UNIVERSIDAD COMPLUTENSE DE MADRID

FACULTAD DE CIENCIAS MATEMÁTICAS
Departamento de Sistemas Informáticos y Computación

UN ESQUEMA DE PROGRAMACIÓN LÓGICO-FUNCIONAL CON RESTRICCIONES: MARCO TEÓRICO Y APLICACIÓN A LA DEPURACIÓN DECLARATIVA

MEMORIA PARA OPTAR AL GRADO DE DOCTOR

PRESENTADA POR

Rafael del Vado Vírseda

Bajo la dirección del doctor
Marío Rodríguez Artalejo

Madrid, 2009

- **ISBN: 978-84-692-2770-1**

Un Esquema de Programación Lógico-Funcional con Restricciones:

Marco Teórico y Aplicación a la Depuración Declarativa

TESIS DOCTORAL

Rafael del Vado Vírseda

Departamento de Sistemas Informáticos y Computación
Facultad de Ciencias Matemáticas
Universidad Complutense de Madrid

Octubre 2008

Un Esquema de Programación Lógico-Funcional con Restricciones: Marco Teórico y Aplicación a la Depuración Declarativa

Memoria presentada para obtener el grado de
Doctor en Ciencias Matemáticas por

Rafael del Vado Vírseda

Dirigida por el profesor

Mario Rodríguez Artalejo

Departamento de Sistemas Informáticos y Computación
Facultad de Ciencias Matemáticas
Universidad Complutense de Madrid

Octubre 2008

Resumen

En la primera parte de este trabajo proponemos un nuevo marco teórico que permite caracterizar la semántica declarativa y operacional de los lenguajes de programación lógico funcionales perezosos con restricciones, a través del desarrollo de un esquema genérico $CFLP(\mathcal{D})$ sobre un dominio $\mathcal{D}$ paramétricamente dado. Sobre la base de una nueva formalización matemática de este dominio y de su resolutor asociado, el nuevo esquema permite definir instancias particulares de interés práctico, como el dominio de Herbrand $\mathcal{H}$, el dominio $\mathcal{R}$ de los números reales, o el dominio $\mathcal{FD}$ de los números enteros con restricciones de dominio finito, para los que el esquema proporciona además lenguajes concretos de programación mediante programas constituidos por reglas de reescritura con restricciones que definen nuevas funciones.

El esquema $CFLP(\mathcal{D})$ permite caracterizar la semántica declarativa de este tipo de programas a partir de una noción adecuada de *c-interpretación* sobre un dominio, gracias a la cual se definen dos clases de semánticas de modelos, denominadas, respectivamente, *semántica débil* y *semántica fuerte*. Demostramos la existencia de un modelo mínimo para cada una de estas dos semánticas, caracterizándolo como mínimo punto fijo. El desarrollo de una *lógica de reescritura con restricciones* denominada $CRWL(\mathcal{D})$, también definida de forma paramétrica sobre un dominio de restricciones $\mathcal{D}$ arbitrario, permite proporcionar un nuevo marco lógico para la programación en el esquema $CFLP(\mathcal{D})$, y en consecuencia, una forma alternativa muy útil de caracterizar la semántica declarativa de los programas.

Con el fin de proporcionar también una base formal a la semántica operacional del esquema $CFLP(\mathcal{D})$, proponemos dos métodos de resolución de objetivos para programas que hacen uso del *estrechamiento* como mecanismo de cómputo. En primer lugar, se presenta un cálculo de transformación de objetivos denominado $CLNC(\mathcal{D})$, basado en *estrechamiento perezoso con restricciones*, para el que demostramos su corrección y completitud fuerte con respecto a la semántica declarativa de $CRWL(\mathcal{D})$. En segundo lugar, introducimos un refinamiento eficiente mediante el cálculo $CDNC(\mathcal{D})$ que permite integrar *árboles definicionales* con el propósito de asegurar que todos los pasos que se ejecutan en los cómputos son realmente necesarios. Como aplicación práctica, describimos el sistema $\mathcal{TOY}(\mathcal{FD})$, una implementación de la instancia particular $CFLP(\mathcal{FD})$ en el sistema lógico funcional con restricciones $\mathcal{TOY}$.

En la segunda parte de esta tesis desarrollamos, como aplicación del esquema genérico $CFLP(\mathcal{D})$, un método de diagnosis de programas con restricciones que permite integrar de una forma natural aproximaciones previas que en el campo de la *depuración declarativa* han sido desarrolladas por separado para los paradigmas lógico funcional y lógico con restricciones, analizando tanto su validez teórica como sus posibles implementaciones en sistemas ya existentes. La técnica de diagnosis que proponemos permite utilizar *árboles de cómputo* en lugar de trazas, de forma que estos puedan ser construidos a posteriori para poder representar la estructura de un cómputo que ha sido calificado como un síntoma de error por parte del usuario.

En este trabajo consideramos dos tipos de errores susceptibles de ser tratados mediante nuestro método de depuración declarativa. En primer lugar, se consideran aquellas respuestas que han sido obtenidas de manera inesperada para un objetivo determinado (*respuestas incorrectas*). Para este primer caso, proponemos un *cálculo de prueba positivo* desarrollado a partir de $CRWL(\mathcal{D})$, mediante el que es posible definir los árboles de cómputo como árboles de derivación lógica. En segundo lugar, se considera como un posible error aquel en el que en el conjunto de todas las respuestas obtenidas para un mismo objetivo falte alguna respuesta esperada (*respuestas perdidas*). En este segundo caso, proponemos otro cálculo denominado *cálculo de prueba negativo*, en el cual las derivaciones lógicas formalizan la recolección de respuestas computadas y sirven para definir árboles de cómputo aplicables a la diagnosis de respuestas perdidas. Demostramos la corrección lógica de los métodos de diagnosis propuestos, tanto para respuestas incorrectas como para respuestas perdidas, en relación a los sistemas de resolución de objetivos presentados. Finalmente, proponemos la implementación en el sistema $\mathcal{TOY}$ de dos posibles herramientas basadas en los métodos de depuración descritos.

Agradecimientos

"*El ser humano es el único animal que tropieza dos veces con la misma piedra*". Yo aún diría más, no solo dos, sino tres, cuatro, ... y las veces que hagan falta si uno se empeña en alcanzar un objetivo como es el de escribir una tesis doctoral. Cuando concluí mi trabajo de tercer ciclo, en un tema como el de las "estrategias de estrechamiento perezoso", tomé la decisión de no volver nunca más a realizar un trabajo sobre aspectos tan teóricos como los que allí se abordaban. Pues bien, a pesar de ello, me encuentro ahora escribiendo los agradecimientos de un trabajo que supera con creces en volumen teórico al anterior. Esto no quiere decir que me arrepienta de ello, en absoluto, pero sí he de reconocer que la tarea no ha sido fácil desde que comencé hace cuatro años a trabajar en el tema que se propone en esta tesis, y que en muchas ocasiones me ha asaltado la idea de "tirar la toalla". Por esta razón, mis agradecimientos van destinados a todos aquellos que me han acompañado en este duro camino y han impedido que tal cosa llegara a suceder. De entre todos ellos, mi familia, mis padres y mi hermana, son los que han llevado a cabo la parte más importante de esta labor, y por ello es a los primeros a los que quiero expresar mis agradecimientos. Además de a mi familia, este apartado va dedicado a las siguientes personas.

A Mario Rodríguez Artalejo, en quien he tenido la gran fortuna de encontrar un cuidadoso y concienzudo director de tesis. Su apoyo y disponibilidad incondicional han hecho de todo este proceso una experiencia de auténtico enriquecimiento personal. Su rigor metodológico, la calidad de su trabajo, su paciencia, su dedicación y amor por la investigación, han contribuido de manera decisiva a formarme como persona y como investigador.

A Teresa Hortalá, a quien en ocasiones he calificado como de mi "segunda madre", por todo el cariño que desde siempre (y ya son bastantes años) me ha dado, por toda la ayuda en cualquier cosa que he necesitado, y por tantas y tantas cosas que necesitaría el resto de esta tesis para poder darle las gracias. A ella le debo mucho de lo que soy y de lo que hay escrito en esta tesis desde que asistí por primera vez a sus clases de doctorado.

A Paco López Fraguas, como a Mario, le debo buena parte de mi pasión por el estudio de los Fundamentos Históricos de la Informática y de sus raíces matemáticas, por la investigación en temas formales, y en especial, por la docencia. Aún recuerdo muchas de sus clases de "Teoría de la Computación" como si acabara de asistir a

ellas, y han constituido para mí todo un referente de inspiración y de motivación a la hora de dar mis clases y de formarme como profesor. Como amigo, como compañero, y como responsable de los proyectos de los que he formado parte en todos estos años siempre he contado con su apoyo, y espero seguir haciéndolo en años sucesivos.

Agradezco también a todo el Grupo de Programación Declarativa su colaboración y apoyo. En particular a Sonia, por su simpatía, por su paciencia, por su buen humor, por todo lo que nos hemos reído (y lo que nos queda) juntos, y en definitiva, por ser como es. Por supuesto, no me olvido de Rafa Caballero, sin el cual nunca podría haberme adentrado en los secretos de la depuración declarativa en $\mathcal{TOY}$, siempre con una sonrisa y con algún comentario ingenioso con el que hacerme olvidar los malos momentos.

Igualmente doy las gracias al resto de mis compañeros del Departamento de Sistemas Informáticos y Computación por haber creado el ambiente de trabajo en el que se ha desarrollado esta tesis. Mi compañera de despacho Clara ha sido fundamental en este aspecto, por toda la ayuda que me ha prestado a la hora de poder compaginar la preparación de mis clases con la investigación. A Ricardo, Narciso, Alberto Verdejo e Isabel Pita, con quienes he participado en la experiencia educativa de llevar con éxito el sistema *Maude* a las aulas. A mis vecinos de despacho, Manuel Núñez y Mercedes Merayo, a quienes su amistad, sus consejos (y alguna que otra copa) me han ayudado especialmente en el último tramo de preparación de esta memoria. Y por supuesto, a mis alumnos de la FDI, los que fueron, son y serán, por lo mucho que aprendo cada día con ellos, por su ingenuidad, por su interés, por su sinceridad y por su paciencia conmigo, por sus ganas de aprender y por la alegría con la que decoran cada día mis clases.

Por último, quisiera agradecer a George Gershwin, Cole Porter, Irving Berlin, Leonard Bernstein, Rodgers & Hammerstein, Kander & Ebb, Cy Coleman, Stephen Sondheim y Andrew Lloyd Webber por haber compuesto música para cada una de las páginas de esta tesis, y a Jerome Robbins, Michael Bennett y Bob Fosse por haberlas coreografiado.

So if you care to find me
Look to the western sky
As someone told me lately
Everyone deserves the chance to fly
And if I'm flying solo
At least I'm flying free
To those who'd ground me
Take a message back from me
Tell them how I am defying gravity
I'm flying high, defying gravity
And soon I'll match them in renown
And nobody in all of Oz
No wizard that there is or was
Is ever gonna bring me down!

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Muchos problemas de interés práctico que surgen en el campo de las Matemáticas, la Informática y de cualquier Ciencia en general pueden resolverse mediante su reducción a un sistema de ecuaciones sobre un dominio concreto de valores y a su posterior resolución mediante la aplicación de un mecanismo de resolución de restricciones adecuado a ese dominio. El diseño e implementación de esquemas teóricos que permitan expresar de una forma sencilla tales problemas, además de incorporar técnicas eficientes para su resolución, ha sido uno de los aspectos teóricos que mayor interés e investigación ha suscitado en las últimas décadas. Entre las distintas aproximaciones que se han propuesto, las que han alcanzado un mayor éxito son aquellas que permiten integrar alguna forma de programación declarativa (por ejemplo, programación lógica, programación funcional o programación lógico funcional) con la resolución de restricciones sobre un dominio dado. El motivo de esta elección radica en que la componente de programación declarativa elegida proporciona unos fundamentos teóricos muy sólidos sobre los que estudiar el modelado de soluciones (a alto nivel) de problemas que involucran restricciones y que admiten la definición de formalismos propios (en la forma de predicados y/o funciones) por parte del usuario en el dominio de restricciones sobre el que se está trabajando.

De entre los paradigmas de programación declarativa que resultan más adecuados para la integración de restricciones, destacan la programación lógica y la programación lógico funcional, debido particularmente a que las restricciones poseen una naturaleza relacional. Precisamente por esta razón, los lenguajes funcionales (por carecer de un componente lógico) no se amoldan especialmente bien al paradigma de la programación con restricciones. El impulso definitivo que ha convertido a la programación lógica con restricciones en uno de los campos de investigación más activos ha sido, sin duda, la incorporación del esquema $CLP(\mathcal{D})$ [JL87] como marco general para la formalización de la semántica operacional y declarativa de una clase extendida de programas lógicos parametrizado por un dominio de restricciones $\mathcal{D}$. Aunque estas ideas han sido extendidas, con mayor o menor éxito, al paradigma lógico funcional con restricciones [DGP92a, DGP92b, Lop92, Lop94, Mar00, MIS00], que

nace con el deseo de incorporar características funcionales, la realidad es que aun no existe un marco teórico tan consolidado y universalmente aceptado como el esquema $CLP(\mathcal{D})$, que sirva además de base para el desarrollo de técnicas y herramientas que ayuden al programador a manipular y optimizar sus programas.

En este contexto, la tesis doctoral que proponemos tiene como objetivo el estudio de un marco teórico que permita caracterizar la semántica declarativa y operacional de los lenguajes de programación lógico funcionales perezosos con restricciones, a través del desarrollo de un nuevo esquema genérico $CFLP(\mathcal{D})$ sobre un dominio $\mathcal{D}$ paramétricamente dado. Debido a su componente funcional, el nuevo esquema $CFLP(\mathcal{D})$ añade una mayor expresividad a $CLP(\mathcal{D})$ al permitir la declaración de funciones y la evaluación perezosa en el estilo más clásico de la programación funcional, además de incluir otros aspectos ventajosos que no están presentes (o que son inusuales) en el paradigma $CLP(\mathcal{D})$ o en otros esquemas $CFLP$, como una semántica declarativa asociada a una lógica de reescritura con restricciones, la formalización del comportamiento de los resolutores para controlar la interacción entre la demanda de ejecución de llamadas a funciones y el cómputo de formas resueltas, y cálculos de resolución de objetivos correctos y fuertemente completos con respecto a la semántica declarativa, basados en la colaboración de un resolutor con métodos de estrechamiento demandado.

Además, y como una aplicación interesante de las ventajas que ofrece el nuevo esquema $CFLP(\mathcal{D})$, desarrollamos en esta tesis un método de diagnosis de programas con restricciones que permite integrar de una forma natural aproximaciones previas que en el campo de la depuración declarativa han sido desarrolladas por separado para los paradigmas lógico funcional y lógico con restricciones, analizando tanto su validez teórica como sus posibles implementaciones en sistemas ya existentes. La elaboración de este método lógicamente correcto de depuración de respuestas perdidas en $CFLP(\mathcal{D})$ es la principal contribución de la segunda parte de la tesis.

Comenzamos este primer capítulo realizando una breve panorámica del estado actual de la investigación en el campo de la programación declarativa multiparadigma que nos ocupa y que nos sirve de contexto en esta tesis. Pondremos especial énfasis en mostrar cuáles son los antecedentes en el desarrollo de esquemas previos de programación lógico funcional con restricciones, así como los antecedentes en la investigación de técnicas de depuración declarativa. Estos dos puntos se recogen en las dos primeras secciones de este capítulo, en las que también comentaremos las ventajas e inconvenientes de las distintas propuestas existentes que han servido de motivación en la elaboración de esta tesis, así como las principales novedades que aporta el presente trabajo con respecto a otros trabajos relacionados. Finalmente, en la tercera y última sección describiremos con mayor detalle los objetivos que se han propuesto en este trabajo, el desglose en capítulos y apéndices en los que se estructura la tesis y las principales publicaciones en las que se basa el material expuesto en esta memoria.

1.1. Programación declarativa con restricciones

La *programación con restricciones* (*CP*, del inglés *Constraint Programming*) [MS98, Apt03] constituye uno de los paradigmas que ha suscitado un mayor interés en las últimas décadas en la comunidad investigadora. Esto se debe a que se sustenta sobre unos fundamentos teóricos muy sólidos [Tsa93], además de ser un campo de investigación muy heterogéneo que abarca desde tópicos puramente teóricos hasta aplicaciones totalmente prácticas en la industria. Como consecuencia de ello, el paradigma *CP* está atrayendo un amplio interés comercial pues es muy apropiado para el modelado de una extensa gama de problemas de optimización y, en particular, de todos aquellos problemas que involucran restricciones heterogéneas y búsqueda combinatoria.

Básicamente, una *restricción* es una relación que se establece entre las entidades (e.g., variables u objetos) de un problema. Las restricciones se usan para modelar problemas reales mediante un enfoque idealizado de la interacción entre las diferentes componentes de un problema. La forma en la cual esta interacción se define depende de la capacidad y la experiencia del programador. De manera más concreta, un *Problema de Satisfacción de Restricciones* (*CSP*, del inglés *Constraint Satisfaction Problem*) es un conjunto de restricciones definidas sobre un número finito de variables que están restringidas a tomar valores en un conjunto de dominios de cómputo. Resolver un *CSP* significa encontrar un conjunto de valores posibles en los dominios de cómputo de las variables restringidas que garanticen la satisfacción de todas las restricciones que modelan el problema. Por ejemplo, podemos considerar un *CSP* definido mediante una única restricción $X < Y$, donde las variables X e Y están inicialmente restringidas a tomar valores en el dominio discreto $\{0, 1, 2\}$. Este *CSP* tiene entonces tres soluciones: $X = 0$, $Y = 1$; $X = 0$, $Y = 2$; y $X = 1$, $Y = 2$. Actualmente, la resolución de *CSP*s puede realizarse mediante el uso de diferentes técnicas, desde técnicas tradicionales hasta otras mucho más modernas [Rut98, Bar03].

Las restricciones han sido integradas en paradigmas de programación muy diferentes, aunque los declarativos han resultado ser más adecuados para ello. El objetivo fundamental de los lenguajes de *programación declarativa* en sentido amplio es el de proporcionar un alto nivel de abstracción, de forma que la especificación de un problema sea un programa capaz de resolver el problema. Se intenta así liberar al programador de tener que describir detalladamente la secuencia de acciones que debe realizar la máquina para obtener el resultado buscado, como es habitual en los tradicionales lenguajes imperativos. Se abstraen los detalles concretos del hardware y, en general, los programas son más breves y más sencillos de mantener que los programas imperativos. Por otra parte, los lenguajes declarativos están basados en formalismos matemáticos que permiten hacer un estudio riguroso y preciso de los aspectos semánticos subyacentes. Con más precisión, la característica fundamental

de la programación declarativa es el uso de la lógica como lenguaje de programación, lo que puede conceptualizarse como sigue: un programa es una teoría formal en una lógica adecuada y la computación se entiende como una forma de inferencia o deducción en dicha lógica. Los principales requisitos que debe entonces cumplir la lógica empleada son:

- Disponer de un *lenguaje* que sea suficientemente expresivo para cubrir un campo de aplicación interesante.
- Disponer de una *semántica operacional*, es decir, de un mecanismo de cómputo que permita ejecutar los programas.
- Disponer de una *semántica declarativa* que permita dar un significado a los programas de forma independiente a su posible ejecución.
- Resultados de *corrección* y *completitud* que aseguren que lo que se computa coincide con aquello que es considerado como verdadero (de acuerdo con la noción de verdad que sirve de base a la semántica declarativa).

Existen varios paradigmas que representan a la programación declarativa. Los dos más importantes, la *Programación Lógica* [Llo87a, Apt90] y la *Programación Funcional* [Bar90] han evolucionado de forma independiente, pero manteniendo como prioridad común la expresividad. Como resultado se han forjado dos estilos de programación distintos que intentan aprovechar las ventajas que ofrece uno u otro enfoque. El potencial, en el caso de la programación funcional, viene dado fundamentalmente por la evaluación perezosa, el orden superior y los tipos, mientras que en programación lógica las variables lógicas, los modos múltiples de uso de los predicados y el indeterminismo suponen la principal aportación. Como amalgama de estas dos vertientes ha surgido más recientemente otro paradigma dentro de la programación declarativa, el denominado paradigma de *programación lógico funcional* (veáse [Han94b] y [Han07a] para una panorámica), en el que se pretende reunir las principales ventajas de ambos estilos de programación en uno nuevo. Esta ha sido la motivación de lenguajes como $\mathcal{TOY}$ [LS99b, ALR07] y *Curry* [Han06], que utilizan una combinación de la reescritura y la unificación denominada *estrechamiento* (en inglés, *narrowing*) [Sla74, Lan75, Fay79, Hul80] como mecanismo operacional.

Como ya se ha comentado anteriormente, las restricciones poseen una naturaleza relacional que hace que los lenguajes con componentes lógicos parezcan más apropiados para su integración. En particular, en el paradigma declarativo, los lenguajes lógicos y los lógico funcionales parecen ser los más apropiados, mientras que los lenguajes funcionales (precisamente por carecer de un componente lógico) no se amoldan especialmente bien al paradigma de la programación con restricciones. La programación lógico funcional con restricciones se sitúa dentro de la *programación multiparadigma con restricciones*, basada en la idea de combinar varios paradigmas

en un mismo marco de programación. Algunas propuestas interesantes incluyen aspectos de concurrencia y orientación a objetos, que caen fuera del ámbito de esta tesis. Entre ellas, podemos mencionar:

- La *programación concurrente con restricciones* [Sar93, SR90, SRP91], en la cual una restricción definida por el usuario es considerada como un *proceso*, y un estado está directamente relacionado con una red de procesos enlazados a través de variables compartidas mediante un almacén de restricciones.
- El enfoque ofrecido por el lenguaje *Oz* [vRBDH+03] (el cual posteriormente evolucionó al lenguaje *Mozart* [vRH04]) y que combina características de los lenguajes lógicos con restricciones, los lenguajes funcionales y los lenguajes concurrentes.
- El enfoque ofrecido por el lenguaje *LIFE* [AP93], que es un lenguaje experimental que propone integrar la programación lógica, la programación funcional y la programación orientada a objetos, en la línea de otros lenguajes similares precursores suyos como *LOGIN* [AN86] y *Le Fun* [ALN87].

En el resto de esta sección vamos a referirnos a los paradigmas de programación declarativa más próximos a la línea de esta tesis.

