examples showed in this paper—and in previous ones—, and a series of benchmarks for measuring the overload actually introduced by proximity relations—or by similarity relations—and by clause annotations and qualifications. While we are aware that the prototype implementation presented in this paper has to be considered a research application (and as such, we have to admit that it cannot be used for industrial applications), we think that it can contribute to the field as a quite complete implementation of an extension of the CLP($\mathcal{R}$) scheme with proximity relations and qualifications. Some related implementation techniques and systems have been cited in the introduction. However, as far as we know, no other implementation in this field has ever provided support for proximity (and similarity) relations, qualifications via clause annotations and CLP($\mathcal{R}$) style programming. Moreover, our results in Section 4 on the semantic correctness of our implementation technique are in our opinion another contribution of this paper which has no counterpart in related approaches.

In the future, and taking advantage of the prototype system we have already developed, we plan to investigate possible applications which can profit from proximity relations and qualifications, such as in the area of flexible query answering. In particular, we plan to investigate application related to flexible answering of queries to XML documents, in the line of (Campi et al. 2009) and other related papers. As support for practical applications, we also plan to increase the repertoire of constraint and qualification domains which can be used in the `(S)QCLP` prototype, adding the constraint domain $\mathcal{FD}$ and the qualification domain $\mathcal{W}_d$ defined in Section 2.2.3 of (Rodríguez-Artalejo and Romero-Díaz 2010b). On a more theoretical line, other possible lines of future work include: a) extension of the SLD($\mathcal{D}$) resolution procedure presented in (Rodríguez-Artalejo and Romero-Díaz 2008) to a SQCLP goal solving procedure able to work with constraints and a proximity relation; b) investigation of the conjecture stated at the end of Section 2; and c) extension of the QCFLP (*qualified constraint functional logic programming*) scheme in (Caballero et al. 2009) to work with a proximity relation and higher-order functions, as well as the implementation of the resulting scheme in the CFLP($\mathcal{C}$)-system Toy (Arenas et al. 2007).

References

Apt, K. R. 1990. Logic programming. In *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed. Vol. B: Formal Models and Semantics. Elsevier and The MIT Press, 493–574.

Arcelli, F. and Formato, F. 1999. Likelog: a logic programming language for flexible data retrieval. In *Proceedings of the 1999 ACM Symposium on Applied computing (SAC'99)*. ACM Press, New York, NY, USA, 260–267.

Arcelli Fontana, F. 2002. Likelog for flexible query answering. *Soft Computing 7*, 107–114.

Arenas, P., Fernández, A. J., Gil, A., López-Fraguas, F. J., Rodríguez-Artalejo, M., and Sáenz-Pérez, F. 2007. $\mathcal{TOY}$, a multiparadigm declarative language. version 2.3.1. R. Caballero and J. Sánchez (Eds.), Available at `http://toy.sourceforge.net`.

Baader, F. and Nipkow, T. 1998. *Term Rewriting and All That.* Cambridge University Press.

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

Bistarelli, S., Montanari, U., and Rossi, F. 2001. Semiring-based constraint logic programming: Syntax and semantics. *ACM Transactions on Programming Languages and Systems 3,* 1 (January), 1–29.

Caballero, R., Rodríguez-Artalejo, M., and Romero-Díaz, C. A. 2008. Similarity-based reasoning in qualified logic programming. In *PPDP '08: Proceedings of the 10th international ACM SIGPLAN conference on Principles and Practice of Declarative Programming.* ACM, Valencia, Spain, 185–194.

Caballero, R., Rodríguez-Artalejo, M., and Romero-Díaz, C. A. 2009. Qualified computations in functional logic programming. In *Logic Programming (ICLP'09)*, P. Hill and D. Warren, Eds. LNCS, vol. 5649. Springer-Verlag Berlin Heidelberg, Pasadena, CA, USA, 449–463.

Campi, A., Damiani, E., Guinea, S., Marrara, S., Pasi, G., and Spoletini, P. 2009. A fuzzy extension of the XPath query language. *Journal of Intelligent Information Systems 33,* 3 (December), 285–305.

Dubois, D. and Prade, H. 1980. *Fuzzy Sets and Systems: Theory and Applications.* Academic Press, New York, NY, USA.

Freuder, E. C. and Wallace, R. J. 1992. Partial constraint satisfaction. *Artificial Intelligence 58,* 1–3, 21–70.

Georget, Y. and Codognet, P. 1998. Compiling semiring-based constraints with CLP(FD,S). In *Proceedings of the 4th International Conference on Principles and Practice of Constraint Programming.* LNCS, vol. 1520. Springer-Verlag, 205–219.

Guadarrama, S., Muñoz, S., and Vaucheret, C. 2004. Fuzzy prolog: A new approach using soft constraint propagation. *Fuzzy Sets and Systems 144,* 1, 127–150.

Hájek, P. 1998. *Metamathematics of Fuzzy Logic.* Dordrecht: Kluwer.

Höhfeld, M. and Smolka, G. 1988. Definite relations over constraint languages. Tech. Rep. LILOG Report 53, IBM Deutschland.