1.1.1. Programación lógica con restricciones

En un lenguaje de programación lógica, los programas se entienden como teorías en una cierta lógica, de forma que los conceptos de computación en la máquina, deducción en la lógica, y satisfacción en un modelo estándar de la teoría resultan equivalentes. Además, un requerimiento adicional de eficiencia y la existencia de un mecanismo efectivo de extracción de respuestas son necesarios para establecer una distinción entre demostración automática de teoremas y programación lógica.

La programación lógica [CR96, Apt90, Kow74, Llo87a] se basa en fragmentos de la lógica de predicados, siendo el más popular la lógica de *cláusulas de Horn* (*HCL*, del inglés *Horn Clause Logic*), que puede emplearse como base para un lenguaje de programación al poseer una semántica operacional susceptible de una implementación eficiente, como es el caso de la *resolución SLD*. Como semántica declarativa se utiliza una semántica basada en la *teoría de modelos* que toma como dominio de interpretación un universo puramente sintáctico: el *universo de Herbrand*. La resolución SLD es un método de prueba por refutación que emplea el algoritmo de *unificación* como mecanismo de base y permite la extracción de respuestas, es decir, la vinculación de un valor a una *variable lógica*. Es un método de prueba correcto y completo para la lógica *HCL*. Destacamos a continuación algunas de las características más interesantes de la programación lógica que han influido en el planteamiento de los objetivos y contenidos de esta tesis:

- Manejo de variables libres en expresiones y cómputo de soluciones para las variables libres mediante *unificación*. El programa puede así responder a diferentes cuestiones (*objetivos*) sin necesidad de efectuar ningún cambio en el programa.

- *Búsqueda no determinista* de soluciones asimilada al mecanismo de cómputo (denominada *built-in search*), lo que permite computar con datos parcialmente definidos y hace posible que la relación de entrada/salida no esté fijada de antemano.

- *Gestión automática de la memoria del sistema*, lo que permite evitar una de las mayores fuentes de errores que habitualmente se cometen en la programación con otros lenguajes (como los que se derivan del manejo de punteros en el lenguaje *C*).

La *programación lógica con restricciones* (*CLP*, del inglés *Constraint Logic Programming*) [vHen89, vHen91, JM94, JMMS98] ha resultado ser uno de los campos de investigación más activos, que recibió un gran impulso gracias a la introducción, por parte de Jaffar y Lassez [JL87], del esquema $CLP(\mathcal{D})$ como marco general para formalizar las semánticas operacionales y declarativas de una clase extendida de programas lógicos parametrizada por un dominio de cómputo $\mathcal{D}$.

Los lenguajes de programación *CLP* nacieron como una combinación de dos paradigmas: la programación con restricciones y la *programación lógica* (*LP*, del inglés *Logic Programming* [Llo87a, Apt90]), basada en una idea puramente declarativa donde los programas se construyen como teorías utilizando fragmentos de la lógica de predicados. El éxito de *CLP* radica en que permite combinar la declaratividad de *LP* con la eficiencia de *CP* [Bar03]. A pesar de que los programas *CLP* dependen de un dominio de aplicación (i.e., el dominio de cómputo), el esquema $CLP(\mathcal{D})$ fue ideado para la creación de lenguajes lógicos que compartieran el mismo mecanismo de evaluación. En concreto, la idea básica del esquema $CLP(\mathcal{D})$ es la de reemplazar la unificación clásica de *LP* por un mecanismo de resolución de restricciones sobre un dominio de cómputo específico $\mathcal{D}$, denominado *resolutor de restricciones*. Las diferentes instancias de $\mathcal{D}$ (e.g., números reales, enteros, conjuntos, booleanos, etc.) generan las diferentes instancias del esquema $CLP(\mathcal{D})$, el cual permite además que *CLP* pueda resolver problemas que *LP* no podía resolver de manera natural, por lo que es especialmente interesante para el paradigma declarativo. En cada una de las instancias del esquema, el lenguaje de programación lógico subyacente es extendido con un conjunto de operaciones y estructuras que pueden ser usadas sobre el dominio de cómputo y que pueden ser directamente utilizadas por el usuario. En este sentido, un lenguaje *CLP* puede considerarse como un lenguaje lógico donde se han incorporado restricciones y métodos de resolución de restricciones. Por lo tanto, la diferencia fundamental entre *LP* y *CLP* estriba en la interpretación operacional

de las restricciones y no en su interpretación declarativa. En consecuencia, la idea fundamental del esquema *CLP* es que, tanto el lenguaje lógico subyacente como sus semánticas operacionales y declarativas pueden ser parametrizadas por un dominio de cómputo $\mathcal{D}$ y las operaciones sobre este dominio.

1.1.2. Programación funcional

Los lenguajes de programación funcional están enraizados en el concepto de *función* (matemática) y su definición mediante *ecuaciones* (generalmente recursivas), que constituyen el programa. Diversos autores [PvE93, Rea93] consideran que un programa funcional incluye también, además de una lista de ecuaciones de definición de funciones, una expresión básica (sin variables) que se pretende evaluar como *objetivo*. La ejecución de un programa consiste entonces en la evaluación de la expresión inicial de acuerdo con las ecuaciones de definición de las funciones y alguna forma de reducción. La *reducción* es un proceso por el cual, mediante una secuencia de pasos, transformamos la expresión inicial, compuesta de símbolos de función y de símbolos de constructoras de datos, en un valor (de su tipo), es decir, una expresión que sólo contiene apariciones de símbolos de constructoras. El valor obtenido se considera el resultado de la evaluación. La secuencia de pasos dada en el proceso de reducción depende de la estrategia de reducción empleada. Podemos distinguir dos *estrategias de reducción* o *modelos de evaluación*, la denominada impaciente y la perezosa. Una estrategia *impaciente* evalúa primero los argumentos de una función mientras que una estrategia *perezosa* evalúa los argumentos sólo si su valor es necesario para el cómputo de dicha función, intentando evitar cálculos innecesarios. Si bien la estrategia impaciente es más fácil de implementar en los computadores con arquitecturas convencionales, puede conducir a secuencias de reducción que no terminan en situaciones en las que una evaluación perezosa es capaz de computar un valor. Este hecho, junto con algunas facilidades de la programación (por ejemplo, la evaluación perezosa libera al programador de la preocupación por el orden de evaluación en las expresiones [Hud89]) y ventajas expresivas que proporcionan, por ejemplo, la habilidad de computar con *estructuras de datos infinitas*, han hecho que la posibilidad de utilizar una estrategia perezosa sea muy apreciada en los lenguajes funcionales modernos. Los cálculos de resolución de objetivos presentados en la primera parte de esta tesis formalizan estrategias eficientes de cómputo perezoso para un paradigma de programación que extiende la programación funcional pura con indeterminismo y resolución de restricciones.

Con este fin, una de las novedades que se incluyen en esta tesis es la incorporación de estrategias perezosas en el mecanismo de cómputo del nuevo esquema genérico de programación *CFLP*, planteado como una extensión y generalización del paradigma de programación funcional al contexto más expresivo del paradigma lógico funcional con restricciones.

En cuanto a los formalismos que sustentan la clase de los lenguajes funcionales, se distinguen dos enfoques: el enfoque funcional *clásico* ([Bar90, Bir98, FH87, Hud89, Pey87, Rea93]) basado en el λ-*cálculo*, y el enfoque *ecuacional* ([oDon77, oDon85, oDon94]) basado en la *lógica ecuacional*. Aunque los dos enfoques difieren en la sintaxis y semántica ([Sco70, Sto77]) del formalismo empleado (el enfoque ecuacional se suele restringir a primer orden, aunque es posible extenderlo a orden superior [Pre98] usando también recursos del λ-cálculo), en ambos casos la *reescritura* [DJ90, Klo92, BN98, Ter03] juega un papel en la reducción de una expresión dada a forma normal o irreducible, que es lo que persigue un cómputo. La utilización de la reescritura permite además establecer una correspondencia sencilla con el formalismo que sustenta la programación lógica. Esta ventaja es apreciable a la hora de integrar los paradigmas de programación lógica y funcional y explica la atención que prestaremos al estudio de los sistemas de reescritura en el desarrollo de esta tesis.

Aunque la inclusión de restricciones como parte de un lenguaje de programación parece ligada de modo más natural al espíritu de la programación lógica, existen diversas propuestas para la combinación de la programación funcional y la programación con restricciones, con el propósito de obtener *programación funcional con restricciones* (*CFP*, del inglés *Constraint Functional Programming*). Las propuestas más interesantes que han servido de referente en la realización de esta tesis son las siguientes:

- En [DG89] se propone una modificación del esquema de Höfeld-Smolka [HS88] mediante el desarrollo de un esquema $CFP(\mathcal{D})$ al estilo de $CLP(\mathcal{D})$. Sin embargo, la expresividad de este enfoque es muy próxima a la del propio esquema $CLP(\mathcal{D})$, ya que se considera un lenguaje de primer orden con funciones posiblemente indeterministas que devuelven conjuntos de valores.

- Otra propuesta que puede servir como base para *CFP* es el λ-*cálculo con restricciones* (del inglés, *constrained* λ-*calculus*) de [Man93, CMW93, CMW96]. Se trata de una extensión del λ-*cálculo* [Bar84, Bar90] (i.e., el sistema estándar de reescritura que proporciona el mecanismo de evaluación para muchos lenguajes funcionales) que permite incluir un almacén de restricciones globales, las cuales son enviadas a este almacén a través de la aplicación de funciones. Permite así incorporar a la sintaxis del λ-cálculo la posibilidad del uso de restricciones, además de una regla de reducción que, en caso de que una restricción determine funcionalmente los valores de algunas variables, permita sustituir éstas por aquellos. Se imponen, sin embargo, condiciones muy fuertes para garantizar la propiedad de Church-Rosser del sistema, y las funciones definidas han de ser estrictas. Además, el mayor problema actual al que ha de hacer frente este enfoque es que el almacén de restricciones juega un papel demasiado pasivo en la evaluación de los objetivos del programa (especialmente con respecto al proceso de búsqueda, ya que no puede guiarlo).

Como ya se ha comentado, la integración de restricciones en el paradigma funcional no ha provocado tanto éxito como en la programación lógica, por lo que en la literatura existen pocas propuestas a tener verdaderamente en cuenta además de las ya mencionadas con respecto a la integración de restricciones en lenguajes funcionales. Sin embargo, a pesar de no ser un marco adecuado para la satisfacción de restricciones, sí que han surgido propuestas híbridas que tienden a combinar funciones y restricciones dentro de un enfoque multiparadigmático (en vez de considerar simplemente un enfoque funcional) que de alguna manera guarda un componente lógico. Estas propuestas son analizadas en las secciones siguientes.

1.1.3. Programación lógico funcional

La integración de la programación lógica y la programación funcional en un solo paradigma que combina sus principales ventajas denominado *programación lógico funcional* (*FLP*, del inglés *Functional Logic Programming*), es quizás la combinación de paradigmas más estudiada y mejor entendida. De la programación lógica, los lenguajes lógico funcionales obtienen el uso de la unificación, la potencia de las variables lógicas, un mecanismo de búsqueda automático indeterminista, y la posibilidad de trabajar con estructuras de datos parciales. De la programación funcional obtienen, entre otros beneficios, la expresividad de las funciones, el empleo de tipos, el orden superior y un mecanismo de evaluación más eficiente (ejecución optimizada de los cálculos deterministas y evaluación perezosa). Algunas recopilaciones de distintas propuestas son [BL86, DL86, AN89, Mor89, GL90, Han94b, Rod01, Han07a].

El mecanismo operacional de los lenguajes lógico funcionales es el resultado de combinar los que utilizan por una parte los lenguajes lógicos (unificación y resolución) y por otra los lenguajes funcionales (ajuste de patrones o *pattern matching*, y reescritura). En general, esto requiere reemplazar la unificación por *unificación semántica* o *E-unificación* [GS89, Sie89, JK91]. Sin embargo, la E-unificación es con frecuencia un problema demasiado difícil de resolver, y sus métodos generales demasiado ineficientes para ser considerada como base del mecanismo de cómputo de un lenguaje de programación real. Por este motivo, la mayor parte de las propuestas adoptan algún tipo de restricción sobre los programas, de modo que se pueda utilizar algún procedimiento más eficiente de E-unificación.

En este sentido, se han formulado muchas propuestas para la combinación eficiente de los estilos de programación lógica y funcional. La gran variedad de escuelas que se aprecia en este campo refleja una diversidad importante de motivaciones. La forma más sencilla de integración consiste en proporcionar una interfaz para un lenguaje lógico y uno funcional ya existentes [RS82, SY84]. Otro método consiste en traducir los programas lógico funcionales a programas lógicos puros, aplanando las llamadas anidadas a función. Por otra parte, se han desarrollado también propuestas de integración en las que se utiliza como mecanismo operacional una combinación de la técnica de resolución SLD con el denominado principio de *residuación*. Infor-

malmente, la residuación [HKM97] consiste en retrasar aquellas llamadas a función que no estén lo suficientemente instanciadas para efectuar un paso (determinista) de reducción. Sin embargo, el principal inconveniente de utilizar la residuación en la definición de la semántica operacional es que se trata de un método incompleto. Muchas propuestas para la integración con fundamento semántico riguroso optan por usar sistemas de reescritura de términos condicionales como programas, y como mecanismo operacional más estudiado y extendido que soporta unificación y variables lógicas en un contexto funcional, la *reescritura con unificación* o, más comúnmente denominada, *estrechamiento* (del inglés, *narrowing*). El procedimiento de estrechamiento puede verse como una extensión de la reescritura, consistente en sustituir el habitual ajuste de patrones por la unificación durante el proceso de reducción. El uso del estrechamiento como mecanismo operacional para realizar las computaciones supone así una extensión muy potente de los programas lógicos tradicionales y el modelo resultante tiene buenas oportunidades para explotar el paralelismo implícito en el lenguaje. La técnica de estrechamiento, como medio para la obtención de un conjunto de soluciones de un problema de unificación ecuacional, fue descrita en los trabajos pioneros de [Sla74] y [Fay79].

LENGUAJE	PRINCIPIO OPERACIONAL	COMPORTAMIENTO
LOGLISP [RS82]	*Resolución + Reducción*	*Impaciente*
K−LEAF [GLMP91]	*Aplanamiento + SLD-Resolución*	*Perezoso*
SLOG [Fri85]	*Estrechamiento*	*Impaciente*
ALF [Han90]	*Estrechamiento*	*Impaciente*
LPG [BE95]	*Estrechamiento*	*Impaciente*
BABEL [MR92]	*Estrechamiento*	*Perezoso*
$\mathcal{TOY}$ [ALR07]	*Estrechamiento*	*Perezoso*
Curry [Han06]	*Estrechamiento + Residuación*	*Perezoso*

Figura 1.1: Algunos lenguajes lógico funcionales y sus características

En general, el procedimiento de estrechamiento es indeterminista, lo que conduce a que se generen espacios de búsqueda de soluciones muy amplios. Esto ha motivado que se diseñen estrategias para reducir el tamaño del espacio de búsqueda, permitiendo así eliminar algunas derivaciones inútiles. Entre las principales estrategias de estrechamiento podemos citar las siguientes: *estrechamiento innermost* [Fri85], *estrechamiento básico* [Hul80, MH94, Ret87], *estrechamiento selección* [BGM88], *estrechamiento perezoso* [AEH94, Han94a, MO98, MOI96, MR92, Red85], etc. Cada una de estas estrategias sigue manteniendo, bajo determinadas condiciones, la completitud del cálculo. En las referencias [AN89, BL86, DP88, Mor89, Han94b, Han07a] y en la Figura 1.1 se puede encontrar una revisión, análisis y clasificación de éstas y otras propuestas para la integración. La colección [DL86] constituye una referencia histórica estándar en el campo. La panorámica más actualizada se presenta en [Han07a]. Estudios detallados sobre las condiciones que debe cumplir un programa, para que una determinada estrategia sea correcta y completa, pueden encontrarse

en [Han94b] y en [MH94]. Varias publicaciones del autor de esta tesis se han centrado en el estudio de *estrategias de estrechamiento perezoso* [Vad02, Vad03a, Vad03b, Vad05, Vad07], mediante las cuales es posible modelar las ventajosas técnicas de la programación funcional perezosa, e incluso permitir la introducción de funciones posiblemente indeterministas que han servido de base en el desarrollo de la semántica operacional del nuevo esquema *CFLP* propuesto en esta tesis.

1.1.4. Programación lógico funcional con restricciones

La *programación lógico funcional con restricciones* (*CFLP*, del inglés *Constraint Functional Logic Programming*) nació a partir del deseo de integrar restricciones en los lenguajes del paradigma lógico funcional. Este paradigma, a pesar de poseer características funcionales, también posee un componente lógico similar al de los lenguajes de programación lógica. El objetivo pretendido con la integración es similar al buscado en el paradigma lógico, es decir, combinar la expresividad de los lenguajes lógico funcionales con la eficiencia que ofrece el paradigma con restricciones. En definitiva, lo que se busca es ampliar el abanico de aplicaciones prácticas que un lenguaje lógico funcional pueda resolver, pues la integración de resolutores de restricciones en dominios concretos puede minimizar en parte la ineficiencia innata inherente a los lenguajes de programación declarativos [FHS03a].

Hasta el presente se han incluido, con mayor o menor éxito, restricciones de dominio finito y restricciones reales [AHLU96, Rod01, Lux01, FHS03c, FHSV07] en los principales sistemas lógico funcionales, como $\mathcal{TOY}$ y *Curry*. Sin embargo, aún no existe un marco teórico tan consolidado y universalmente aceptado como el esquema $CLP(\mathcal{D})$ para la programación lógica con restricciones. No obstante, sí existen varios precedentes de propuestas de combinación de la programación lógico funcional y la programación con restricciones (veánse por ejemplo [DGP92a, Lop92, MIS00]). Resumiremos con mayor detalle aquellas que, a nuestro juicio, han resultado ser más relevantes para la elaboración de esta tesis, analizando las principales ventajas e inconvenientes que ofrece cada una de ellas, con el fin de poner de manifiesto cuáles son sus limitaciones, las ideas que se han aprovechado en la propuesta de nuestro esquema *CFLP* para abordar tales limitaciones, y cuáles son las novedades que realmente se ofrecen en esta memoria con respecto a estos trabajos previos.

- El primer intento de combinar la programación lógica con restricciones y la programación lógico funcional en un esquema $CFLP(\mathcal{D})$ lo encontramos en la propuesta de J. Darlington, Y. K. Guo y H. Pull en [DGP92a, DGP92b], donde se introduce el paradigma *DCP* (*Definitional Constraint Programming*) para definir lenguajes lógico funcionales con restricciones. *DCP* es en realidad un esquema parametrizado por un dominio de restricciones $\mathcal{D}$, donde la idea que subyace a esta primera aproximación puede ser descrita informalmente a través de la ecuación

$$CFLP(\mathcal{D}) = CLP(FP(\mathcal{D}))$$

mediante la cual un lenguaje *CFLP* sobre el dominio de restricciones $\mathcal{D}$ puede ser visto como un lenguaje *CLP* sobre un dominio de restricciones extendido $FP(\mathcal{D})$, cuyas restricciones incluyan ecuaciones entre expresiones que involucren a su vez funciones definidas por el usuario, para ser resueltas mediante *estrechamiento*. En *DCP* no se permite el uso de restricciones en la definición de las funciones, con lo que la integración conseguida está bastante limitada. Por otra parte, las propiedades semánticas del esquema no están del todo aclaradas en los trabajos citados.

- En [MS94] se presenta el lenguaje $CFLP{-}L$, definido como una extensión del λ-cálculo para incluir variables lógicas, elecciones no deterministas y restricciones. La extensión está inspirada en el λ-cálculo con restricciones ya comentado en la sección anterior, con quien comparte también alguna de sus limitaciones, en particular la naturaleza estricta de las funciones. Como estructuras de base se consideran modelos iniciales de especificaciones algebraicas. Los modelos están restringidos a ser álgebras generadas por términos. Los autores alegan para ello que no tiene interés considerar dominios de base en los que haya elementos no representables como términos; pero uno de los aspectos importantes de las restricciones es su capacidad para representar implícitamente elementos del dominio, que no tienen por qué poderse representar como términos. Por ejemplo, la restricción $X * X = 2, X > 0$ representa implícitamente en $CLP(\mathcal{R})$ al número $\sqrt{2}$, que no es representable como término en el lenguaje, por lo que uno de los lenguajes con restricciones más representativos incumple el requisito pedido en $CFLP{-}L$. En cuanto a la semántica del lenguaje, se proporciona una semántica de continuaciones (*continuation semantics*) que determina la semántica operacional, pero no se aporta una semántica declarativa.

  Otras propuestas concernientes a la combinación de restricciones con programación funcional [DG89], deducción ecuacional [DG91], demostración automática [KKR90, CZ92, Rub94] y λ-cálculo [Man95] aparecieron al mismo tiempo. El interés de estos trabajos se circunscribe a restricciones sobre dominios simbólicos (términos, habitualmente), como son igualdad, desigualdad, prioridad según algún orden de términos, etc., estando por tanto alejados de nuestra idea de que la programación con restricciones se refiere a computación sobre dominios predeterminados cualesquiera. Se puede, de todos modos, descubrir una relación más estrecha en algunos trabajos relativos a problemas de unificación en dominios combinados [Bou93, KR94].

- El esquema propuesto por F. J. López-Fraguas en [Lop92, Lop94] es, sin duda, la propuesta más próxima y cercana al esquema $CFLP(\mathcal{D})$ presentado en esta tesis, ya que ha servido tanto de punto de partida en su desarrollo actual como de referencia a la hora de establecer los principales objetivos en base a sus limitaciones.

  De manera más detallada, el esquema de López-Fraguas proporciona resultados sobre la semántica declarativa de programas $CFLP(\mathcal{D})$ muy próxima a la ya conocida para el esquema $CLP(\mathcal{D})$. Además, con el fin de soportar una semántica perezosa para las funciones definidas por el usuario, los dominios de restricciones $\mathcal{D}$ se formalizan como estructuras continuas, con un *dominio de Scott* [GS90] y una interpretación continua de funciones y símbolos de predicado. Asimismo, los programas están constituidos por reglas de reescritura con restricciones para poder definir nuevas funciones por parte de los usuarios. Para esta clase de programas, se desarrolla una semántica declarativa de modelo mínimo, caracterizado como mínimo punto fijo, y una semántica operacional basada en estrechamiento perezoso con restricciones como principal mecanismo de cómputo, para la que se prueban resultados de corrección y completitud con respecto a la semántica declarativa.

  La semántica resultante posee interesantes propiedades, pero también algunas limitaciones. En particular, las definiciones de funciones han de ser de primer orden y deterministas y el uso de patrones en las definiciones de funciones tiene que ser simulado mediante el uso de restricciones especiales. Como novedad en nuestra propuesta, todas estas limitaciones se abordan en el nuevo esquema propuesto en esta tesis mediante la introducción de orden superior en el esquema y de funciones perezosas posiblemente indeterministas definidas sobre patrones. Para conseguir esta nueva aproximación, ha sido necesario desarrollar una nueva formulación de las bases teóricas del esquema $CFLP$, de manera que se pudieran adoptar e integrar las aproximaciones más fructíferas que sobre programación lógico funcional con indeterminismo y evaluación perezosa se han desarrollado en los últimos años, y que toman como punto de partida la lógica de reescritura basada en constructoras *CRWL* [GHLR96, GHLR99]. Este nuevo enfoque, basado en una lógica de reescritura con restricciones denominada $CRWL(\mathcal{D})$, la cual puede ser parametrizada por un dominio genérico $\mathcal{D}$, nos ha permitido obtener mejores resultados teóricos que los que ya se habían obtenido en el esquema de López-Fraguas, especialmente para la completitud del estrechamiento perezoso con restricciones, al tiempo que se ha conseguido dotar al nuevo esquema de una semántica lógica y algebraica apropiada, además de extender la ya existente de modelo mínimo, caracterizada también como mínimo punto fijo. La nueva semántica de modelos definida en esta tesis permite conservar las buenas propiedades conseguidas mediante la noción de 'estructura continua' con dominios de Scott como soporte y operaciones primi-

tivas continuas. En consecuencia, sigue siendo posible modelar en nuestro nuevo esquema $CFLP(\mathcal{D})$ lenguajes con la capacidad de realizar cómputos que involucran objetos infinitos definidos como límites de aproximaciones finitas. También se ha aprovechado del esquema de López-Fraguas la idea natural y suficientemente general de considerar programas como conjuntos de reglas condicionales para la definición de nuevas funciones y predicados.

Como instancia particular del esquema de López-Fraguas sobre la que se aplican los resultados obtenidos, se propone en [Lop92, Lop94] un lenguaje que permite aumentar la expresividad de la programación lógico funcional perezosa mediante el uso de desigualdades, tanto en programas como en respuestas. Sin embargo, en estos trabajos no se estudian las bases teóricas de otras instancias del esquema que también resultan de interés práctico, como el dominio de los números reales o los dominios finitos de los números enteros. Esta deficiencia de carácter más pragmática también ha servido de motivación en el desarrollo del nuevo esquema propuesto en esta tesis, de manera que en esta memoria no sólo se incluyen otras instancias interesantes del esquema sino que se proporcionan las bases para que cualquier usuario pueda formalizar sus propias instancias en base a las necesidades prácticas del problema que se quiere resolver. La implementación de todas estas instancias en un sistema actual como es $\mathcal{TOY}$, recogidas ahora de un modo uniforme e integrado en la presente tesis, es sin duda una de las aplicaciones más interesantes de nuestro esquema actual con respecto al esquema precedente de López-Fraguas, del que se ha conservado sin embargo la misma idea de plantear su implementación como un proceso de compilación de programas a *Prolog*.

Finalmente, otra cuestión técnica importante que no se aborda en [Lop92, Lop94] es la determinación de propiedades generales de los sistemas de resolución de restricciones. La nueva formalización de resolutor de restricciones sobre un dominio que se propone en la presente memoria junto a su especificación mediante reglas de transformación de almacenes constituye uno de los aspectos más relevantes de nuestra propuesta, ya que es otro de los pilares teóricos que permiten garantizar resultados de completitud más fuertes que los que se han obtenido en estos trabajos previos.

- Más recientemente, otro esquema *CFLP* ha sido propuesto en la tesis doctoral de Mircea Marin [Mar00]. Este trabajo parte también del esquema de López-Fraguas, el cual puede ser alternativamente descrito mediante la siguiente ecuación:

$$CFLP(\mathcal{D},\mathcal{C}) = FLP(\mathcal{C}) + CP(\mathcal{D})$$

la cual expresa la combinación de una componente lógico funcional cuya semántica operacional viene dada por un cálculo de estrechamiento perezoso con restricciones $\mathcal{C}$ y un esquema de programación con restricciones $CP(\mathcal{D})$, definido a partir de un dominio de restricciones $\mathcal{D}$ y de su resolutor de restricciones asociado. Los trabajos de Marin permiten extender este esquema del modo siguiente:

1. El esquema $CP(\mathcal{D})$ se extiende mediante el reemplazamiento del resolutor de restricciones del dominio $\mathcal{D}$ por una cooperación de diversos resolutores de restricciones, también sobre el dominio $\mathcal{D}$, de modo similar a como se propone en programación lógica con restricciones en los trabajos [Hon92, Hon94].
2. Se define un modelo distribuido alternativo para el esquema *CFLP*, extendido ahora mediante la cooperación de los diversos resolutores.

El resultado es el esquema $CFLP(\mathcal{D}, \mathcal{S}, \mathcal{C})$, descrito formalmente a través de la ecuación:

$$CFLP(\mathcal{D}, \mathcal{S}, \mathcal{C}) = FLP(\mathcal{C}) + CP(\mathcal{D}, \mathcal{S})$$

mediante el que es posible caracterizar una familia de lenguajes parametrizados por un dominio de restricciones $\mathcal{D}$, una estrategia $\mathcal{S}$ mediante la cual se define la cooperación de diversos resolutores de restricciones sobre $\mathcal{D}$ y un cálculo de estrechamiento perezoso con restricciones $\mathcal{C}$ para la resolución de restricciones que involucren funciones definidas por el usuario mediante reglas de reescritura con restricciones.

La principal ventaja de esta aproximación es que se basa en un sólido trabajo sobre cálculos de estrechamiento perezoso de orden superior definidos sobre álgebras de λ-términos y sistemas de reescritura de patrones, con los que se consigue así alcanzar una mayor expresividad al soportar λ-abstracción y unificación de orden superior [Ham96, Mar00, IMS01, KMI01, KMI03]. El interés teórico de esta propuesta ha servido de motivación y de punto de partida en el desarrollo de algunos de los trabajos más recientes que el autor de esta tesis ha realizado en [Vad07, Vad08] y que guardan relación, o tratan de extender, algunos de los aspectos más novedosos tratados en esta memoria, como la utilización de los llamados *árboles definicionales*, mediante los cuales es posible controlar y guiar de forma eficiente las computaciones que se realizan a través de un cálculo de estrechamiento perezoso. Sin embargo, a pesar de las ventajas teóricas que ofrece esta aproximación en *CFLP*, su principal inconveniente radica en que sus posibilidades de implementación son habitualmente mucho

menos eficientes, lo que supone sin duda un serio inconveniente para la consecución de los objetivos que inicialmente se han propuesto en el desarrollo de nuestro actual esquema *CFLP*. Esto ha motivado la decisión de que en esta tesis no se opte por utilizar un marco teórico basado en el λ-cálculo para abordar la introducción del orden superior en el esquema. En su lugar, se ha optado por utilizar sistemas de reescritura aplicativos condicionales, siguiendo la línea de los trabajos previos [GHR97, GHR01], más afines a los objetivos que se pretende alcanzar en esta tesis con respecto a la fundamentación de la semántica declarativa y operacional de nuestro esquema a través de la utilización de lógicas de reescritura y a sus posibilidades reales de implementación en sistemas *CFLP* actuales como $\mathcal{TOY}$ o *Curry*.

No obstante, la propuesta de Marin sí ha sido implementada en [MIS00] a través de un prototipo en *Mathematica*^{TM} [Wol96], un sistema de álgebra computacional originalmente desarrollado por Stephen Wolfram que ofrece un poderoso lenguaje de programación capaz de emular múltiples paradigmas utilizando reescritura de términos, y también mediante el sistema *OpenCFLP* en [KMI01]. Más recientemente, la línea de investigación propuesta en este esquema ha sido extendida en los trabajos [KMIC02, KMI03], en los que se propone la resolución de restricciones simbólicas mediante la colaboración de diversos resolutores de restricciones distribuidos en un entorno abierto como es Internet. Los resolutores actúan como proveedores de servicios de resolución de restricciones, de forma que el sistema *OpenCFLP* es capaz de usarlos sin tener que conocer necesariamente ni su ubicación ni los detalles concretos de su implementación.

La principal limitación y desventaja del esquema $CFLP(\mathcal{D}, \mathcal{S}, \mathcal{C})$ es, sin embargo, la falta de una semántica declarativa en la que las nociones formales de resolutor de restricciones y de respuesta correcta puedan ser fijadas con total precisión. Con el fin de abordar esta limitación, el esquema *CFLP* que se presenta en esta tesis propone una nueva formalización matemática de dominio de restricciones y de resolutor asociado, sobre la base de una semántica declarativa claramente establecida que trata de seguir la línea del esquema propuesto por López-Fraguas y de los buenos resultados allí obtenidos. Gracias a ello, y a pesar de que la cooperación de diferentes resolutores no se aborda en esta tesis, la flexibilidad y la potencia expresiva que ofrece nuestra propuesta actual permite definir también un tipo especial de dominios de restricciones denominado *dominios de coordinación* sobre los que asentar las bases de la coordinación de varios dominios y de sus resolutores. Las principales ideas de esta nueva aproximación para la integración de la cooperación de resolutores en programación lógico funcional con restricciones en un entorno que sin embargo no es distribuido, se presentan en los trabajos [EFHR+07a, EFHR+07b, EFHR+08], y se comentarán con mayor detalle en el último capítulo de esta memoria.

1.2. Técnicas de depuración declarativa

El desarrollo de programas software lleva asociado desde sus orígenes el problema de la detección de errores. Para ello se han intentado definir técnicas y herramientas que automaticen, al menos parcialmente, esta tarea. La mayoría de las técnicas de depuración de síntomas de errores observados en tiempo de ejecución se basan en la inspección de algún tipo de traza del cómputo. Ciertos paradigmas de programación (en particular aquellos que incluyen resolución de restricciones y/o evaluación perezosa como los que se abordan en esta tesis) tienen un mecanismo de cómputo tan complejo que no resulta viable reproducir los detalles operacionales en la traza empleada para depurar.

Para resolver esta dificultad se han considerado técnicas de depuración alternativas basadas en un enfoque más adecuado denominado *depuración declarativa*, el cuál fue propuesto inicialmente en el contexto de la programación lógica [Sha82]. Su principal ventaja radica en el hecho de que sus ideas pueden abstraerse para dar lugar a un esquema general de depuración declarativa que resulta ser aplicable a todo tipo de paradigmas, incluyendo paradigmas no declarativos. Exponemos a continuación detalladamente las ideas básicas de este esquema, ya que han servido de base en el desarrollo de la segunda parte de esta memoria para el paradigma lógico funcional con restricciones.

De acuerdo con [Nai97], *la depuración declarativa puede entenderse como la búsqueda de un nodo crítico en un árbol de cómputo*. Este tipo de árboles se construye a posteriori con el fin de representar la estructura de un cómputo con resultado erróneo que ha sido calificado por el usuario, en su nivel más externo, como un síntoma de error. Cada nodo del árbol se corresponderá con el resultado de un subcómputo, permitiendo así representar la computación de algún resultado observable y las dependencias entre los distintos resultados intermedios. En particular, la raíz del árbol representará el resultado del cómputo principal, y el resultado asociado a cada nodo estará determinado por los resultados de sus nodos hijos. En este sentido, cada nodo representará un paso de cómputo y deberá tener asociado el fragmento de código responsable de dicho paso. Así, el propósito del depurador será detectar fragmentos erróneos de programa a partir del árbol de cómputo. Para ello, debe explorar el árbol de cómputo buscando lo que se denominan *nodos críticos*, los cuales computan un resultado incorrecto a partir de hijos cuyos resultados son todos correctos; tales nodos críticos han de apuntar a un fragmento de programa incorrecto como responsable del cómputo erróneo. Obsérvese, sin embargo, que un nodo puede ser erróneo aunque su fragmento de código asociado sea correcto, debido a que su resultado puede haberse obtenido partiendo de resultados ya erróneos. Por eso, el depurador sólo detectará como fragmentos de código erróneos los correspondientes a nodos erróneos sin ningún hijo erróneo (resultado incorrecto obtenido a partir de resultados correctos), a los que hemos llamado nodos críticos. El depurador

debe ser capaz de decidir cuándo el resultado asociado a un nodo es erróneo. Esto se hará normalmente mediante preguntas a un oráculo externo (usualmente el usuario con algún tipo de soporte semiautomático), quien tiene un conocimiento declarativo subyacente de la semántica esperada del programa, la denominada *interpretación pretendida* del programa.

La propuesta anterior establece una distinción clara entre el concepto de árbol de cómputo y el algoritmo de depuración utilizado, distinción que no existía en los primeros depuradores para lenguajes lógicos. Además, no hace referencia a ningún lenguaje ni paradigma particular, por lo que puede ser usado para definir depuradores declarativos genéricos. Diferentes tipos de errores requerirán diferentes tipos de árbol y también diferentes definiciones del concepto de nodo erróneo. Es decir, constituirán distintas *instancias* del esquema general, pero en todas ellas serán aplicables las mismas ideas. Así por ejemplo, en [Nai97] se proponen distintas instancias para la programación lógica, la programación funcional y la programación orientada a objetos que permiten consideran dos tipos habituales de síntomas de error susceptibles de ser tratados mediante depuración declarativa:

- *Respuestas incorrectas*: se obtiene una respuesta inesperada para un objetivo determinado. A este tipo de síntomas de error se les suele denominar *síntomas positivos*.

- *Respuestas perdidas*: en el conjunto de todas las respuestas obtenidas para un objetivo dado falta alguna respuesta esperada. Un caso particular de esta situación se tiene cuando no se obtiene ninguna respuesta y se esperaba al menos una. A este tipo de síntomas de error se les suele denominar *síntomas negativos*.

Como veremos, en programación funcional sólo se considera el primer tipo de respuestas, las respuestas incorrectas, mientras que en programación lógico funcional (con o sin restricciones), al igual que sucede en programación lógica, podemos hablar tanto de respuestas incorrectas como de respuestas perdidas.

En las siguientes secciones describimos con mayor detalle cómo se han aplicado las técnicas de depuración declarativa para el desarrollo de herramientas para diversos lenguajes y plataformas en los distintos paradigmas de depuración declarativa: la programación lógica con restricciones [TF00, FLT03], la programación funcional [NF94, Nil01b, PN03a] y la programación lógico funcional [CLR01, CR04, Cab04]. Fuera del paradigma declarativo, las mismas ideas se han utilizado en programación imperativa [SF89, FSKG92] y en el diseño de lenguajes de programación para bases de datos deductivas [ST90].

1.2.1. Depuración declarativa de lenguajes lógicos

Como ya se ha comentado anteriormente, las primeras ideas acerca de la depuración declarativa provienen del paradigma de la programación lógica. En concreto, fue E.Y. Shapiro quien en 1982 [Sha82] introdujo, bajo el nombre de *Depuración Algorítmica*, un método para localizar errores en programas lógicos para el lenguaje *Prolog* [SS86] basado en la semántica declarativa de estos lenguajes. Describimos a continuación, a grandes rasgos, el proceso de depuración propuesto en este trabajo, ya que en el se condensan gran parte de las ideas clave de la actual depuración declarativa.

La depuración comienza cuando el usuario observa que un programa lógico produce un resultado inesperado como resultado del cómputo de un objetivo. El depurador entonces repite el cómputo paso a paso, pero preguntando a cierto oráculo (normalmente el usuario) si los resultados intermedios obtenidos en cada paso son los esperados. De esta forma, comparando el modelo presentado por el programa con el modelo pretendido conocido por el oráculo se puede localizar el error. Dicho modelo pretendido es, en el caso del trabajo de Shapiro, un subconjunto de la *base de Herbrand*, es decir un conjunto de átomos sin variables.

Posteriores trabajos de G. Ferrand [Fer87] y J. W. Lloyd [Llo87a, Llo87b] continúan y amplían la línea propuesta por Shapiro. En [Fer87] se extiende la noción de modelo pretendido a conjuntos de átomos con variables, mientras que en sus trabajos, J.W. Lloyd [Llo87a, Llo87b], en lugar de tratar con programas formados por cláusulas de Horn considera programas lógicos más generales formados por fórmulas $A \leftarrow W$ con A un átomo y W una fórmula de primer orden general. En todas las propuestas los depuradores presentados son meta-intérpretes que repiten el cómputo erróneo para analizarlo paso a paso.

La programación lógica con restricciones amplía el marco tradicional de la programación lógica, definiendo las restricciones atómicas sobre un cierto dominio $\mathcal{D}$, de forma que puedan formar parte de cláusulas y objetivos. Existen diversos trabajos en este campo que intentan establecer un marco para depurar programas lógicos incluyendo restricciones. Algunos de los más destacados se han realizado en el marco del proyecto *DiSCiPl* (véase, `http://discipl.inria.fr/`); en particular destacamos los trabajos [TF00, FLT03], los cuales han inspirado el diseño de las técnicas específicas de depuración declarativa en el esquema $CFLP(\mathcal{D})$ que se presentan en la segunda parte de esta memoria.

1.2.2. Depuración declarativa de lenguajes funcionales perezosos

Repasamos en esta sección las principales propuestas de depuración declarativa para el paradigma funcional, y en particular, para los lenguajes funcionales perezosos, ya que son los más próximos a nuestro marco de programación lógico funcional perezosa con restricciones. Analizaremos por qué la depuración declarativa es particularmente

interesante para este tipo de lenguajes y cuáles han sido los principales trabajos que se han realizado en este área.

Al contrario que en programación lógica, donde es habitual encontrar depuradores de traza integrados en los sistemas disponibles, tradicionalmente las implementaciones de lenguajes funcionales perezosos tipo *Haskell* [PHAB+02] han prescindido de la incorporación de herramientas tales como depuradores. Sin embargo, según indica P. Wadler en [Wad98], esta deficiencia dificulta la difusión de este tipo de lenguajes fuera del ambiente puramente académico, donde el desarrollo de aplicaciones reales demanda siempre la existencia de tales herramientas. La causa principal a la que puede achacarse esta carencia en el caso de la depuración es, sin duda, el alto nivel de abstracción de estos lenguajes, pues dificulta enormemente el uso de herramientas tradicionales tales como los depuradores de traza utilizados en programación lógica. En particular, características tales como el orden superior, el polimorfismo o la evaluación perezosa, hacen que la ejecución de un cómputo sea difícilmente "observable" por el usuario. Por esta razón, la depuración declarativa resulta particularmente útil en estos lenguajes. En cambio, en el caso de los lenguajes funcionales impacientes, los métodos tradicionales de depuración sí continúan siendo eficaces (véase, por ejemplo, [TA95]).

Resulta importante resaltar que, a diferencia de lo que ocurre en el paradigma lógico, en los trabajos sobre programación funcional el caso de las respuestas perdidas no es normalmente considerado. Esto se debe a que en la programación funcional no se tienen soluciones múltiples representadas como sustituciones o restricciones para un objetivo dado, sino una única respuesta (o un fallo). Por tanto, si en la evaluación de una expresión tenemos una respuesta perdida, existen dos posibilidades:

- Se ha obtenido una respuesta, pero esta respuesta ha de ser forzosamente una respuesta incorrecta. En este caso, el síntoma negativo tiene asociado un síntoma positivo (la respuesta incorrecta obtenida en lugar de la respuesta perdida).

- El cómputo ha fallado. En este segundo caso, se considera el fallo como una respuesta errónea, y por tanto, se puede transformar igualmente el síntoma negativo en un síntoma positivo (véase, por ejemplo, [NF94]).

En cualquier caso, la depuración declarativa de estos lenguajes también plantea nuevas dificultades, como se señala ya en los primeros trabajos publicados sobre este tema en [Nai92, NF92, NF94].

En [NF94], H. Nilsson y P. Fritzson proponen un árbol de cómputo para programación funcional perezosa basado en la técnica denominada de *estrictificación*. También se presenta en este trabajo un depurador declarativo llamado *LADT* para el lenguaje *Freja* [Nil01a], un subconjunto del lenguaje funcional perezoso *Miranda*

[Tur85]. En relación con esta implementación, se discuten dos problemas que siguen hoy en día contándose entre los principales obstáculos de la implementación de un depurador declarativo realmente útil: la cantidad de preguntas formuladas al usuario y el enorme espacio requerido por el árbol de cómputo generado. Para el segundo problema, los autores proponen una técnica consistente en producir en cada momento sólo la parte del árbol que se esté inspeccionando. Si el error no puede ser localizado en este fragmento, entonces el programa ha de ser recomputado con el fin de producir otro fragmento del árbol de cómputo, y así sucesivamente. Como se verá en la segunda parte de esta memoria, ambos problemas también siguen estando presentes en el planteamiento general del nuevo esquema *CFLP*, especialmente en la implementación de las herramientas de depuración propuestas en esta tesis, tanto para el diagnóstico declarativo de respuestas incorrectas como de respuestas perdidas.

En [SN95] y en [NS97], H. Nilsson y J. Sparud establecen el árbol de cómputo que se utilizará frecuentemente en trabajos posteriores (por ejemplo, [NB96, Pop98, Spa99, Nil01b]), el denominado *EDT* (del inglés, *Evaluation Dependence Tree*). Además, estos autores proponen dos técnicas para la obtención de dicho árbol: la primera mediante una transformación de la máquina abstracta y la segunda mediante una transformación de programas, de forma que el programa transformado sea capaz de producir como resultado el *EDT*, señalando además las ventajas (eficiencia en la transformación de la máquina abstracta y portabilidad en el caso de la transformación de programas) que se consiguen en cada caso. En esta tesis se ha optado sin embargo por la segunda técnica, siguiendo el enfoque propuesto en trabajos previamente desarrollados sobre depuración declarativa en lenguajes lógico funcionales [CR04, Cab04, Cab05], donde tanto la traducción de programas de la que se ha hecho uso para implementar la depuración de respuestas incorrectas como el prototipo desarrollado en el sistema $\mathcal{TOY}$, pueden considerarse como una modificación de estos trabajos anteriores. La principal aportación del trabajo del doctorando en esta tesis para el caso de la depuración de respuestas incorrectas desarrollada por nuestro grupo de investigación, ha consistido en la utilización de la lógica de reescritura con restricciones $CRWL(\mathcal{D})$, mediante la cual ha sido posible extender y establecer los resultados teóricos que justifican el buen comportamiento de la técnica de depuración propuesta.