Ishizuka, M. and Kanai, N. 1985. Prolog-ELF incorporating fuzzy logic. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence (IJCAI'85)*, A. K. Joshi, Ed. Morgan Kaufmann, Los Angeles, CA, USA, 701–703.

Jaffar, J. and Lassez, J. L. 1987. Constraint logic programming. In *Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of Programming Languages (POPL'87).* ACM New York, NY, USA, Munich, West Germany, 111–119.

Jaffar, J., Maher, M., Marriott, K., and Stuckey, P. J. 1998. Semantics of constraints logic programs. *Journal of Logic Programming 37,* 1-3, 1–46.

Julián, R., Moreno, G., and Penabad, J. 2009. An improved reductant calculus using fuzzy partial evaluation techniques. *Fuzzy Sets and Systems 160,* 2, 162–181.

Julián-Iranzo, P., Rubio, C., and Gallardo, J. 2009. Bousi∼Prolog: a prolog extension language for flexible query answering. In *Proceedings of the Eighth Spanish Conference on Programming and Computer Languages (PROLE 2008)*, J. M. Almendros-Jiménez, Ed. ENTCS, vol. 248. Elsevier, Gijón, Spain, 131–147.

Julián-Iranzo, P. and Rubio-Manzano, C. 2009a. A declarative semantics for Bousi∼Prolog. In *PPDP'09: Proceedings of the 11th ACM SIGPLAN conference on Principles and practice of declarative programming.* ACM, Coimbra, Portugal, 149–160.

Julián-Iranzo, P. and Rubio-Manzano, C. 2009b. A similarity-based WAM for Bousi∼Prolog. In *Bio-Inspired Systems: Computational and Ambient Intelligence (IWANN 2009).* LNCS, vol. 5517. Springer Berlin / Heidelberg, Salamanca, Spain, 245–252.

Kifer, M. and Subrahmanian, V. S. 1992. Theory of generalized annotated logic programs and their applications. *Journal of Logic Programming 12,* 3&4, 335–367.

Lee, R. C. T. 1972. Fuzzy logic and the resolution principle. *Journal of the Association for Computing Machinery (ACM) 19,* 1 (January), 109–119.

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

Lloyd, J. W. 1987. *Foundations of Logic Programming, Second Edition.* Springer.

Loia, V., Senatore, S., and Sessa, M. I. 2004. Similarity-based SLD resolution and its role for web knowledge discovery. *Fuzzy Sets and Systems 144,* 1, 151–171.

Medina, J., Ojeda-Aciego, M., and Vojtáš, P. 2001a. Multi-adjoint logic programming with continuous semantics. In *Logic Programming and Non-Monotonic Reasoning (LPNMR'01)*, T. Eiter, W. Faber, and M. Truszczyinski, Eds. LNAI, vol. 2173. Springer-Verlag, 351–364.

Medina, J., Ojeda-Aciego, M., and Vojtáš, P. 2001b. A procedural semantics for multi-adjoint logic programming. In *Progress in Artificial Intelligence (EPIA'01)*, P. Brazdil and A. Jorge, Eds. LNAI, vol. 2258. Springer-Verlag, 290–297.

Medina, J., Ojeda-Aciego, M., and Vojtáš, P. 2004. Similarity-based unification: a multi-adjoint approach. *Fuzzy Sets and Systems 146*, 43–62.

Riezler, S. 1998. Probabilistic constraint logic programming. Ph.D. thesis, Neuphilologischen Fakultät del Universität Tübingen.

Rodríguez-Artalejo, M. and Romero-Díaz, C. A. 2008. Quantitative logic programming revisited. In *Functional and Logic Programming (FLOPS'08)*, J. Garrigue and M. Hermenegildo, Eds. LNCS, vol. 4989. Springer-Verlag, Ise, Japan, 272–288.

Rodríguez-Artalejo, M. and Romero-Díaz, C. A. 2010a. A declarative semantics for CLP with qualification and proximity. *Theory and Practice of Logic Programming, 26th Int'l. Conference on Logic Programming (ICLP'10) Special Issue 10,* 4–6, 627–642.

Rodríguez-Artalejo, M. and Romero-Díaz, C. A. 2010b. Fixpoint & Proof-theoretic Semantics for CLP with Qualification and Proximity. Tech. Rep. SIC-1-10, Universidad Complutense, Departamento de Sistemas Informáticos y Computación, Madrid, Spain.

Sessa, M. I. 2001. Translations and similarity-based logic programming. *Soft Computing 5,* 2.

Sessa, M. I. 2002. Approximate reasoning by similarity-based SLD resolution. *Theoretical Computer Science 275,* 1-2, 389–426.

van Emden, M. H. 1986. Quantitative deduction and its fixpoint theory. *Journal of Logic Programming 3,* 1, 37–53.

Vojtáš, P. 2001. Fuzzy logic programming. *Fuzzy Sets and Systems 124*, 361–370.

Zadeh, L. A. 1965. Fuzzy sets. *Information and Control 8,* 3, 338–353.

Zadeh, L. A. 1971. Similarity relations and fuzzy orderings. *Information Sciences 3,* 2, 177–200.