Finalmente, y al contrario que en el caso de la programación lógica, hacemos notar que ninguno de estos trabajos anteriores, propuestos para el paradigma de programación funcional perezosa, permite relacionar los árboles de cómputo con un sistema de inferencias lógicas, lo que supone un fuerte distanciamiento con respecto a los planteamientos iniciales que se proponen en esta tesis para abordar las técnicas de depuración declarativa en el esquema $CFLP(\mathcal{D})$. En [NS97] se propone su especificación mediante semántica denotacional, mientras que en el resto de trabajos se da tan solo una descripción verbal.

1.2.3. Depuración declarativa de lenguajes lógico funcionales perezosos

Los argumentos de P. Wadler en [Wad98] sobre la necesidad de integrar herramientas tales como depuradores en los entornos de desarrollo para lenguajes funcionales son también perfectamente aplicables a los lenguajes lógico funcionales (con o sin restricciones).

Son L. Naish y T. Barbour quienes en [NB95] se plantean por primera vez la depuración de lenguajes que permiten combinar las principales propiedades de los paradigmas lógico y funcional. El lenguaje elegido es una extensión del sistema lógico *NU-Prolog* descrita en [Nai91] y las ideas propuestas se basan en gran medida en la transformación de las características funcionales del programa al paradigma lógico. Los árboles de prueba considerados están constituidos por nodos 'lógicos' y por nodos 'funcionales', y el depurador trata cada nodo de manera acorde con el paradigma al que pertenece. Con este planteamiento, la programación lógico funcional no se presenta como un nuevo paradigma con su marco teórico propio, sino como una combinación de características de los paradigmas lógico y funcional. En consecuencia, esta idea también se extiende al tratamiento de la depuración declarativa, por lo que no constituirá un referente representativo en el marco de esta tesis.

En [CLR01] se propone la utilización del cálculo semántico de la lógica *CRWL* para la definición de unos árboles de prueba que permitan realizar la depuración declarativa de respuestas incorrectas dentro del paradigma lógico funcional sin restricciones. Estos árboles corresponden, al igual que ya sucedía en el caso de la programación lógica, con inferencias lógicas. Como consecuencia, es posible probar la corrección del método. En [CR02] se amplían los resultados de [CLR01] mediante la definición precisa de una transformación de programas que permite llevar los resultados teóricos obtenidos a la práctica en el lenguaje $\mathcal{TOY}$. Para ello, se prueba primero que el programa transformado es correcto desde el punto de vista de los tipos, y a continuación, que el árbol de prueba obtenido se corresponde efectivamente con el árbol de prueba definido en [CLR01]. También se propone en este trabajo un método para disminuir el número de preguntas que se le plantean al oráculo. La idea consiste en deducir, si es posible, la validez (o no validez) de un nodo del árbol de prueba a partir de la validez (o no validez) de otros nodos ya visitados. La posibilidad de extender los planteamientos propuestos por nuestro grupo de investigación en estos trabajos previos al campo de la programación lógico funcional con restricciones, ha sido una de las mayores motivaciones que nos han llevado a desarrollar el nuevo esquema genérico $CFLP(\mathcal{D})$ en la primera parte de esta tesis, y a la adaptación en la segunda parte de las técnicas ya estudiadas en depuración declarativa de respuestas incorrectas, tomando ahora como base la lógica para la reescritura con restricciones $CRWL(\mathcal{D})$. Así, gracias a la relación existente entre la semántica de nuestro esquema y la definición del árbol de cómputo como un árbol de prueba en $CRWL(\mathcal{D})$, se ha podido establecer también de modo consistente la corrección del método seguido.

Además, dado que nuestro esquema abarca tanto a la programación lógica con restricciones como a la programación funcional, los resultados obtenidos en esta tesis representan también resultados de corrección útiles en el campo de la depuración declarativa de lenguajes funcionales con restricciones.

En cuanto a los sistemas existentes de depuración declarativa en *FLP* (algunos de los cuales son también válidos en sistemas *CFLP*), destacamos la herramienta $\mathcal{DDT}$ [CR04] incorporada al sistema $\mathcal{TOY}$ y que se encuentra disponible en `http://toy.sourceforge.net`, así como el depurador que se ha incorporado al compilador de *Curry* [Han06] sobre la versión desarrollada en la Universidad de Münster por Wolfgang Lux. Más recientemente, en [Cab05] se extiende la herramienta $\mathcal{DDT}$ para la depuración declarativa de respuestas incorrectas en $\mathcal{TOY}$ que involucran el tratamiento de programas con restricciones de igualdad y desigualdad. La extensión efectiva de la herramienta sobre otros dominios de interés práctico como el dominio de los números reales o los dominios finitos de números enteros es otra de las propuestas que se plantean en la presente tesis como un posible trabajo futuro.

Es importante resaltar que todos estos sistemas, bien sean aplicables a la programación lógico funcional con o sin restricciones, solo sirven para la depuración declarativa de respuestas incorrectas, por lo que no permiten tratan el caso particular de la diagnosis de respuestas perdidas. Ya hemos comentado anteriormente que dentro del paradigma funcional, las respuestas perdidas pueden reducirse al caso de las respuestas incorrectas. Sin embargo, en los lenguajes lógico funcionales (y por tanto también en los lenguajes lógico funcionales con restricciones), debido precisamente a la existencia de cómputos indeterministas, vuelven a encontrarse los dos síntomas de error que ya aparecían en el caso de la programación lógica. Esta es sin duda una de las propuestas más novedosas ofrecidas en esta tesis en relación a los trabajos anteriormente citados, ya que hasta el momento no se disponía de ningún marco teórico adecuado que sirviera para resolver las dificultades semánticas que conlleva la depuración declarativa de respuestas perdidas en programación lógico funcional con o sin restricciones. El diagnóstico de respuestas perdidas plantea la necesidad de controlar la recolección de las respuestas calculadas en el espacio de búsqueda finito correspondiente a un cierto objetivo *CFLP*. Por este motivo, la obtención de árboles de prueba adecuados para el caso de las respuestas perdidas no se puede realizar mediante su inferencia en un cálculo semántico como pueda ser *CRWL*, o de manera más general $CRWL(\mathcal{D})$, sino mediante su definición en un nuevo cálculo semántico que resulte apropiado para la representación de la recolección de respuestas calculadas en un cómputo erróneo. Para conseguir este propósito, se ha seguido la línea de trabajos previamente desarrollados para el esquema *CLP* en [TF00, FLT03]. Sin embargo, la componente de estrechamiento perezoso de los lenguajes *CFLP* exige ahora considerar extensiones no triviales de las técnicas de depuración de respuestas perdidas conocidas para *CLP* y de los métodos formales necesarios para justificar su corrección.

Por último, otra de las novedades que aporta la presente tesis es el desarrollo de un prototipo de depurador declarativo basado en el método propuesto de depuración de respuestas perdidas, el cual nos permite trabajar con restricciones de igualdad y desigualdad sintáctica sobre el dominio de Herbrand (aunque los mismos principios de diseño serían también aplicables sobre otros dominios de restricciones de interés práctico, como los números reales o los dominios finitos). Si bien el prototipo actual presenta aún limitaciones en cuanto a su aplicación práctica en un sistema *CFLP* como pueda ser $\mathcal{TOY}$ o *Curry* (especialmente a la hora de plantear preguntas al usuario durante una sesión de depuración, ya que con frecuencia suelen ser demasiado complejas), creemos que en esta tesis se muestran con suficiente claridad las técnicas de implementación que podrían ser de utilidad en su implantación e integración real en este tipo de sistemas en un futuro próximo.

Concluimos esta sección comentando brevemente algunas otras propuestas de interés en el área de la depuración de programas que han influido en la elaboración de esta memoria. El entorno de desarrollo gráfico *CIDER* presentado en [HK01] (en la línea de trabajos similares como [TF00]), incluye un depurador gráfico de trazas para el lenguaje *Curry*. El depurador muestra el proceso de evaluación de una expresión paso a paso. La expresión se muestra como un árbol donde la subexpresión que va a ser reducida en el siguiente paso aparece marcada en rojo. El usuario tiene la posibilidad de fijar puntos de parada en el código para 'saltar' aquellas partes del programa que no está interesado en inspeccionar. Siguiendo esta idea y las ventajas que tiene trabajar con un entorno gráfico en la depuración de programas, las herramientas de depuración propuestas en esta tesis también proporcionan una interfaz gráfica implementada en *Java*, extensión de la ya proporcionada por la herramienta gráfica de depuración declarativa $\mathcal{DDT}$ [CR04, Cab05] en el sistema $\mathcal{TOY}$. Mediante esta interfaz gráfica del depurador, el usuario tiene la posibilidad de inspeccionar el árbol de depuración con el fin de examinarlo en su conjunto y poder marcar como fiables todas aquellas funciones que desee, así como eliminar los nodos asociados a estas funciones durante la propia navegación visual, lo que facilita la localización de nodos críticos.

Al igual que en el caso de la programación lógica, también en el paradigma lógico funcional se ha empleado el método de diagnosis abstracta, también llamada *diagnosis declarativa*. Aunque también se trata de comparar la semántica del programa con la interpretación pretendida del mismo, no se trata de un tipo de depuración declarativa, sino de una técnica diferente en la que se utiliza *interpretación abstracta* [CC77, CC92] para tratar de probar ciertas propiedades que se cumplen en el modelo pretendido y que el usuario indica, generalmente mediante el uso de aserciones. Una ventaja con respecto a la depuración declarativa es que no se precisa de un síntoma inicial, ni habitualmente interacción alguna con el usuario durante el proceso de depuración. Un caso sencillo pero ilustrativo de esta técnica son las declaraciones de tipo indicadas por el usuario y el correspondiente análisis de tipos en tiempo de

compilación. El sistema *CIAO* [CLIP97] desarrollado en la Universidad Politécnica de Madrid propone el uso de aserciones con tres propósitos diferentes:

1. La detección de errores en tiempo de compilación mediante diagnosis abstracta.

2. La detección de síntomas positivos o negativos durante la ejecución de un programa sin la necesidad de la intervención del usuario.

3. Evitar algunas de las preguntas al usuario utilizando las aserciones como oráculo cuando ello sea posible.

En [ABCF03, ACF02] se siguen también estas ideas, pero incorporando además un mecanismo basado en técnicas de desplegado para poder tratar de corregir automáticamente el programa. Como es habitual en este tipo de depuradores, no se requiere un síntoma de incorrección inicial, aunque sí una especificación (parcial) ejecutable del modelo pretendido. Los autores presentan además un prototipo de nombre *BUGGY* disponible en `http://www.dsic.upv.es/users/elp/soft.html`.

En comparación con todos estos métodos de depuración que emplean técnicas de análisis y abstracción para detectar errores en tiempo de compilación, las técnicas de depuración declarativa presentadas en esta tesis (al igual que le ocurre en general a la depuración declarativa), tienen el inconveniente ya mencionado de que las preguntas realizadas al oráculo durante el diagnóstico de un síntoma de error pueden ser demasiado complejas. Para aliviar este problema se han intentado diversas soluciones alternativas al uso de especificaciones parciales ejecutables del programa, como son la inferencia de ciertas respuestas a partir de respuestas anteriores [CR02] o el diseño de árboles de cómputo ajustados a las necesidades de un problema de depuración específico sobre un dominio [FLT03]. La profundización de esta línea de trabajo, así como la optimización de la eficiencia de las herramientas de depuración, son parte de los retos más importantes de la investigación actual en este campo, y se plantean en esta tesis como una parte relevante del trabajo futuro a realizar.

Otro inconveniente de los métodos de depuración declarativa es la sobrecarga de la ejecución causada por la construcción del árbol de cómputo. Las propuestas más recientes ideadas para hacer frente a este problema incluyen técnicas de construcción gradual de los árboles de cómputo [PN03a, PN03b]. La tesis doctoral de B. Pope [Pop07] incluye una buena exposición de las dificultades prácticas del desarrollo de herramientas efectivas de depuración declarativa en el contexto del lenguaje funcional perezoso *Haskell* [Bir98, PHAB+02], que son relevantes también para otros lenguajes declarativos multiparadigma. En particular, el estudio de esta clase de técnicas en el esquema genérico $CFLP(\mathcal{D})$ constituye un buen punto de partida para su adaptación y aplicación a las herramientas de depuración declarativa propuestas en la segunda parte de esta memoria, lo que nos permitiría mejorar su eficiencia y favorecer tanto su desarrollo como su difusión.

1.3. Objetivos y estructura de la tesis

En los apartados siguientes precisamos con mayor detalle cuáles son los principales objetivos y el plan de trabajo que se ha seguido para la realización de esta tesis, así como la organización de la presente memoria, que sigue la pauta marcada en la descripción de estos objetivos. Describimos también, para cada uno de ellos, el desglose en capítulos y apéndices de los que se compone la presente tesis, y resumimos cuáles han sido las principales publicaciones en las que ha participado el autor y que han servido como base para la redacción de la misma. Las referencias detalladas de todas estas publicaciones pueden ser consultadas en el Apéndice C.

El primer objetivo de esta tesis ha consistido en desarrollar una propuesta de marco genérico para el paradigma de programación lógico funcional perezosa con restricciones, de forma que permita integrar de una manera natural los fundamentos teóricos que por separado se han desarrollado tanto para la programación lógica con restricciones como para la programación lógico funcional perezosa. Esta integración ha sido concebida con el fin de poder superar algunas de las principales limitaciones que exhiben las aproximaciones previas al paradigma *CFLP* y que han sido brevemente comentadas en la Subsección 1.1.4, como son la falta de una semántica declarativa clara y concisa que permita el uso de funciones perezosas de orden superior posiblemente indeterministas, la formulación precisa de la noción de resolutor de restricciones o la obtención de resultados de completitud más generales para una semántica operacional basada en estrechamiento que puedan ser aplicados a diversas instancias de interés práctico del esquema. Así se ha pretendido poder sacar el máximo provecho con la integración de ambos paradigmas al permitir extender de una forma sencilla los trabajos más recientes que han sido realizados por separado en cada una de estas áreas. Describimos a continuación, de forma más detallada, los capítulos de esta memoria que desarrollan este primer objetivo de la tesis.

- El Capítulo 2 presenta las bases teóricas que sirven de fundamento al esquema genérico $CFLP(\mathcal{D})$, a cuyo estudio se dedica la primera parte de esta memoria. Gran parte del material presentado en este capítulo ha sido adaptado de nuestra publicación preliminar [LRV04a] y de su versión extendida de revista [LRV07]. Sin embargo, la actual presentación del capítulo mejora ambas publicaciones mediante la incorporación de un tratamiento explícito de una disciplina de tipos polimórficos en el estilo de Hindley-Milner-Damas [DM82], así como una presentación mejorada de las nociones de dominio de restricciones, resolutor y de sus propiedades formales. De acuerdo con esta nueva presentación, primero se introducen los conceptos preliminares sobre disciplina de tipos y signaturas que resultan esenciales en la formalización matemática de un dominio de restricciones $\mathcal{D}$ y de un resolutor asociado a ese dominio. A continuación, se presentan varias instancias particulares de interés práctico del esquema $CFLP(\mathcal{D})$, como son las proporcionadas por el dominio de Herbrand

$\mathcal{H}$, el dominio de los reales $\mathcal{R}$ y el dominio $\mathcal{FD}$ de restricciones de dominio finito sobre los números enteros. Por último, se formaliza la sintaxis de los programas en el contexto del nuevo marco de programación proporcionado por el esquema $CFLP(\mathcal{D})$ y se muestran ejemplos concretos para cada uno de los lenguajes presentados $CFLP(\mathcal{H})$, $CFLP(\mathcal{R})$ y $CFLP(\mathcal{FD})$.

- El Capítulo 3 también parte del material presentado en las publicaciones [LRV04a, LRV07] y tiene como objetivo introducir una semántica declarativa para $CFLP(\mathcal{D})$-programas basada en una nueva noción de interpretación sobre un dominio de restricciones $\mathcal{D}$. Usaremos esta clase de interpretaciones para definir dos clases de semánticas de modelos, denominadas, respectivamente, semántica débil y semántica fuerte. Demostramos la existencia de un modelo mínimo para cada una de estas dos semánticas, caracterizado como el mínimo punto fijo de un operador de transformación sobre interpretaciones, e investigamos la relación existente entre los dos modelos mínimos obtenidos. Por último, presentamos una lógica para la reescritura con restricciones denominada $CRWL(\mathcal{D})$ parametrizada por un dominio de restricciones $\mathcal{D}$, cuyo propósito es el de proporcionar un marco lógico para la programación en el esquema $CFLP(\mathcal{D})$ y una forma alternativa de caracterizar la semántica declarativa de los programas, en la línea de trabajos previos para programación lógico funcional sin restricciones [GHLR96, GHLR99]. Formalizamos un cálculo lógico para $CRWL(\mathcal{D})$ e investigamos sus principales propiedades teóricas, en especial la relación existente entre la derivación formal en este cálculo y las dos semánticas de modelos propuestas.

- El Capítulo 4 proporciona varios métodos de resolución de objetivos que pueden ser usados para describir formalmente la semántica operacional del esquema $CFLP(\mathcal{D})$ y que sirven de base teórica a implementaciones reales desarrolladas en el sistema $\mathcal{TOY}$ sobre instancias concretas de nuestro esquema. Sobre la base de la semántica declarativa introducida en el capítulo anterior mediante la lógica $CRWL(\mathcal{D})$ es posible definir, de una manera clara y precisa, las nociones necesarias de objetivo, respuesta y solución. A partir de ellas, extendemos el esquema genérico $CFLP(\mathcal{D})$ mediante la propuesta de una semántica operacional basada en dos cálculos de estrechamiento perezoso con restricciones, los cuales pueden ser parametrizados por un resolutor de restricciones sobre el dominio $\mathcal{D}$ considerado. El material propuesto en este capítulo para la presentación de ambos cálculos, presentados ahora de una manera homogénea e integrada, está basado en las publicaciones [LRV04b, Vad05], incorporando ahora algunas propiedades y métodos adicionales, como la denominada propiedad de partición de deducciones en $CRWL(\mathcal{D})$ o el algoritmo de transformación de $CFLP(\mathcal{D})$-programas. En concreto, de la publicación [LRV04b] hemos tomado el cálculo de estrechamiento perezoso con restricciones denominado $CLNC(\mathcal{D})$, planteado como una extensión del presentado en [GHLR96, GHLR99] para la

programación lógico funcional sin restricciones, y para el que demostramos su corrección y completitud con respecto a la semántica declarativa de $CFLP(\mathcal{D})$-programas formalizada mediante $CRWL(\mathcal{D})$. En segundo lugar, de la publicación [Vad05] proceden las ideas utilizadas en este capítulo para presentar un refinamiento del cálculo $CLNC(\mathcal{D})$ basado en el uso de árboles definicionales, mediante los que es posible asegurar que todos los pasos de estrechamiento efectuados son realmente necesarios en el cómputo. El cálculo así obtenido, denominado $CDNC(\mathcal{D})$, sigue la línea de trabajos previos en programación lógico funcional de primer orden [Vad03a, Vad03b] y programación lógico funcional de orden superior [Vad07] (en ambos casos, sin un tratamiento explícito de un sistema de restricciones paramétricamente dado), cuyo propósito es de poder controlar eficientemente la computación en $CFLP(\mathcal{D})$. Este nuevo cálculo de estrechamiento con restricciones dirigido por demanda conserva las propiedades de corrección y completitud de $CLNC(\mathcal{D})$ y mantiene las buenas propiedades mostradas para las estrategias de estrechamiento necesario [AEH94, AEH00] y dirigido por demanda [LLR93] en programación lógico funcional perezosa. Más aún, como novedad ofrecida en esta tesis con respecto a [Vad05], demostramos que el cálculo $CDNC(\mathcal{D})$ puede ser aplicado a cualquier $CFLP(\mathcal{D})$-programa aunque éste no admita directamente un árbol definicional, gracias a la aplicación de un algoritmo de transformación de programas con restricciones que es capaz de preservar su semántica declarativa.

Nuestro segundo objetivo fundamental en esta tesis ha consistido en la aplicación del nuevo marco genérico $CFLP(\mathcal{D})$ y sus buenas propiedades tanto lógicas como operacionales al estudio de una extensión del método de depuración declarativa de *respuestas incorrectas* propuesto en [CLR01, CR04, Cab04] al paradigma lógico funcional con restricciones. Asimismo, hemos utilizado dicho marco teórico para poder extender el método de depuración declarativa al caso de *respuestas perdidas* en $CFLP(\mathcal{D})$, el cual no había sido estudiado antes para el caso de la programación lógico funcional perezosa. De forma más concreta, los capítulos de esta memoria que desarrollan este objetivo son los siguientes.

- Con el Capítulo 5 comienza la segunda parte de esta memoria, en la que se estudian técnicas de depuración declarativa para el esquema $CFLP(\mathcal{D})$. En primer lugar, se presenta un método declarativo de diagnosis de respuestas computadas incorrectas en este esquema. Para ello, se han utilizado las ideas propuestas en la publicación [CRV06a], donde se esboza tan solo la propuesta de nuestro método, ya que al tratarse de una publicación perteneciente a la categoría de póster, la mayor parte del material utilizado en este capítulo no ha sido aún publicada (aunque puede ser consultado en [CRV06b]). No obstante, el método propuesto en este capítulo puede considerarse como una extensión natural (si bien no trivial) de otros trabajos previos realizados en programación

lógico funcional perezosa sin restricciones [CLR01, CR02, Cab04], gracias a los sólidos cimientos semánticos que la lógica para la reescritura $CRWL(\mathcal{D})$ proporciona ahora en el contexto actual de *CFLP*. Así, en el desarrollo de este capítulo de la tesis se han utilizado las ideas expuestas en el Capítulo 2 sobre la semántica declarativa de los programas, con el fin de obtener en primer lugar una noción adecuada de interpretación pretendida de un programa y de un tipo de árboles de prueba apropiados en el cálculo $CRWL(\mathcal{D})$ que jueguen el papel de árboles de cómputo. Proporcionamos resultados teóricos que demuestran que el método de depuración propuesto es lógicamente correcto para cualquier sistema de resolución de objetivos cuyas respuestas computadas sean consecuencia lógica del programa en el sentido de la lógica para la reescritura $CRWL(\mathcal{D})$, como ocurre con los cálculos de estrechamiento perezoso con restricciones presentados en el Capítulo 4. Además, presentamos un prototipo de depurador automático desarrollado en el sistema $\mathcal{TOY}$ como una extensión de la herramienta $\mathcal{DDT}$ [CR04] ya existente en programación lógico funcional perezosa sin restricciones (y con restricciones de igualdad y desigualdad sintáctica de Herbrand en [Cab05]) para el caso de otros dominios de restricciones de interés presentados en esta tesis, como es el dominio de los números reales.

- En el Capítulo 6 complementamos el estudio de las técnicas de depuración declarativa iniciado en el capítulo anterior presentando un novedoso método para el diagnóstico de respuestas computadas incompletas en el esquema $CFLP(\mathcal{D})$, basándonos en consideraciones similares a las que aparecen en [TF00] para el caso del esquema $CLP(\mathcal{D})$. En esta ocasión, la publicación [CRV07], también perteneciente a la categoría de póster, juega el mismo papel que la publicación [CRV06a], aunque su contenido ha sido extendido y recogido en la publicación más reciente [CRV08], en la que por primera vez se presenta un cálculo semántico, lógicamente correcto con respecto a la lógica $CRWL(\mathcal{D})$, para el diagnóstico de respuestas computadas incompletas en un marco teórico tan expresivo como es el esquema $CFLP(\mathcal{D})$. El Capítulo 6 desarrolla el material de estas dos publicaciones presentando en detalle un cálculo lógico que formaliza la recolección de respuestas para un objetivo dado e investigando su relación con cálculos operacionales de resolución de objetivos. El cálculo de recolección de respuestas es lógicamente correcto con respecto a la semántica declarativa del esquema $CFLP(\mathcal{D})$, y sus árboles de derivación dan lugar a los árboles de cómputo que pueden ser empleados para la diagnosis declarativa de respuestas perdidas. La corrección del método de diagnosis se demuestra para cualquier sistema de resolución de objetivos cuyo cómputo de respuestas se pueda formalizar en el cálculo de recolección. El Capítulo 6 también discute la implementación en el sistema $\mathcal{TOY}$ de un prototipo de herramienta que implementa el método de depuración de respuestas perdidas.

- Finalmente, en el Capítulo 7 se resumen y precisan cuáles son los principales resultados alcanzados en este trabajo mediante el nuevo esquema genérico $CFLP(\mathcal{D})$ y su aplicación a la depuración declarativa de programas lógico funcionales con restricciones, tanto a nivel teórico como en cuanto a su aplicación práctica, y se recogen las conclusiones más importantes que se deducen a partir de ellos. También se esbozan sugerencias de trabajo futuro, como son la cooperación de resolutores en el esquema $CFLP(\mathcal{D})$ o la verificación de programas declarativos con restricciones, con el fin de arrojar alguna luz sobre cuestiones que han quedado abiertas y que podrán abordarse en trabajos posteriores en base a los resultados ofrecidos en esta tesis. Parte de este trabajo "futuro" es ya una realidad, que nos permite seguir planteando nuevos retos y metas científicas de interés tanto teórico como práctico en el campo de la programación declarativa con restricciones. En particular, las publicaciones más recientes [EFHR+07a, EFHR+07b, EFHR+08] citadas en este capítulo nos permiten ilustrar una línea de trabajo en curso dentro del grupo de investigación del autor de la tesis, en la cual se aplican los resultados obtenidos para el esquema $CFLP(\mathcal{D})$. Estas publicaciones abordan el problema de la incorporación a $CFLP$ de restricciones heterogéneas pertenecientes a más de un dominio de restricciones y de la cooperación de resolutores pertenecientes a esos dominios, así como de su correspondiente implementación en $\mathcal{TOY}$.

Además de los capítulos anteriores, la presente memoria incluye varios apéndices:

- El Apéndice A incluye la demostración de todas aquellas proposiciones, lemas y teoremas que por su excesiva longitud hemos preferido demostrar por separado. Con ello hemos pretendido facilitar la lectura de los capítulos en los que se presentan estos resultados, ya que algunos de ellos requieren de varios resultados auxiliares y podrían hacer "perder el hilo" al lector. Las demostraciones han sido clasificadas según el capítulo del trabajo en el que se presentan sus correspondientes enunciados. La mayoría de las pruebas no son complicadas y se realizan por inducción, ya sea inducción estructural sobre expresiones o patrones o sobre la profundidad de una prueba en alguno de los cálculos semánticos u operacionales que se presentan. Muchas son, sin embargo, prolijas, sobre todo debido a la necesidad de realizar muchas distinciones de casos. Aunque hemos tratado de ser suficientemente precisos, en ocasiones hemos abreviado la redacción de las demostraciones refiriéndonos a resultados análogos.

- En el Apéndice B presentamos $\mathcal{TOY}(\mathcal{FD})$, una implementación de la instancia particular $CFLP(\mathcal{FD})$ de gran interés práctico que nos permite extender el sistema $\mathcal{TOY}$ para poder tratar restricciones $\mathcal{FD}$. Resaltamos las principales ventajas de este sistema y mostramos algunos resultados sobre su ejecución, así como comparativas sobre su eficiencia con respecto a otros sistemas CLP y $CFLP$ similares. El material básico recogido en este apéndice procede en gran

medida de nuestra publicación de revista [FHSV07], en la que se detalla cómo la implementación de $\mathcal{TOY}(\mathcal{FD})$ se adecúa de una manera efectiva a la instancia $CFLP(\mathcal{FD})$ siguiendo las ideas introducidas en [EV05] sobre árboles definicionales. Finalmente, ofrecemos también una pequeña recopilación de ejemplos de programas en el esquema $CFLP(\mathcal{D})$ escritos en la sintaxis concreta del sistema $\mathcal{TOY}$. Estos ejemplos se clasifican de acuerdo a las instancias básicas de dominios de restricciones que se han utilizado en esta tesis, lo que nos permite ilustrar las facilidades que el sistema $\mathcal{TOY}$ ofrece a la hora de programar con restricciones en cada uno de estos casos particulares de dominios.

- Por último, en el Apéndice C recogemos las referencias de los trabajos publicados en los que se basa el contenido de esta tesis.

Parte I: Un nuevo esquema genérico $CFLP(\mathcal{D})$ para la programación lógico-funcional con restricciones

Capítulo 2

El esquema de programación $CFLP(\mathcal{D})$

En este capítulo presentamos las bases teóricas que constituyen el esquema genérico $CFLP(\mathcal{D})$ de programación lógico funcional con restricciones. Primero se introducen todas aquellas nociones y notaciones preliminares sobre disciplina de tipos y signaturas que resultan esenciales para la formalización matemática de un dominio de restricciones $\mathcal{D}$ y de un resolutor de restricciones sobre un dominio paramétricamente dado. A continuación, se presentan varias instancias de interés práctico del esquema $CFLP(\mathcal{D})$, como son las proporcionadas por el dominio de Herbrand $\mathcal{H}$, con restricciones de igualdad y desigualdad estricta, el dominio de los reales $\mathcal{R}$, y finalmente el dominio finito $\mathcal{FD}$ sobre los números enteros. Para cada uno de estos dominios particulares, se discutirá tanto su formalización teórica como la descripción formal y utilización práctica de resolutores de restricciones apropiados, así como de las propiedades que exhiben cada uno de ellos. Por último, se formaliza la sintaxis de los programas y de los objetivos a resolver en base a los programas en el contexto del nuevo marco de programación proporcionado por el esquema $CFLP(\mathcal{D})$, mostrándose ejemplos concretos de aplicación práctica para cada uno de los lenguajes presentados $CFLP(\mathcal{H})$, $CFLP(\mathcal{R})$ y $CFLP(\mathcal{FD})$.

2.1. Preliminares

En esta primera sección, vamos a comenzar fijando todos aquellos conceptos y notaciones preliminares que van a permitir entender la construcción posterior de dominio de restricciones. Esta presentación sigue en gran parte la realizada en la publicación [LRV07], pero ahora se añade como novedad una consideración explícita de un *sistema de tipos polimórficos* en el estilo de los trabajos de *Hindley-Milner-Damas* [Hin69, Mil78, DM82] siguiendo la línea utilizada en la publicación [FHSV07] para la instancia particular del esquema $CFLP(\mathcal{D})$ sobre el dominio finito $\mathcal{FD}$. Sobre la

base de este sistema de tipos, y sobre el concepto inicial de *signatura*, se fundamentarán las nociones básicas de *expresiones*, *patrones* y *sustituciones*, las cuales constituirán los elementos esenciales de la sintaxis de nuestras restricciones, programas y objetivos en el esquema $CFLP(\mathcal{D})$.

2.1.1. Tipos y signaturas

Comenzamos asumiendo la *signatura universal* constituida por todos los posibles símbolos que es posible utilizar en la sintaxis de cualquier dominio de restricciones y de cualquier programa lógico funcional con restricciones sobre ese dominio: símbolos de constructoras de tipo (incluyendo los símbolos de tipos básicos), símbolos de constructoras de datos, y símbolos de función primitiva y de función definida. Posteriormente, cada dominio de restricciones que se considere de manera particular, especializará dicha signatura universal con el conjunto específico de tipos básicos y de funciones primitivas que mejor se ajusten a las características concretas de las restricciones que se pretenden describir, dando así lugar al concepto de *signatura específica* de un dominio.

Signatura universal y signaturas específicas de un dominio

La *signatura universal* $\Omega = \langle\ TC,\ BT,\ DC,\ DF,\ PF\ \rangle$ está formada por cinco conjuntos mutuamente disjuntos de símbolos, donde:

- $TC = \bigcup_{n\in\mathbb{N}} TC^n$ es una familia de conjuntos numerables y mutuamente disjuntos de símbolos de *constructoras de tipo*, indexados por sus correspondientes aridades.
- BT es un conjunto de símbolos de *tipos básicos*.
- $DC = \bigcup_{n\in\mathbb{N}} DC^n$ es una familia de conjuntos numerables y mutuamente disjuntos de símbolos de *constructoras de datos*, indexados por sus correspondientes aridades.
- $DF = \bigcup_{n\in\mathbb{N}} DF^n$ es una familia de conjuntos numerables y mutuamente disjuntos de símbolos de *función definida*, indexados por sus correspondientes aridades.
- $PF = \bigcup_{n\in\mathbb{N}} PF^n$ es una familia de conjuntos numerables y mutuamente disjuntos de *símbolos de función primitiva*, indexados por sus correspondientes aridades.

La signatura universal representa la idea de que los tipos básicos y los símbolos de función primitiva están asociados a dominios de restricciones específicos, mientras que las contructoras de tipos, las constructoras de datos y los símbolos de función

definida están relacionados con los programas dados por el usuario. Así, para cada elección de una familia específica de tipos básicos $SBT \subseteq BT$ y una familia específica de símbolos de función primitiva $SPF \subseteq PF$, diremos que

$$\Sigma = \langle\ TC,\ SBT,\ DC,\ DF,\ SPF\ \rangle$$

es una *signatura específica* de un dominio. Como puede observarse, cada signatura específica Σ de un dominio hereda todas las constructoras de tipos, las constructoras de datos y los símbolos de función definida a partir de la signatura universal Ω. Esto es debido a que los diferentes programas que se diseñan sobre un mismo dominio dado de restricciones de signatura Σ podrían hacer uso de todos estos símbolos comunes. Por otra parte, todos los símbolos que pertenecen a la familia específica de cada dominio $DC \cup DF \cup SPF$ se denominarán, conjuntamente, *símbolos de función* del dominio.

Tipos

Como ya se ha indicado anteriormente, a lo largo de la presentación del esquema $CFLP(\mathcal{D})$ vamos a hacer uso de una disciplina de tipos estática basada en el sistema clásico de tipos de *Hindley-Milner-Damas* [Hin69, Mil78, DM82]. Un estudio detallado de esta disciplina de tipos polimórficos en el contexto de la programación lógico-funcional (sin restricciones) puede consultarse en [GHR01].

En lo sucesivo, asumiremos disponible un conjunto infinito numerable $\mathcal{TV}ar$ de *variables de tipo*. Sea entonces Σ una signatura específica de un dominio. Un *tipo* $\tau \in Type_\Sigma$ tiene la siguiente sintaxis:

$$\tau ::= A \mid d \mid (c_t\ \tau_1 \ldots \tau_n) \mid (\tau_1, \ldots, \tau_n) \mid (\tau_1 \to \tau_0)$$

donde $A \in \mathcal{TV}ar$, $d \in SBT$ y $c_t \in TC^n$. Por convenio, omitiremos los paréntesis en todos aquellos casos de expresiones de tipos en los que no exista ambigüedad por el contexto en el que se encuentran. Adicionalmente, $c_t\ \overline{\tau}_n$ abrevia $c_t\ \tau_1 \ldots \tau_n$, "$\to$" asocia por la derecha, y $\overline{\tau}_n \to \tau$ abrevia $\tau_1 \to \cdots \to \tau_n \to \tau$. Además, los tipos $c_t\ \overline{\tau}_n$, $(\tau_1, \ldots, \tau_n)$ y $\tau_1 \to \tau_0$ se usarán para representar valores construidos, tuplas y funciones, respectivamente. Finalmente, todos aquellos tipos que no posean apariciones del símbolo "$\to$" se denominarán *tipos de datos*.

Sustituciones de tipos e instancias de tipos

Las *sustituciones de tipos* $\sigma_t, \theta_t \in \mathcal{TS}ub_\Sigma$ se definen como aplicaciones de $\mathcal{TV}ar$ en $Type_\Sigma$, extendidas a aplicaciones de $Type_\Sigma$ en $Type_\Sigma$ en la forma usual. Por convenio, escribiremos $\tau\sigma_t$ en lugar de $\sigma_t(\tau)$ para cualquier tipo τ. Además, siempre que $\tau' = \tau\sigma_t$ para algún σ_t, diremos que τ' es una *instancia* de τ (o también que τ es *más general* que τ') y lo denotaremos por $\tau \preceq \tau'$.

El conjunto de variables de tipo que aparecen en un tipo τ se denota por $tvar(\tau)$. Un tipo τ se denomina *monomórfico* si $tvar(\tau) = \emptyset$ y *polimórfico* en otro caso. Un tipo polimórfico τ se entiende como representante de todas las posibles instancias monomórficas $\tau' \succeq \tau$.

Declaraciones de tipos en signaturas

Para cada símbolo de función perteneciente a una signatura específica Σ cualquiera requeriremos una *declaración de tipo principal*, mediante la cual indicaremos cuál es su tipo más general posible. De manera más precisa:

- Cada símbolo $c \in DC^n$ de aridad $ar(c) = n$ debe tener asociado una declaración de tipo principal de la forma $c :: \overline{\tau}_n \to c_t\, \overline{A}_k$, donde $n, k \geq 0$, $A_1, \ldots, A_k$ son variables de tipo mutuamente diferentes, $c_t \in TC^k$, $\tau_1, \ldots, \tau_n$ son tipos de datos y $\bigcup_{i=1}^{n} tvar(\tau_i) \subseteq \{A_1, \ldots, A_k\}$ (*propiedad de transparencia*).
- Cada símbolo $f \in DF^n$ de aridad $ar(f) = n$ debe tener asociado una declaración de tipo principal de la forma $f :: \overline{\tau}_n \to \tau$, donde τ_i $(1 \leq i \leq n)$ y τ son tipos arbitrarios.
- Cada símbolo $p \in SPF^n$ de aridad $ar(p) = n$ debe tener asociada una declaración de tipo principal de la forma $p :: \overline{\tau}_n \to \tau$, donde $\tau_1, \ldots, \tau_n$ y τ son tipos de datos.

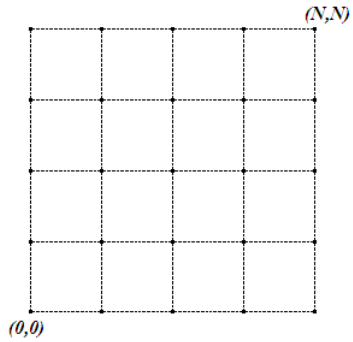
Símbolos especiales

Para tratar aspectos semánticos, asumiremos adicionalmente una constructora de datos especial $(\bot :: A) \in DC^0$, con el propósito de representar un *valor indefinido* que pertenezca a cualquier tipo. También asumiremos que el tipo y las constructoras de datos necesarias para poder trabajar con valores booleanos y con listas se encuentran presentes en la signatura universal Ω. Asumiremos también que SPF^2 incluye el símbolo de función primitiva polimórfica $== :: A \to A \to bool$, que será escrito en notación infija y usado para representar restricciones de *igualdad estricta* en aquellos dominios en los que esta primitiva esté disponible.

En lenguajes concretos de programación lógico funcional con restricciones, como es el caso de $\mathcal{TOY}$ [ALR07] y de *Curry* [Han06], las constructoras de tipos y sus tipos principales se introducen mediante declaraciones de tipos de datos. Así, los tipos principales de las funciones definidas pueden ser, o bien declarados explícitamente o bien inferidos por el compilador, mientras que los tipos principales de las funciones primitivas son predefinidos, y por tanto, conocidos por los usuarios. En cuanto al símbolo $\bot$ no aparece textualmente en la sintaxis de los programas puesto que su uso obedece más bien a motivos puramente semánticos.

Ejemplo 1 (Signaturas y Tipos) *Con el fin de ilustrar las principales nociones relativas a signaturas y tipos que hemos introducido, vamos a considerar las signaturas subyacentes a dos ejemplos concretos de programas con restricciones escritos en el lenguaje de programación lógico funcional $\mathcal{TOY}$. La sintaxis concreta de los programas, tal y como se utiliza en el esquema $CFLP(\mathcal{D})$, será presentada en la Sección 2.6. El primero de los dos programas permite definir una región discreta de puntos en forma de rejilla cuadrada en el plano, utilizando para ello el tipo básico de los enteros y las funciones primitivas específicas del dominio $\mathcal{FD}$ de restricciones finitas que se definirá formalmente en la Subsección 2.2.3.*

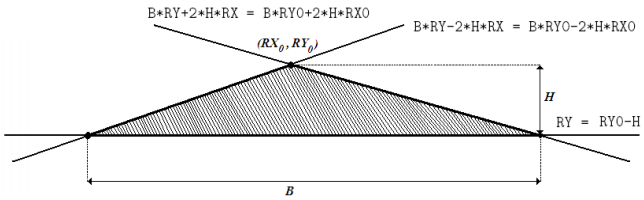
```
% Puntos en el plano:

type dPoint = (int, int)

% Conjuntos y pertenencia:

type setOf A = A -> bool

isIn :: setOf A -> A -> bool
isIn Set Element = Set Element

% Regiones como conjuntos de puntos

type grid = setOf dPoint

% Región de puntos en forma de cuadrado:

square :: int -> grid
square N (X,Y) :- domain [X,Y] 0 N
```



El segundo ejemplo es similar pero hace uso del tipo básico de los reales y de los símbolos de función primitiva del dominio $\mathcal{R}$ de los reales para poder definir regiones triangulares continuas en el plano. El dominio de los reales será formalmente definido en la Subsección 2.2.2.

```
% Puntos en el plano:

type cPoint = (real, real)

% Conjuntos y pertenencia:

type setOf A = A -> bool
```

```
isIn :: setOf A -> A -> bool
isIn Set Element = Set Element

% Regiones como conjuntos de
% puntos

type region = setOf cPoint

% Región triangular:
```




```
triangle :: cPoint -> real -> real -> region

triangle (RX0,RY0)  B H (RX,RY) :-
                        RY >= RY0 - H,
                        B * RY - 2 * H * RX <= B * RY0 - 2 * H * RX0,
                        B * RY + 2 * H * RX <= B * RY0 + 2 * H * RX0
```

En estos dos ejemplos encontramos:

- *Dos tipos básicos,* `int` *y* `real`*, para representar valores numéricos enteros y reales, respectivamente.*
- *Un símbolo de constructora de aridad cero de tipo* `bool` *para el tipo de los valores booleanos, y un símbolo de constructora de tipo de aridad uno* `list` *para el tipo de las listas polimórficas. La sintaxis concreta que utilizaremos para representar el tipo polimórfico* `list A` *de las listas de elementos de tipo* `A` *será la habitual en lenguajes de programación funcional* `[A]`*.*
- `[A]` *es un tipo de datos, puesto que no tiene apariciones de la constructora de tipos* `->`*. Más aún, es polimórfico, puesto que incluye una variable de tipo. Entre las instancias de* `[A]` *podemos encontrar* `[int]` *(para representar listas de enteros) y* `[int -> int]` *(para representar listas de funciones de tipo* `int -> int`*). Se observa así que algunas instancias de un tipo de datos pueden no ser a su vez tipos de datos.*
- *Dos constructoras de datos de aridad cero,* `false,true::bool` *(para representar valores booleanos); una constructora de datos de aridad cero* `nil::[A]` *(para representar la lista vacía); y una constructora de datos binaria* `cons::A -> [A] -> [A]` *(para representar listas no vacías). La sintaxis concreta para* `nil` *(respectivamente,* `cons`*) es* `[]` *(respectivamente,* `:`*), donde* `:` *será utilizado como un operador infijo.*

- *Los tipos principales de las constructoras en los apartados anteriores pueden ser derivados a partir de las declaraciones de tipos de datos*

```
data bool = false | true
data [A] = [] | (A : [A])
```

los cuales son tipos de datos predefinidos y no necesitan ser incluidos dentro de los programas.

- *En el ejemplo de aplicación hay también declaraciones de* tipos sinónimos*, como por ejemplo*

```
type dPoint = (int,int)
type setOf A = A -> bool
type region = setOf dPoint
```

Tales declaraciones son de utilidad práctica para designar ciertos tipos con las siguientes restricciones: no pueden involucrar recursión y los nombres de tipos sinónimos que sean introducidos de este modo no se considerarán como pertenecientes a la signatura.

- *Símbolos de función definida de distintas aridades, como por ejemplo,* `isIn`*,* `square` $\in DF^2$*. Estos dos símbolos de función son binarios porque las reglas de reescritura dadas para ellos dentro del programa esperan dos parámetros formales en sus lados izquierdos. En general, las reglas de reescritura incluidas en los programas para definir el comportamiento de los símbolos de función* $f \in DF^n$ *se espera que tengan n parámetros formales en sus lados izquierdos. En algunos casos, este n podría no corresponder idénticamente al número de flechas observadas en el tipo principal de la función f. Por ejemplo, aunque* `square` $\in DF^2$*, el tipo principal de esta función es* `square :: int -> grid`*. La aparente contradicción desaparece sin embargo cuando se observa que* `grid` *está declarado como un tipo sinónimo para* `(int,int) -> bool`*. Puesto que la constructora de tipo* `->` *asocia por la derecha, tenemos que* `square :: int -> (int,int) -> bool`*.*

- *Símbolos de función primitiva de varias aridades, como por ejemplo, las primitivas binarias* `*` *y* `<=`*, y la primitiva ternaria* `domain`*. La sintaxis concreta requiere que* `*` *y* `<=` *sean usadas en notación infija. Cada primitiva tiene un tipo principal predefinido. Por ejemplo:* `* :: real -> real -> real` *y* `domain :: [int] -> int -> int -> bool`*. Estas declaraciones no necesitan incluirse dentro de los programas.*

2.1.2. Expresiones y patrones

A partir de una signatura específica Σ se definen los dos tipos básicos de construcciones sintácticas de los que haremos uso en adelante: *expresiones* y *patrones*. Mediante las primeras, representaremos valores de datos que aún pueden ser reducidos para su evaluación mediante la aplicación específica de reglas de programa correspondientes a funciones definidas, mientras que con los segundos, representaremos valores de datos ya evaluados. En ambos casos, estaremos interesados en utilizar sólo aquellas expresiones y patrones para los que se pueda demostrar que poseen un tipo adecuado en el sistema de tipos definido en la subsección anterior.

Valores básicos

Para cualquier dominio de signatura específica Σ, usaremos expresiones que pueden tener apariciones de ciertos valores de un tipo básico. Por tanto, con el fin de definir la sintaxis de las expresiones, asumiremos en primer lugar una SBT-familia indexada $\mathcal{B} = \{\mathcal{B}_d\}_{d \in SBT}$, donde cada $\mathcal{B}_d$ es un conjunto no vacío cuyos elementos se entenderán como *valores básicos* de tipo d. En lo sucesivo, usaremos las letras $u, v, \ldots$ para indicar valores básicos. Por comodidad, también escribiremos $u \in \mathcal{B}$ en lugar de $u \in \bigcup_{d \in SBT} \mathcal{B}_d$.

Expresiones aplicativas

Asumamos un conjunto infinito numerable $\mathcal{Var}$ de *variables de datos* (disjunto del conjunto de variables de tipo $\mathcal{TVar}$ y de los demás símbolos en la signatura específica Σ). Definimos Σ-*expresiones aplicativas* $e \in Exp_\Sigma(\mathcal{B})$ sobre $\mathcal{B}$ mediante la utilización de las siguientes reglas sintácticas:

$$e ::= X \mid u \mid h \mid (e_1, \ldots, e_n) \mid (e\, e_1)$$

donde $X \in \mathcal{Var}$, $u \in \mathcal{B}$ y $h \in DC \cup DF \cup SPF$. Las expresiones de la forma $(e_1, \ldots, e_n)$ representan *n-tuplas* ordenadas, mientras que expresiones de la forma $(e\, e_1)$ (no confundir con pares ordenados (e, e_1)) denotan la *aplicación* de la función representada por e al argumento representado por e_1. Siguiendo una convención usual para los lenguajes de programación funcional, asumimos que las aplicaciones asocian por la izquierda, y usamos la notación $(e\, \overline{e}_n)$ para abreviar $(e\, e_1 \ldots e_n)$. De forma más general, los paréntesis pueden omitirse en caso de que no exista ambigüedad en el contexto en el que se encuentran. La sintaxis 'aplicativa' es común en lenguajes funcionales de orden superior, mientras que la sintaxis usual de primer orden para expresiones puede ser traducida fácilmente a sintaxis aplicativa por medio de la denominada *notación currificada*. Por ejemplo, una expresión de la forma $f(X, g(Y))$ se podría representar como $(f\, X\, (g\, Y))$.

Patrones

Las expresiones que no poseen apariciones de variables repetidas se denominan expresiones *lineales*, las expresiones sin apariciones de variables se denominan *cerradas* y las expresiones sin ninguna aparición del símbolo $\bot$ se denominan *totales*. Como ya se ha indicado, algunas expresiones particulares pueden utilizarse para representar valores de datos que no necesitan ser evaluados. Tales expresiones reciben el nombre de Σ-*patrones* $t \in Pat_\Sigma(\mathcal{B})$ sobre $\mathcal{B}$ y tienen la siguiente sintaxis:

$$t ::= X \mid u \mid (t_1, \ldots, t_n) \mid c\,\overline{t}_m \mid f\,\overline{t}_m \mid p\,\overline{t}_m$$

donde $X \in \mathcal{V}ar$, $u \in \mathcal{B}$, $c \in DC^n$ para algún $n \geq m$, $f \in DF^n$ para algún $n > m$, y $p \in SPF^n$ para algún $n > m$. Las restricciones que involucran aridades en los últimos tres casos están motivadas por la idea de que una expresión de la forma $h\,\overline{t}_m$ (donde $h \in DF^n \cup SPF^n$ y $n \leq m$) es potencialmente evaluable, y por tanto, no debería ser vista como representante de un valor.

El conjunto de todos los patrones básicos sobre $\mathcal{B}$ se denotará como $GPat_\Sigma(\mathcal{B})$. Algunas veces, escribiremos $\mathcal{U}_\Sigma(\mathcal{B})$ en lugar de $GPat_\Sigma(\mathcal{B})$, entendiendo este conjunto como el *universo de valores* sobre $\mathcal{B}$. La siguiente clasificación de expresiones es también útil:

- $X\,\overline{e}_m$ (con $X \in \mathcal{V}ar$ y $m \geq 0$) se denomina expresión *flexible*.
- $u \in \mathcal{B}$, y todas las expresiones de la forma $h\ \overline{e}_m$ (con $h \in DC \cup DF \cup SPF$), se denominan expresiones *rígidas*.
- Una expresión rígida $h\,\overline{e}_m$ se denomina *activa* si y sólo si $h \in DF^n \cup SPF^n$ para algún $n \leq m$ y *pasiva* en otro caso.

La sintaxis de tupla $(e_1, \ldots, e_n)$ se considera también un caso particular de expresiones pasivas. La idea es que cualquier expresión pasiva tiene la apariencia externa de un patrón, aunque podría no tratarse de un patrón en el caso de que alguna subexpresión más interna sea a su vez activa. Como puede verse en nuestro ejemplo inicial, las tuplas son útiles para programación, y por tanto, la sintaxis de tuplas es soportada por múltiples lenguajes de programación, incluyendo $\mathcal{TOY}$. Por otro lado, las tuplas pueden ser tratadas como un caso particular de valores construidos, asumiendo constructoras de datos $tup_n \in DC^n$ en la signatura universal, y viendo cualquier tupla $(e_1, \ldots, e_n)$ como 'azúcar sintáctico' para $tup_n\ e_1\ \ldots\ e_n$. Por esta razón, en lo que sigue, omitiremos la mención explícita de las tuplas, aunque continuaremos haciendo uso de ellas en otros ejemplos.

Expresiones bien tipadas

Como es usual en los lenguajes de programación que adoptan una disciplina de tipos estática, todas las expresiones que aparecen en los programas se espera que estén

bien tipadas. La derivación o comprobación de los tipos de las expresiones recae fundamentalmente sobre dos tipos de información:

(1) Los tipos principales de los símbolos que pertenecen a la signatura, que asumimos están asociados a la signatura en sí misma.

(2) Los tipos de las variables que aparecen en las expresiones.

Con el fin de representar este segundo tipo de información, usaremos *contextos de tipos* $\Gamma = \{X_1 :: \tau_1, \ldots, X_n :: \tau_n\}$, mediante los cuales expresaremos la suposición de que cada variable X_i tiene el tipo asociado τ_i, para todo $1 \leq i \leq n$.

Siguiendo las ideas bien conocidas de los trabajos de Hindley, Milner y Damas [Hin69, Mil78, DM82], es posible definir reglas de inferencia de tipos para poder derivar *juicios de tipos* de la forma $\Sigma, \Gamma \vdash_{WT} e :: \tau$, mediante los cuales aseguramos que la aserción $e :: \tau$ (en palabras, "la expresión e tiene tipo τ") puede ser deducida a partir de las suposiciones de tipos para símbolos (respectivamente para variables), dadas en Σ (respectivamente en Γ).

Una expresión e se denomina *bien tipada* si y sólo si es posible encontrar un contexto de tipos Γ tal que $\Sigma, \Gamma \vdash_{WT} e :: \tau$ pueda ser derivado para al menos un cierto tipo τ. Aunque este tipo τ no es único en general, puede demostrarse que para cualquier expresión bien tipada e existe un tipo τ tal que los tipos derivables para e son precisamente las instancias τ' de τ. A τ se le llama *tipo principal* de e, y es único salvo renombramiento de variables de tipo. En la práctica, los tipos principales de expresiones bien tipadas pueden ser inferidos automáticamente como una tarea más del proceso de compilación.

Escribiremos $\Sigma, \Gamma \vdash_{WT} \overline{e}_n :: \overline{\tau}_n$ para indicar que $\Sigma, \Gamma \vdash_{WT} e_i :: \tau_i$ puede ser derivado para todo $1 \leq i \leq n$. En el caso particular de tener una expresión cerrada e, es suficiente con considerar el contexto de tipos vacío $\Gamma = \emptyset$. En este caso, la notación $\Sigma, \Gamma \vdash_{WT} e :: \tau$ puede simplificarse en la forma abreviada $\Sigma \vdash_{WT} e :: \tau$. Más aún, en ejemplos concretos escribiremos simplemente $e :: \tau$ para indicar que $\Sigma, \Gamma \vdash_{WT} e :: \tau$ puede ser derivado usando la signatura subyacente Σ y algún contexto de tipos adecuado Γ.

Reglas de inferencia de tipos monomórficos

A continuación mostramos las reglas de inferencia de tipos monomórficos que pueden ser usadas para derivar juicios de tipos de la forma $\Sigma \vdash_{WT} e :: \tau$ cuando e es una expresión cerrada y τ es una instancia monomórfica del tipo principal de e.

$$\frac{}{\Sigma \vdash_{WT} \bot :: \tau} \qquad \frac{}{\Sigma \vdash_{WT} u :: d} \qquad \frac{}{\Sigma \vdash_{WT} h :: \tau'}$$

$$\frac{\Sigma \vdash_{WT} e_i :: \tau_i \quad (1 \leq i \leq n)}{\Sigma \vdash_{WT} (e_1,\ldots,e_n) :: (\tau_1,\ldots,\tau_n)} \qquad \frac{\Sigma \vdash_{WT} e :: \tau_1 \to \tau \qquad \Sigma \vdash_{WT} e_1 :: \tau_1}{\Sigma \vdash_{WT} (e\ e_1) :: \tau}$$

donde $u \in \mathcal{B}_d$, $d \in SBT$, $(h :: \tau) \in DC \cup DF \cup SPF$ y τ' es una instancia monomórfica de τ. Observamos que para una expresión fija e podría haber ninguno, uno o varios tipos monomórficos τ tales que $\Sigma \vdash_{WT} e :: \tau$.

El lector interesado puede remitirse a [GHR01] para una presentación más extensa de las reglas de inferencia de tipos adaptadas a los lenguajes lógico funcionales sin restricciones. Añadir el tratamiento de restricciones será un aspecto relativamente inmediato como veremos más adelante.

El orden de información $\sqsubseteq$

Con el fin de poder abordar cuestiones semánticas sobre el esquema $CFLP(\mathcal{D})$ en los próximos capítulos, resulta de utilidad definir un *orden de información* $\sqsubseteq$ sobre $Exp_\Sigma(\mathcal{B})$, de forma que $e \sqsubseteq e'$ signifique que la información proporcionada por e' es mayor o igual que la información proporcionada por e. Matemáticamente, la relación $\sqsubseteq$ se define como el menor orden parcial sobre $Exp_\Sigma(\mathcal{B})$ tal que $\bot \sqsubseteq e'$ para todo e' $\in Exp_\Sigma(\mathcal{B})$ y $(e\, e_1) \sqsubseteq (e'\, e'_1)$ siempre que $e \sqsubseteq e'$ y $e_1 \sqsubseteq e'_1$.

Asimismo, aceptaremos sin demostración el siguiente lema. Este resultado es similar al denominado "*Typing Monotonicity Lemma*" utilizado en [GHR01], y nos permite asegurar que el tipo de cualquier expresión es también válido para todas sus correspondientes aproximaciones semánticas. Este lema puede ser demostrado gracias al hecho de que el valor indefinido $\bot$ pertenece a todos los tipos.

Lema 1 (Lema de Preservación de Tipos) *Asumamos que* $\Sigma, \Gamma \vdash_{WT} e' :: \tau$ *y que* $e \sqsubseteq e'$ *se satisface. Entonces* $\Sigma, \Gamma \vdash_{WT} e :: \tau$ *es también cierto.*

Patrones transparentes y opacos

Como parte de la definición de cualquier signatura Σ hemos requerido que se satisfaga una propiedad especial denominada *propiedad de transparencia* para los tipos principales de las constructoras de datos. Debido a esta propiedad, los tipos de las variables que aparecen en un término de datos t pueden ser siempre deducidos a partir del tipo de t. Resulta de gran utilidad poder aislar todos aquellos patrones que tienen una propiedad similar. Con el fin de conseguir este propósito, necesitamos adaptar algunas de las definiciones introducidas en [GHR01] a nuestro contexto actual. En concreto, diremos que un tipo que puede ser escrito en la forma $\overline{\tau}_m \to \tau$ es *m-transparente* si y sólo si $tvar(\overline{\tau}_m) \subseteq tvar(\tau)$, y diremos que es *m-opaco* en otro caso. Adicionalmente, los símbolos de función definida f y los símbolos de función primitiva p se denominan m-transparentes si y sólo si sus tipos principales son m-transparentes, y se dice que son m-opacos en otro caso. Se observa así que una constructora de datos c es siempre m-transparente para todo $m \leq ar(c)$.

Siguiendo esta idea, consideraremos en lo que sigue como *patrones transparentes* a todos aquellos patrones que queden definidos a través de las siguientes reglas sintácticas:

$$t ::= X \mid u \mid (t_1, \ldots, t_n) \mid c\,\overline{t}_m \mid f\,\overline{t}_m \mid p\,\overline{t}_m$$

donde $X \in \mathcal{Var}$, $u \in \mathcal{B}$, $c \in DC^n$ para algún $m \leq n$, $f \in DF^n$ para algún $n > m$, y $p \in SPF^n$ para algún $n > m$, donde los subpatrones t_i en $(t_1, \ldots, t_n)$, $(c\,\overline{t}_m)$, $(f\ \overline{t}_m)$ y $(p\,\overline{t}_m)$ han de ser recursivamente transparentes, y los tipos principales tanto del símbolo de función definida f en $(f\ \overline{t}_m)$ como del símbolo de función primitiva p en $(p\,\overline{t}_m)$ también han de ser m-transparentes.

Como ejemplo concreto, asumamos un símbolo de función definida *snd* con declaración de tipo principal $snd :: A \to B \to B$. Entonces *snd* es 1-opaco, y el patrón $(snd\ X)$ también es opaco. De hecho, el tipo principal $B \to B$ de $(snd\ X)$ no revela información sobre el tipo de X, por lo que las diferentes instancias de $(snd\ X)$ siempre tienen el tipo principal $B \to B$, con independencia del tipo de la expresión que ha sido sustituida en el lugar de X. Semejante comportamiento no es posible si se utilizan patrones transparentes, debido al siguiente *Lema de Transparencia* que se enuncia a continuación. De nuevo, se omite su demostración, ya que resultados análogos han sido demostrados en [GHR01] en un contexto similar al que aquí se ha presentado.

Lema 2 (Lema de Transparencia) *Se cumplen los dos resultados siguientes:*

(1) Asumamos un patrón transparente t y dos contextos de tipos Γ_1 y Γ_2 tales que $\Sigma, \Gamma_1 \vdash_{WT} t :: \tau$ y $\Sigma, \Gamma_2 \vdash_{WT} t :: \tau$ para un tipo común τ. Entonces, $\Gamma_1(X) = \Gamma_2(X)$ se cumple para cualquier $X \in var(t)$.

(2) Asumamos que $\Sigma, \Gamma \vdash_{WT} h\ \overline{a}_m :: \tau :: h\ \overline{b}_m$ se cumple para algún $h \in DC \cup DF \cup PF$ que sea m-transparente y algún tipo común τ. Entonces, existen tipos τ_i tales que $\Sigma, \Gamma \vdash_{WT} a_i :: \tau_i :: b_i$ se cumple para cualquier $1 \leq i \leq m$.

Ejemplo 2 *Consideramos las signaturas específicas Σ y las familias de valores básicos $\mathcal{B}$ utilizadas en los programas del Ejemplo 1. Para este ejemplo, se tiene que:*

- *El conjunto de valores básicos es $\mathcal{B}_{int} = \mathbb{Z}$ y $\mathcal{B}_{real} = \mathbb{R}$.*
- *Expresiones bien tipadas, como por ejemplo $square\ 4\ (2,3) :: bool$, $RX - RY :: real$ y $(RY - RX \leq RY_0 - RX_0) :: bool$.*
- *Patrones bien tipados, como por ejemplo $3 :: int$, $3.01 :: real$, $[X, Y] :: [int]$ y $square\ 4 :: dPoint \to bool$. Observamos que $[X, Y]$ permite abreviar la expresión $(X : (Y : [\,]))$, como es usual en los lenguajes funcionales que usan una constructora de listas infija.*

- *Finalmente, es posible observar que* $\bot \sqsubseteq (0 : \bot) \sqsubseteq (0 : (1 : \bot)) \sqsubseteq \ldots$ *ilustra el comportamiento del orden de información* $\sqsubseteq$ *cuando se restringe a la comparación de patrones pertenecientes al universo* $\mathcal{U}_\Sigma(\mathcal{B})$*. Los patrones de listas de tipo* `[int]` *que se utilizan en este ejemplo no se permite que aparezcan en la sintaxis concreta de los programas, debido a las apariciones del valor indefinido* $\bot$*, aunque son de utilidad como representaciones semánticas de listas parcialmente computadas de enteros.*

2.1.3. Sustituciones

De la misma manera que hemos definido sustituciones de tipos, en lo que sigue haremos uso de *sustituciones* $\sigma, \theta \in Sub_\Sigma(\mathcal{B})$ sobre $\mathcal{B}$ definidas como aplicaciones de $\mathcal{Var}$ en $Pat_\Sigma(\mathcal{B})$, las cuales pueden ser a su vez extendidas a aplicaciones de $Exp_\Sigma(\mathcal{B})$ en $Exp_\Sigma(\mathcal{B})$ del modo habitual.

Para cualquier expresión $e \in Exp_\Sigma(\mathcal{B})$ y para cualquier sustitución $\sigma \in Sub_\Sigma(\mathcal{B})$, escribiremos habitualmente $e\sigma$ en lugar de $\sigma(e)$. Además, siempre que $e' = e\sigma$ para alguna sustitución σ, diremos que e' es una *instancia* de e (o también que e es *más general* que e') y lo representaremos mediante la notación $e \preceq e'$.

Asimismo, escribiremos ε para identificar la *sustitución identidad*, y $\sigma\theta$ para la composición de σ y θ, de tal modo que $e(\sigma\theta) = (e\sigma)\theta$ para cualquier expresión e. Una sustitución σ tal que $\sigma\sigma = \sigma$ se denomina *idempotente*. El *dominio* $vdom(\sigma)$ y el *rango de variables* $vran(\sigma)$ de una sustitución se definen del modo usual:

$$vdom(\sigma) = \{X \in \mathcal{Var} \mid X\sigma \neq X\} \qquad vran(\sigma) = \bigcup_{X \in\, vdom(\sigma)} var(X\sigma)$$

Una sustitución σ se denomina *finita* si y sólo si $vdom(\sigma)$ es un conjunto finito, y se dice que es *cerrada* si y sólo si $X\sigma$ es un patrón cerrado para todo $X \in vdom(\sigma)$. En lo sucesivo, asumiremos que las sustituciones con las que trabajamos son siempre finitas, a menos que se especifique lo contrario. Así, adoptaremos la notación usual $\sigma = \{X_1 \mapsto t_1, \ldots, X_n \mapsto t_n\}$ siempre que $vdom(\sigma) = \{X_1, \ldots, X_n\}$ y $X_i\sigma = t_i$ para todo $1 \leq i \leq n$. En particular, $\varepsilon = \{\,\} = \emptyset$. También escribiremos $\sigma[X \mapsto t]$ para la sustitución σ' tal que $X\sigma' = t$ y $Y\sigma' = Y\sigma$ para cualquier variable $Y \in \mathcal{Var} \setminus \{X\}$.

Para cualquier conjunto de variables $\mathcal{X} \subseteq \mathcal{Var}$ definimos la *restricción* $\sigma\restriction_{\mathcal{X}}$ como la sustitución σ' tal que $vdom(\sigma') = \mathcal{X}$ y $\sigma'(X) = \sigma(X)$ para todo $X \in \mathcal{X}$. Usamos la notación $\sigma =_{\mathcal{X}} \theta$ para indicar que $\sigma\restriction_{\mathcal{X}} = \theta\restriction_{\mathcal{X}}$, y abreviamos $\sigma =_{\mathcal{Var}\setminus\mathcal{X}} \theta$ como $\sigma =_{\setminus\mathcal{X}} \theta$. Finalmente, consideramos dos modos diferentes de comparar dos sustituciones dadas σ, $\sigma' \in Sub_\Sigma(\mathcal{B})$:

- σ se dice que es *más general* que σ' sobre $\mathcal{X} \subseteq \mathcal{Var}$ (en símbolos, $\sigma \preceq_{\mathcal{X}} \sigma'$) si y sólo si $\sigma\theta =_{\mathcal{X}} \sigma'$ para algún $\theta \in Sub_\Sigma(\mathcal{B})$. Abreviamos $\sigma \preceq_{\mathcal{Var}} \sigma'$ como $\sigma \preceq \sigma'$ y $\sigma \preceq_{\mathcal{Var}\setminus\mathcal{X}} \sigma'$ como $\sigma \preceq_{\setminus\mathcal{X}} \sigma'$.

- σ se dice que *porta menos información* que σ' sobre $\mathcal{X} \subseteq \mathcal{Var}$ (en símbolos, $\sigma \sqsubseteq_{\mathcal{X}} \sigma'$) si y sólo si $\sigma(X) \sqsubseteq \sigma'(X)$ para todo $X \in \mathcal{X}$. Abreviamos $\sigma \sqsubseteq_{\mathcal{Var}} \sigma'$ como $\sigma \sqsubseteq \sigma'$ y $\sigma \sqsubseteq_{\mathcal{Var} \setminus \mathcal{X}} \sigma'$ como $\sigma \sqsubseteq_{\setminus \mathcal{X}} \sigma'$.

2.2. Dominios de restricciones

Intuitivamente, un *dominio de restricciones* ha de ser capaz de proporcionar valores de datos y restricciones concretas, con el fin de que ambos puedan ser utilizados en aplicaciones particulares de ese dominio. Por esta razón, todo dominio de restricciones debe proporcionar una familia de valores básicos de tipos conocidos, así como una familia de funciones primitivas, dadas estas como interpretaciones de los símbolos de función primitiva que aparecen en su signatura específica asociada.

La obtención de una formalización matemática de la noción de dominio de restricciones ha pasado en las últimas décadas por diferentes enfoques y propuestas, aunque muchas de ellas usan nociones matemáticas conocidas que han sido tomadas del *Álgebra*, la *Lógica Matemática* o la *Teoría de Categorías* (el lector interesado puede remitirse a [JL87, Sar92, Lop92, Lop94, JM94, JMMS98] como referencias de algunas de las propuestas más relevantes en este sentido). En esta sección, vamos a establecer la definición de dominio de restricciones que va a ser utilizada en lo sucesivo en todo este trabajo, como una versión revisada y mejorada de la que aparece en la publicación [LRV07] mediante la introducción explícita de tipos:

Definición 1 (Dominio de Restricciones) *Un* dominio de restricciones $\mathcal{D}$ *de signatura específica* Σ *(abreviadamente* Σ-dominio*, o simplemente* dominio *si queda claro por el contexto quién es* Σ*) es una estructura de la forma:*

$$\mathcal{D} = \langle\ \mathcal{B}^{\mathcal{D}},\ \{p^{\mathcal{D}}\}_{p\,\in\,SPF}\ \rangle$$

donde $\mathcal{B}^{\mathcal{D}} = \{\mathcal{B}^{\mathcal{D}}_d\}_{d\,\in\,SBT}$ *es una SBT-familia indexada de conjuntos de valores básicos, y donde la* interpretación $p^{\mathcal{D}}$ *de cada símbolo de función primitiva* $p :: \overline{\tau}_n \to \tau$ *en* SPF^n *se requiere que sea un conjunto de* $(n+1)$*-tuplas* $p^{\mathcal{D}} \subseteq \mathcal{U}_\Sigma(\mathcal{B}^{\mathcal{D}})^{n+1}$*. En lo sucesivo, abreviamos* $\mathcal{U}_\Sigma(\mathcal{B}^{\mathcal{D}})$ *como* $\mathcal{U}_{\mathcal{D}}$ *y escribiremos* $p^{\mathcal{D}}\overline{t}_n \to t$ *para* $(\overline{t}_n, t) \in p^{\mathcal{D}}$*. El significado pretendido de* $p^{\mathcal{D}}\overline{t}_n \to t$ *es el de que la función primitiva* $p^{\mathcal{D}}$*, con argumentos dados* $\overline{t}_n$*, pueda devolver un resultado* t*. Más aún, cada interpretación de símbolos primitivos se requiere que satisfaga las siguientes cuatro condiciones:*

(1) Polaridad*: Para todo* $p \in SPF$*,* $p^{\mathcal{D}}\overline{t}_n \to t$ *se comporta monótonamente con respecto a los argumentos* $\overline{t}_n$*, y antimonótonamente con respecto al resultado* t*. Formalmente: para todos* $\overline{t}_n, \overline{t'}_n, t, t' \in \mathcal{U}_{\mathcal{D}}$ *tales que* $p^{\mathcal{D}}\overline{t}_n \to t$*,* $\overline{t}_n \sqsubseteq \overline{t'}_n$ *y* $t \sqsupseteq t'$*,* $p^{\mathcal{D}}\overline{t'}_n \to t'$ *también se cumple.*

(2) Radicalidad*: Para todo* $p \in SPF$*, tan pronto como los argumentos dados a* $p^{\mathcal{D}}$ *tengan información suficiente para devolver un resultado distinto de* $\bot$*,*

esos mismos argumentos son ya suficientes para devolver un resultado total. Formalmente: para todos $\bar{t}_n, t \in \mathcal{U}_\mathcal{D}$*, si* $p^\mathcal{D}\bar{t}_n \to t$ *entonces* $t = \bot$*, o si no, existe un patrón total* $t' \in \mathcal{U}_\mathcal{D}$ *tal que* $p^\mathcal{D}\bar{t}_n \to t'$ *y* $t' \sqsupseteq t$*.*

(3) Bien tipado*: Para todo* $p \in SPF$*, el comportamiento de* $p^\mathcal{D}$ *está bien tipado con respecto a cualquier instancia monomórfica del tipo principal de* p*. Formalmente: para cualquier instancia de tipo monomórfica* $(\overline{\tau'}_n \to \tau') \succeq (\overline{\tau}_n \to \tau)$*, y para todos* $\bar{t}_n, t \in \mathcal{U}_\mathcal{D}$ *tales que* $\Sigma \vdash_{WT} \bar{t}_n :: \overline{\tau'}_n$ *y* $p^\mathcal{D}\bar{t}_n \to t$*, la sentencia de tipos* $\Sigma \vdash_{WT} t :: \tau'$ *también se cumple.*

(4) Igualdad estricta*: La primitiva* $==$ *(en caso de que ésta pertenezca a* SPF*) se interpreta como* igualdad estricta *sobre* $\mathcal{U}_\mathcal{D}$*, de modo que para todos* $t_1, t_2, t \in \mathcal{U}_\mathcal{D}$*, se tiene que* $t_1 ==^\mathcal{D} t_2 \to t$ *si y sólo si alguna de las tres siguientes condiciones se cumple:*

(a) t_1 *y* t_2 *son el mismo patrón total y* $true \sqsupseteq t$*.*

(b) t_1 *y* t_2 *no tienen una cota superior común en* $\mathcal{U}_\mathcal{D}$ *con respecto al orden de información* $\sqsubseteq$ *y* $false \sqsupseteq t$*.*

(c) $t = \bot$*.*

Con esta definición, es fácil comprobar que $==^\mathcal{D}$ *satisface las condiciones de polaridad, de radicalidad y de bien tipado.*

Para todo dominio dado $\mathcal{D}$ de signatura Σ, el conjunto $\mathcal{U}_\mathcal{D} = \mathcal{U}_\Sigma(\mathcal{B}^\mathcal{D}) = GPat_\Sigma(\mathcal{B}^\mathcal{D})$ se denomina el *universo de valores* del dominio $\mathcal{D}$. Escribiremos en lo sucesivo también $Exp_\mathcal{D}$ y $Pat_\mathcal{D}$, en lugar de $Exp_\Sigma(\mathcal{B}^\mathcal{D})$ y $Pat_\Sigma(\mathcal{B}^\mathcal{D})$, respectivamente, para facilitar el empleo de estas notaciones. Incluso el subíndice $\mathcal{D}$ será omitido en algunas ocasiones en las éste pueda ser deducido a partir del contexto.

Ofrecemos a continuación unas breves explicaciones sobre las condiciones pedidas a las interpretaciones de símbolos primitivos en la definición anterior.

- La condición de *polaridad* dada en la definición se utiliza para asegurar que la interpretación $p^\mathcal{D}$ codifica el comportamiento de una aplicación continua y monótona de $\overline{\mathcal{U}}^n_\mathcal{D}$, la n-ésima potencia del universo de valores obtenido a partir de $\mathcal{U}_\mathcal{D}$ por *complección mediante ideales* [Mol85] en el *dominio potencia de Hoare* $\mathcal{HP}(\overline{\mathcal{U}_\mathcal{D}})$ [Sco82, Win85, GS90]. Intuitivamente, se puede pensar en $p^\mathcal{D}$ como la descripción del comportamiento de una función posiblemente indeterminista sobre elementos de información finitos. El tipo de indeterminismo involucrado aquí está basado en trabajos previos desarrollados en el marco de la *Lógica para la Reescritura CRWL* para programación lógico funcional [GHLR99, GHR01, AR01], los cuales a su vez están inspirados en las ideas de los trabajos previos [Hus88, Hus92, Hus93].

- La condición de *radicalidad* es más novedosa e importante para nuestros propósitos formales, pues permite que las llamadas a funciones primitivas necesarias en el contexto de un cierto cómputo se puedan ejecutar de manera impaciente sin pérdida de completitud. Hasta el momento, todas las funciones primitivas que conocemos y que son usadas en dominios de restricciones de utilidad práctica son radicales en este sentido, como se verá en posteriores secciones de este capítulo.
- El último requisito de la Definición 1 impone una interpretación fijada de la primitiva $==$ como la operación de igualdad estricta $==^{\mathcal{D}}$ sobre $\mathcal{U}_{\mathcal{D}}$, para cualquier dominio $\mathcal{D}$ cuya signatura específica incluya esta primitiva.

Ilustramos a continuación la definición de dominio de restricciones mediante ejemplos concretos que serán utilizados tanto a lo largo de las restantes secciones de este capítulo como de los demás capítulos de este trabajo. Se trata del dominio de Herbrand $\mathcal{H}$, el dominio finito $\mathcal{FD}$ y el dominio de los reales $\mathcal{R}$.

2.2.1. El dominio de Herbrand $\mathcal{H}$

El *dominio de Herbrand* permite representar cómputos con restricciones de igualdad y desigualdad simbólica sobre valores de cualquier tipo. Formalmente, este dominio de restricciones queda definido como sigue:

- Su signatura específica es $\Sigma = \langle\ TC,\ SBT,\ DC,\ DF,\ SPF\ \rangle$, donde SBT es un conjunto vacío de tipos básicos y SPF incluye exclusivamente el operador de igualdad estricta $== :: A \to A \to bool$.
- La interpretación $==^{\mathcal{H}}$ se define del mismo modo que para cualquier otro dominio de restricciones cuya signatura específica incluya la primitiva $==$. Esta primitiva tiene, por tanto, sentido para cualquier dominio de restricciones que haya sido construido sobre un conjunto cualquiera de elementos primitivos.

En lo sucesivo, escribiremos $\mathcal{H}$ para denotar el dominio de restricciones construido sobre el conjunto vacío de elementos primitivos que tiene $==$ como su única primitiva. Así, el lenguaje $CFLP(\mathcal{H})$ puede verse como una nueva fundamentación de los trabajos anteriores sobre programación lógico funcional con restricciones de desigualdad [KLMR92, AGL94, LS99a]. Por otra parte, $\mathcal{H}$ también muestra una analogía clara con la extensión del dominio de Herbrand con restricciones de desigualdad, introducido por A. Colmerauer en [Col82, Col84] como una de las primeras extensiones de la programación lógica, y más tarde investigada por M. J. Maher en [Mah88]. Algunas diferencias importantes, sin embargo, deben quedar claras en este momento:

- En primer lugar, el dominio base de $\mathcal{H}$ en nuestra formalización es un conjunto parcialmente ordenado de patrones cerrados parciales, el cual permite incluir

representaciones de valores de orden superior, mientras que el dominio base en la aproximación de Colmerauer está formado por *árboles racionales*, posiblemente infinitos, los cuales no pueden ser interpretados como valores de orden superior.

- En segundo lugar, las restricciones de igualdad y de desigualdad están basadas en dos predicados diferentes en la aproximación de Colmerauer, mientras que en nuestra formalización de $\mathcal{H}$, una sola función primitiva con valor booleano permite expresar ambas restricciones de igualdad y de desigualdad estricta, como se verá más adelante. De manera más general, podemos decir que las restricciones en el esquema $CFLP(\mathcal{D})$ que proponemos se expresan siempre mediante el uso de funciones primitivas con semántica radical.

2.2.2. El dominio real $\mathcal{R}$

El siguiente ejemplo concreto de dominio presenta un dominio de restricciones $\mathcal{R}$ muy similar al que se utiliza habitualmente en el lenguaje lógico con restricciones $CLP(\mathcal{R})$ [JMSY92, JM94, MS98]. Si bien en el caso de *CLP* el conjunto base de $\mathcal{R}$ se define como el conjunto de todos los posibles *términos cerrados* construidos a partir de los números reales y las constructoras de datos, ahora en nuestro esquema *CFLP* usaremos un conjunto parcialmente ordenado de *patrones cerrados* que es, por tanto, estrictamente más grande.

De manera más concreta, en el contexto de nuestro marco *CFLP* podemos dar una definición formal del dominio $\mathcal{R}$ como sigue:

- Una signatura específica $\Sigma = \langle\ TC,\ SBT,\ DC,\ DF,\ SPF\ \rangle$, donde $SBT = \{real\}$ incluye exclusivamente un tipo básico cuyos valores representan números reales y *SPF* incluye los siguientes símbolos binarios de función primitiva, todos ellos pensados para ser utilizados en notación infija:
  - El operador de igualdad estricta $== :: A \to A \to bool$.
  - Los operadores aritméticos $+, -, *, / :: real \to real \to real$.
  - Los operadores de desigualdad $\leq, <, \geq, > :: real \to real \to bool$.
- Un conjunto de valores básicos $\mathcal{B}^{\mathcal{R}}_{real} = \mathbb{R}$ que coincide con el conjunto de los números reales.
- La interpretación $==^{\mathcal{R}}$ queda definida del mismo modo que para cualquier otro dominio de restricciones cuya signatura específica incluya la primitiva $==$.
- La interpretación de la primitiva $+^{\mathcal{R}}$ se define de tal modo que para todos $t_1, t_2, t \in \mathcal{U}_{\mathcal{R}}$, $t_1 +^{\mathcal{R}} t_2 \to t$ se satisface si y sólo si alguno de los casos siguientes se satisface:

- t_1, t_2 y t son números reales y t es igual a la suma de t_1 y t_2.
- $t = \bot$.

La interpretación de otras primitivas similares como $-, *$ y $/$ se define de un modo completamente análogo.

- La interpretación de la primitiva $\leq^{\mathcal{R}}$ se define de tal modo que para todos $t_1, t_2, t \in \mathcal{U}_{\mathcal{R}}$, $t_1 \leq^{\mathcal{R}} t_2 \rightarrow t$ se satisface si y sólo si alguno de los casos siguientes se satisface:

  - t_1, t_2 son números reales tales que t_1 es menor o igual que t_2 y $t = true$.
  - t_1, t_2 son números reales tales que t_1 es más grande que t_2 y $t = false$.
  - $t = \bot$.

  La interpretación de otras primitivas similares como $<, \geq$ y $>$ se define de un modo completamente análogo.

2.2.3. El dominio finito $\mathcal{FD}$

Otros dominios de restricciones conocidos por su valor práctico en programación con restricciones incluyen los denominados "*feature tree constraints*" [AP94, ST94, Bac95, BS95], los cuales pueden ser vistos como una extensión de los árboles racionales de Colmerauer [Col82, Col84]. Sin embargo, de entre todos los posibles dominios de restricciones, son los *dominios finitos* [vHen89, vHen91, vHSD94, vHSD98] los que gozan de una mayor popularidad en la comunidad *CLP*, debido a su gran aplicabilidad práctica en numerosos problemas (de planificación, optimización, problemas combinatorios, etc.) al permitir la computación con restricciones sobre números enteros. Ambos tipos de dominios de restricciones juegan un papel importante en lenguajes de programación como el lenguaje multiparadigma *Oz* [vRBDH+03]. Asimismo, en el caso concreto de las restricciones de dominio finito, estas han sido utilizadas recientemente para resolver problemas combinatorios en programación lógico funcional con restricciones [FHS03a], usando una extensión del sistema $\mathcal{TOY}$ [FHS03b, FHS03c].

En el contexto de nuestro esquema *CFLP* también es posible dar una definición formal del dominio finito $\mathcal{FD}$ de manera análoga a como hemos hecho en los casos anteriores para los dominios $\mathcal{H}$ y $\mathcal{R}$:

- La signatura específica de todo dominio finito $\mathcal{FD}$ es $\Sigma = \langle\ TC, SBT, DC, DF, SPF\ \rangle$, donde $SBT = \{int\}$ incluye exclusivamente un solo tipo básico, cuyos valores representan números enteros, y SPF incluye los siguientes símbolos de función primitiva:

- El operador de igualdad estricta $== :: A \to A \to bool$.
- Los operadores aritméticos $\#+, \#-, \#*, \#/ :: int \to int \to int$.
- Los siguientes símbolos de función primitiva relativos a la computación con dominios finitos:
  - $domain :: [int] \to int \to int \to bool$.
  - $belongs :: int \to [int] \to bool$.
  - $labeling :: [labelType] \to [int] \to bool$, donde $labelType$ es un tipo de datos enumerado usado para representar las diferentes *estrategias de etiquetado* que se comentarán más adelante.
- Los operadores de desigualdad $\#<=, \#<, \#>=, \#> :: int \to int \to bool$.

- El conjunto de valores básicos $\mathcal{B}_{int}^{\mathcal{FD}} = \mathbb{Z}$ es el conjunto de los números enteros.
- La interpretación $==^{\mathcal{FD}}$, definida del mismo modo que para cualquier otro dominio de restricciones cuya signatura específica incluya la primitiva $==$.
- La interpretación de la primitiva $\#+^{\mathcal{FD}}$, definida de tal modo que para todos $t_1, t_2, t \in \mathcal{U}_{\mathcal{FD}}$, $t_1 \ \#+^{\mathcal{FD}} t_2 \to t$ se satisface si y sólo si alguno de los casos siguientes se cumple:
  - t_1, t_2 y t son números enteros y t es igual a la suma entera de t_1 y t_2.
  - $t = \bot$.

  Las interpretaciones de $\#-, \#*$ y $\#/$ se definen de manera análoga.
- La interpretación de la primitiva $domain^{\mathcal{FD}}$, definida de tal modo que para todos $t_1, t_2, t_3, t \in \mathcal{U}_{\mathcal{FD}}$, $domain^{\mathcal{FD}}\ t_1\, t_2\, t_3 \to t$ se satisface si y sólo si alguno de los casos siguientes se cumple:
  - t_2 y t_3 son números enteros a y b tales que $a \leq_{\mathbb{Z}} b$, t_1 es una lista no vacía de números enteros que pertenecen al intervalo $a..b$ y $t = true$.
  - t_2 y t_3 son números enteros a y b tales que $a \leq_{\mathbb{Z}} b$, t_1 es una lista no vacía de números enteros, algunos de los cuales no pertenecen al intervalo $a..b$, y $t = false$.
  - t_2 y t_3 son números enteros a y b tales que $a >_{\mathbb{Z}} b$ y $t = false$.
  - $t = \bot$.
- La interpretación de la primitiva $belongs^{\mathcal{FD}}$, definida de tal modo que para todos $t_1, t_2, t \in \mathcal{U}_{\mathcal{FD}}$, $belongs^{\mathcal{FD}}\ t_1\, t_2 \to t$ se satisface si y sólo si alguno de los casos siguientes se cumple:

  - t_1 es un número entero, t_2 es una lista finita de números enteros que incluye t_1 como elemento y $t = true$.
  - t_1 es un número entero, t_2 es una lista finita de números enteros que no incluye t_1 como elemento y $t = false$.
  - $t = \bot$.

- La interpretación de la primitiva $labeling^{\mathcal{FD}}$, definida de tal modo que para todos t_1, t_2, $t \in \mathcal{U}_{\mathcal{FD}}$, $labeling^{\mathcal{FD}}\ t_1\, t_2 \rightarrow t$ se satisface si y sólo si alguno de los siguientes casos se cumple:

  - t_1 es un valor definido de tipo $labelType$, t_2 es una lista finita de números enteros y $t = true$.
  - $t = \bot$.

- La interpretación de la primitiva $\#<=^{\mathcal{FD}}$, definida de tal modo que para todos t_1, t_2, $t \in \mathcal{U}_{\mathcal{FD}}$, $t_1\ \#<=^{\mathcal{FD}}\ t_2 \rightarrow t$ se satisface si y sólo si alguno de los casos siguientes se cumple:

  - t_1, t_2 son números enteros tales que t_1 es menor o igual que t_2 y $t = true$.
  - t_1, t_2 son números enteros tales que t_1 es más grande que t_2 y $t = false$.
  - $t = \bot$.

  Las interpretaciones de $\#<, \#>=$ y $\#>$ se definen de manera análoga.

En la práctica, la primitiva *domain* se utiliza para poder restringir los posibles valores que puede tomar una variable entera con el fin de que estos pertenezcan siempre a un dominio finito concreto, representado mediante una lista creciente de enteros. Por otra parte, la primitiva *labeling* corresponde al denominado predicado de *etiquetado* [vHen89, MS98, SIC03]. El efecto operacional de las restricciones de etiquetado combinadas con restricciones de dominio es que se enumeren las soluciones correspondientes a todas las instanciaciones de ciertas variables enteras con valores enteros elegidos de sus dominios. La combinación inteligente del etiquetado con las técnicas de *propagación* usadas por los resolutores de dominios finitos con el fin de podar los dominios de las variables es crucial para el éxito de aplicaciones prácticas que se planteen sobre los dominios finitos [vHen91, FHS03a].

La elección de una estrategia particular de etiquetado tiene, por tanto, un efecto importante sobre el comportamiento operacional, y en consecuencia sobre su eficiencia (véanse [FHS03c, SIC03] para más detalles). Sin embargo, cualquier estrategia de etiquetado permite computar el mismo conjunto de soluciones, restringiendo a las variables enteras a tomar valores concretos a partir de sus dominios de todos los modos posibles. En este sentido, la interpretación semántica de la primitiva *labeling* es siempre la misma, con independencia del tipo de datos concreto usado para representar la estrategia de etiquetado que describa su comportamiento operacional.

2.3. Restricciones sobre un dominio

Asumimos un dominio de restricciones $\mathcal{D}$ fijado arbitrariamente y con dominio de valores $\mathcal{U}_\mathcal{D}$. Definimos a continuación la sintaxis de las restricciones que utilizaremos en el esquema $CFLP(\mathcal{D})$ sobre el dominio $\mathcal{D}$. Como en el caso de CLP, las restricciones se representarán mediante fórmulas lógicas construidas a partir de restricciones atómicas usando conjunción y cuantificación existencial. Sin embargo, en contraste con CLP, nuestras restricciones pueden incluir también apariciones de funciones definidas por el usuario.

2.3.1. Sintaxis de las restricciones

La siguiente definición permite distinguir entre *restricciones primitivas*, sin ninguna aparición activa de símbolos de función definida, y *restricciones definidas* por el usuario, que sí pueden tener este tipo de apariciones de símbolos. En ocasiones, utilizaremos simplemente el nombre de 'restricciones', en lugar de 'restricciones definidas por el usuario', para referirnos a este segundo tipo de restricciones sobre el dominio.

Definición 2 (Sintaxis de Restricciones) *Definimos la sintaxis de los siguientes tipos de restricciones sobre un dominio $\mathcal{D}$ dado.*

(1) Las restricciones primitivas atómicas *tienen la forma sintáctica $p\,\bar{t}_n \to! \; t$, con $p \in SPF^n$, $\bar{t}_n \in Pat_\mathcal{D}$, $t \in Pat_\mathcal{D}$ y t se requiere que sea total (es decir, sin apariciones de $\bot$). Las constantes especiales $\Diamond$ y $\blacklozenge$, las cuales representan, respectivamente, un valor de cierto (éxito) y de falso (fallo), son también restricciones primitivas atómicas.*

(2) Las restricciones primitivas *se construyen a partir de restricciones primitivas atómicas mediante conjunción lógica $\wedge$ y cuantificación existencial $\exists$. De manera más precisa, las restricciones π sobre el dominio de restricciones $\mathcal{D}$ tienen la sintaxis*

$$\pi ::= \alpha \mid (\pi_1 \wedge \pi_2) \mid \exists X.\ \pi$$

donde α es cualquier restricción primitiva atómica sobre $\mathcal{D}$ y $X \in \mathcal{V}ar$ es cualquier variable.

(3) Las restricciones atómicas *tienen la forma sintáctica $p\,\bar{e}_n \to! \; t$, con $p \in SPF^n$, $\bar{e}_n \in Exp_\mathcal{D}$, $t \in Pat_\mathcal{D}$ y t se requiere que sea total. Las constantes especiales $\Diamond$ y $\blacklozenge$ son también restricciones atómicas.*

(4) Las restricciones *se construyen a partir de restricciones atómicas por medio de conjunción lógica $\wedge$ y de cuantificación existencial $\exists$. De manera más precisa, las restricciones δ sobre el dominio de restricciones $\mathcal{D}$ tienen la sintaxis:*

$$\delta ::= \alpha \mid (\delta_1 \wedge \delta_2) \mid \exists X.\ \delta$$

donde α es cualquier restricción atómica de $\mathcal{D}$ y $X \in \mathcal{V}ar$ es cualquier variable.

Intuitivamente, una restricción atómica $p\,\overline{e}_n \rightarrow! \ t$ restringe el valor devuelto por la llamada $p\,\overline{e}_n$ a que sea un patrón total que se ajuste a la forma de t. Por convenio, las restricciones de la forma $p\,\overline{e}_n \rightarrow! \ true$ se abreviarán en la forma $p\,\overline{e}_n$. En ocasiones, las restricciones de la forma $p\,\overline{e}_n \rightarrow! \ false$ se abreviarán como $p'\,\overline{e}_n$, haciendo uso de un símbolo p' (no necesariamente perteneciente a SPF) para sugerir la 'negación' de p. En particular, las *restricciones de igualdad estricta* $e_1 == e_2$ y las *restricciones de desigualdad estricta* $e_1 /= e_2$ se entenderán como abreviaturas de $e_1 == e_2 \rightarrow! \ true$ y de $e_1 == e_2 \rightarrow! \ false$, respectivamente.

En lo sucesivo, escribiremos $Con_{\mathcal{D}}$ (respectivamente, $ACon_{\mathcal{D}}$) para representar el conjunto de todas las restricciones (respectivamente, restricciones atómicas) sobre $\mathcal{D}$. Dos subconjuntos útiles de $Con_{\mathcal{D}}$ son los siguientes:

- El conjunto $APCon_{\mathcal{D}}$ de todas las *restricciones primitivas atómicas* π sobre $\mathcal{D}$. Por definición, una restricción atómica $\pi \in ACon_{\mathcal{D}}$ se denomina primitiva si y sólo si π no incluye subexpresiones de la forma $f\,\overline{e}_n$ con $f \in DF^n$. En particular, todas las restricciones atómicas de la forma $\Diamond$, $\blacklozenge$ o $p\,\overline{t}_n \rightarrow! \ t$ (donde $\overline{t}_n \in Pat_{\mathcal{D}}$ son patrones) son primitivas.
- El conjunto $PCon_{\mathcal{D}}$, formado por todas las *restricciones primitivas*, construidas a partir de restricciones primitivas atómicas por medio de conjunción y cuantificación existencial. Se observa así que $APCon_{\mathcal{D}} = ACon_{\mathcal{D}} \cap PCon_{\mathcal{D}}$.

Una aparición particular de una variable X dentro de una restricción δ (respectivamente, de una restricción primitiva π) se denomina *libre* si y sólo si no está afectada por ninguna cuantificación, y se denomina *ligada* en otro caso. En lo que sigue, escribiremos $var(\pi)$ y $var(\delta)$ (respectivamente, $fvar(\pi)$ y $fvar(\delta)$) para representar el conjunto de todas las variables que tienen alguna aparición (respectivamente, alguna aparición libre) en las restricciones π y δ.

Adicionalmente, utilizaremos las siguientes notaciones para representar los subconjuntos útiles de $PCon_{\mathcal{D}}$:

- $GPCon_{\mathcal{D}}$, el conjunto de todas las restricciones primitivas cerradas sobre $\mathcal{D}$, definido como $\{\pi \in PCon_{\mathcal{D}} \mid fvar(\pi) = \emptyset\}$.
- $TPCon_{\mathcal{D}}$, el conjunto de todas las restricciones primitivas totales sobre $\mathcal{D}$, definido como $\{\pi \in PCon_{\mathcal{D}} \mid \pi \text{ no tiene apariciones de } \bot\}$.
- $GTPCon_{\mathcal{D}}$, el conjunto de todas las restricciones primitivas cerradas y totales, definido como $GPCon_{\mathcal{D}} \cap TPCon_{\mathcal{D}}$.

También escribiremos en ocasiones $GCon_{\mathcal{D}}$, $TCon_{\mathcal{D}}$ y $GTCon_{\mathcal{D}}$ para representar los subconjuntos de $Con_{\mathcal{D}}$ formados por restricciones cerradas, totales, y cerradas y totales, respectivamente. Además, reservaremos la letra griega mayúscula Π (respectivamente, Δ) para los conjuntos de restricciones primitivas (respectivamente, restricciones definidas por el usuario), interpretados usualmente como conjunciones. Las notaciones $fvar(\Pi)$ (respectivamente, $fvar(\Delta)$) se referirán al conjunto de variables libres que aparecen en los conjuntos $\Pi \subseteq PCon_{\mathcal{D}}$ (respectivamente, $\Delta \subseteq Con_{\mathcal{D}}$).

Restricciones bien tipadas

Las reglas de inferencia de tipos presentadas en la Subsección 2.1.2 pueden extenderse de una forma natural con el fin de poder derivar sentencias de tipos de la forma $\Sigma, \Gamma \vdash_{WT} \delta$, cuyo significado sea el de que la restricción δ esté bien tipada con respecto a la suposición de tipos para símbolos (respectivamente, para variables), dada en Σ (respectivamente, en Γ). En ejemplos concretos, simplemente afirmaremos que δ está *bien tipada* para indicar que $\Sigma, \Gamma \vdash_{WT} \delta$ puede ser derivada usando la signatura subyacente Σ y algún tipo adecuado de contexto de tipos Γ. Por ejemplo, las siguientes restricciones están bien tipadas:

- $domain\ [X, Y]\ 0\ N$ está bien tipada (con respecto a cualquier contexto de tipos que incluya las suposiciones de tipos $X :: int$, $Y :: int$, $N :: int$).
- $RY + RX \leq RY_0 + RX_0$ está también bien tipada (con respecto a cualquier contexto de tipos que incluya las suposiciones de tipos $RY :: real$, $RX :: real$, $RY_0 :: real$, $RX_0 :: real$).

Por supuesto, la signatura subyacente permite también escribir restricciones tales como $domain\ [X, Y]\ true$ 3.2, las cuales no pueden estar bien tipadas en ningún contexto de tipos. Debido a la disciplina estática de tipos, el compilador rechazará cualquier programa que incluya restricciones mal tipadas.

2.3.2. Semántica de las restricciones primitivas

La semántica de las restricciones definidas por el usuario depende de la interpretación concreta de las funciones definidas a través de las reglas de un programa dadas por el usuario. Es decir, para todas aquellas restricciones δ que incluyan subexpresiones de la forma $f\,\overline{e}_n$ para algún $f \in DF^n$, las soluciones de δ van a depender del comportamiento de f, el cual no está incluido en el dominio $\mathcal{D}$, sino que debe ser deducido a partir de algún $CFLP(\mathcal{D})$-programa dado por el usuario. Por esta razón, su semántica será presentada en el Capítulo 3 como parte de la semántica de los $CFLP(\mathcal{D})$-programas.

Sin embargo, la semántica de las restricciones primitivas y su concepto asociado de solución sí depende exclusivamente del dominio $\mathcal{D}$, por lo que pasamos a presentar su definición formal en esta subsección.

Definición 3 (Soluciones de Restricciones Primitivas) *Definimos las siguientes nociones y notaciones en relación a la semántica de las restricciones primitivas:*

(1) El conjunto de las valoraciones $Val_\mathcal{D}$ *sobre el dominio* $\mathcal{D}$ *se define como el conjunto de todas aquellas sustituciones cerradas* $\eta \in GSub_\mathcal{D}$ *tales que* $ran(\eta) \subseteq \mathcal{U}_\mathcal{D}$*, siendo* $ran(\eta) = \{X\eta \in vdom(\eta)\}$ *el* rango *de la sustitución* η*. Análogamente, el conjunto de valoraciones totales sobre* $\mathcal{D}$ *se define como* $TVal_\mathcal{D} = GTSub_\mathcal{D}$*.*

(2) Aquellas valoraciones que satisfacen una restricción dada se denominan soluciones*. De manera más precisa, el conjunto de soluciones de una restricción primitiva* $\pi \in PCon_\mathcal{D}$ *es un subconjunto* $Sol_\mathcal{D}(\pi) \subseteq Val_\mathcal{D}$ *definido recursivamente sobre la estructura sintáctica de* π *como sigue:*

(a) $Sol_\mathcal{D}(\Diamond) = Val_\mathcal{D}$*.*

(b) $Sol_\mathcal{D}(\blacklozenge) = \emptyset$*.*

(c) $Sol_\mathcal{D}(p\,\bar{t}_n \to!\ t) = \{\eta \in Val_\mathcal{D} \mid (p\,\bar{t}_n \to!\ t)\eta$ es cerrado y $p^\mathcal{D}\bar{t}_n\eta \to t\eta$, siendo $t\eta$ total$\}$*.*

(d) $Sol_\mathcal{D}(\pi_1 \wedge \pi_2) = Sol_\mathcal{D}(\pi_1) \cap Sol_\mathcal{D}(\pi_2)$*.*

(e) $Sol_\mathcal{D}(\exists X.\ \pi) = \{\eta \in Val_\mathcal{D} \mid$ existe $\eta' \in Sol_\mathcal{D}(\pi)$ tal que $\eta' =_{\setminus\{X\}} \eta\}$*.*

Para cualquier solución $\eta \in Sol_\mathcal{D}(\pi)$ *observamos que se ha de verificar que* $vdom(\eta) \supseteq fvar(\pi)$*.*

(3) El conjunto de soluciones de un conjunto de restricciones primitivas $\Pi \subseteq PCon_\mathcal{D}$ *se define como* $Sol_\mathcal{D}(\Pi) = \bigcap_{\pi\in\Pi} Sol_\mathcal{D}(\pi)$*, correspondiendo a una lectura lógica de* Π *como la conjunción de sus miembros. En particular,* $Sol_\mathcal{D}(\emptyset) = Val_\mathcal{D}$*, correspondiendo a la lectura lógica de una conjunción vacía que se identifica con la restricción idénticamente cierta* $\Diamond$*.*

De acuerdo con el apartado (2)(c) de esta definición, las soluciones de una restricción primitiva atómica $p\,\bar{t}_n \to! t$ son todas aquellas valoraciones para las cuales $p\,\bar{t}_n$ puede devolver un valor que se ajusta al patrón total t. Por ejemplo, $\eta \in Sol_\mathcal{R}(X+Y \to! 5)$ se satisface si y sólo si $\eta(X) = x \in \mathbb{R}$, $\eta(Y) = y \in \mathbb{R}$ y $x +^\mathcal{R} y = 5$. Los demás enunciados de la definición son bastante usuales en programación con restricciones.

Para un uso posterior, vamos a considerar el siguiente lema técnico, el cual puede demostrarse fácilmente razonando por inducción sobre la estructura sintáctica de Π:

Lema 3 (Lema de Sustitución) *Para cualquier conjunto* $\Pi \subseteq PCon_\mathcal{D}$*, cualquier sustitución* $\sigma \in Sub_\mathcal{D}$ *y cualquier solución* $\eta \in Val_\mathcal{D}$*, se cumple la siguiente equivalencia:*

$$\eta \in Sol_{\mathcal{D}}(\Pi\sigma) \Leftrightarrow \sigma\eta \in Sol_{\mathcal{D}}(\Pi)$$

La semántica de restricciones atómicas $p\,\bar{t}_n \rightarrow\!!\, t$ de nuestro esquema *CFLP* depende directamente del comportamiento de las funciones primitivas disponibles en el dominio $\mathcal{D}$ sobre el que estamos trabajando, las cuales pueden ser de tipo booleano o no serlo. En el caso de una primitiva p de tipo booleano, una restricción de la forma $p\,\bar{t}_n \rightarrow\!!\, R$ (donde R es una variable) impone siempre una relación entre el valor de R y el resultado que ha de ser devuelto por $p\,\bar{t}_n$, el cual sólo puede ser $true$ o $false$. Este comportamiento podría entenderse, en algunos contextos de la programación con restricciones, como la evidencia de que existe una cierta analogía formal entre las restricciones de la forma $p\,\bar{t}_n \rightarrow\!!\, R$ de nuestro esquema *CFLP* y las denominadas *restricciones reificadas* de la forma $c\,\# \Leftrightarrow R$, usadas en diversos lenguajes de programación con restricciones para relacionar el valor de una variable booleana R con el cumplimiento o no de una cierta restricción c (ver [SIC03] para una explicación más detallada del uso de restricciones reificadas en SICStus Prolog). Esta analogía, sin embargo, debe entenderse con cautela; en nuestro contexto *CFLP*, la restricción $p\,\overline{t_n} \rightarrow\!!\, R$ no es entendida como una relación entre el valor de la variable R y el comportamiento de otra restricción $p\,\bar{t}_n$. De hecho, $p\,\bar{t}_n$, por sí misma, no es una restricción, sino una expresión funcional como ha quedado explicado en las secciones anteriores. Además, algunos aspectos operacionales del comportamiento de las restricciones reificadas en SICStus Prolog y otros lenguajes de programación lógica con restricciones no están cubiertos por la semántica de nuestro esquema.

En cualquier caso, nuestras restricciones atómicas $p\,\bar{t}_n \rightarrow\!!\, t$, basadas en el comportamiento de funciones primitivas, sí son más expresivas que las restricciones atómicas puramente relacionales. Esto ha sido argumentado, de algún modo, en [LS03] para el caso particular de las restricciones de igualdad y desigualdad estricta sobre términos construidos. Con el fin de clarificar este aspecto, vamos a considerar un ejemplo concreto de restricciones de igualdad y desigualdad sobre los números reales. De acuerdo con el punto de vista tradicional (relacional), se podría hacer uso de dos predicados primitivos diferentes, digamos $=_{\mathcal{R}}$ y $/=_{\mathcal{R}}$, para escribir restricciones atómicas tales como $X =_{\mathcal{R}} Y$ o $X /=_{\mathcal{R}} Y$. En $CFLP(\mathcal{R})$, estas restricciones atómicas pueden ser escritas como $X == Y \rightarrow\!!\, true$ y $X == Y \rightarrow\!!\, false$, respectivamente. Más aún, también se podría escribir la restricción atómica $X == Y \rightarrow\!!\, R$, cuyo uso en programación da lugar a una mayor expresividad. Incluso podría esperarse también una mejora de eficiencia en los cómputos dependiendo del valor obtenido para la variable R mediante la resolución de restricciones, como veremos más adelante, debido a que es posible resolver la restricción $X == Y \rightarrow\!!\, R$ una sola vez, en lugar de tener que comprobar cuál de las dos posibles restricciones, $X =_{\mathcal{R}} Y$ y $X /=_{\mathcal{R}} Y$, tiene éxito. Consideraciones similares pueden aplicarse a otras primitivas en el dominio $\mathcal{R}$, así como a la primitiva de igualdad estricta $==$ en cualquier otro dominio de restricciones en el que esta primitiva se encuentre disponible.

Soluciones bien tipadas

Debido a la disciplina de tipos introducida en las primeras secciones de este capítulo, estaremos interesados en determinar el conjunto $WTSol_{\mathcal{D}}(\pi) \subseteq Sol_{\mathcal{D}}(\pi)$ de todas aquellas soluciones bien tipadas de una restricción $\pi \in PCon_{\mathcal{D}}$ que sea a su vez bien tipada. Este conjunto se define mediante recursión sobre la estructura sintáctica de π, de manera similar a como se ha definido el conjunto de soluciones $Sol_{\mathcal{D}}(\pi)$, excepto que ahora, en el caso $WTSol_{\mathcal{D}}(\pi)$ en el que π sea una restricción atómica, la restricción atómica instanciada $\pi\eta$ se requiere que sea también bien tipada además de cerrada. El conjunto $WTSol_{\mathcal{D}}(\Pi) \subseteq Sol_{\mathcal{D}}(\Pi)$ de todas las soluciones bien tipadas de un conjunto de restricciones bien tipadas $\Pi \subseteq PCon_{\mathcal{D}}$ se define análogamente.

Nociones básicas de la semántica de restricciones primitivas

Usando la noción de solución dada en la Definición 3, es posible definir también algunas otras nociones semánticas naturales relativas a las restricciones primitivas que resultarán de utilidad en adelante.

Definición 4 (Nociones Semánticas Primitivas) *Asumamos un conjunto finito* $\Pi \subseteq PCon_{\mathcal{D}}$ *de restricciones primitivas, una restricción primitiva* $\pi \in PCon_{\mathcal{D}}$*, expresiones* $e, e' \in Exp_{\mathcal{D}}$*, patrones* $\bar{t}_n, t \in Pat_{\mathcal{D}}$*, y un símbolo de función primitiva* $p \in SPF^n$ *en la signatura específica del dominio* $\mathcal{D}$*. Definimos las siguientes nociones:*

(1) π *se denomina* satisfactible *en* $\mathcal{D}$ *(en símbolos,* $Sat_{\mathcal{D}}(\pi)$*) si y sólo si* $Sol_{\mathcal{D}}(\pi) \neq \emptyset$*. En otro caso,* π *se denomina* insatisfactible *(en símbolos,* $Insat_{\mathcal{D}}(\pi)$*). Analogamente se definiría la satisfactibilidad o insatisfactibilidad para conjuntos de restricciones* $\Pi \subseteq PCon_{\mathcal{D}}$*.*

(2) π *es* consecuencia *de* Π *en* $\mathcal{D}$ *(en símbolos,* $\Pi \models_{\mathcal{D}} \pi$*) si y sólo si* $Sol_{\mathcal{D}}(\Pi) \subseteq Sol_{\mathcal{D}}(\pi)$*. Decimos que* π *es* válido *en* $\mathcal{D}$ *(en símbolos,* $\models_{\mathcal{D}} \pi$*) si y sólo si* $\emptyset \models_{\mathcal{D}} \pi$*, lo que es equivalente a pedir que* $Sol_{\mathcal{D}}(\pi) = Val_{\mathcal{D}}$*. En particular,* $p\,\bar{t}_n \to! \; t$ *es* consecuencia *de* Π *en* $\mathcal{D}$ *(en símbolos,* $\Pi \models_{\mathcal{D}} p\,\bar{t}_n \to! \; t$*) si y sólo si* $p^{\mathcal{D}}\,\bar{t}_n\eta \to t\eta$ *se cumple para todo* $\eta \in Sol_{\mathcal{D}}(\Pi)$*. Decimos que* $p\,\bar{t}_n \to! \; t$ *es válido en* $\mathcal{D}$ *(en símbolos,* $\models_{\mathcal{D}} p\,\bar{t}_n \to! \; t$*) si y sólo si* $\emptyset \models_{\mathcal{D}} p\,\bar{t}_n \to! \; t$*, es decir, si y sólo si* $p^{\mathcal{D}}\,\bar{t}_n\eta \to t\eta$ *se cumple para todo* $\eta \in Val_{\mathcal{D}}$*.*

(3) $e \sqsubseteq e'$ *es* consecuencia *de* Π *en* $\mathcal{D}$ *(en símbolos,* $\Pi \models_{\mathcal{D}} e \sqsubseteq e'$*) si y sólo si* $e\eta \sqsubseteq e'\eta$ *se cumple para todo* $\eta \in Sol_{\mathcal{D}}(\Pi)$*. Decimos que* $e \sqsubseteq e'$ *es* válido *en* $\mathcal{D}$ *(en símbolos,* $\models_{\mathcal{D}} e \sqsubseteq e'$*) si y sólo si* $\emptyset \models_{\mathcal{D}} e \sqsubseteq e'$*, lo que es equivalente a pedir que* $e\eta \sqsubseteq e'\eta$ *se cumple para todo* $\eta \in Val_{\mathcal{D}}$*. Finalmente,* $\Pi \models_{\mathcal{D}} e \sqsupseteq e'$ *y* $\models_{\mathcal{D}} e \sqsupseteq e'$ *se definen análogamente.*

(4) Π *es* admisible *en* $\mathcal{D}$ *si y solo si se cumple alguno de los dos casos siguientes: (a)* $Insat_{\mathcal{D}}(\Pi)$ *o (b)* $Sat_{\mathcal{D}}(\Pi)$ *y no se pueden encontrar una variable* $X \in \mathcal{V}ar$ *y un patrón* t *tales que* $\Pi \models_{\mathcal{D}} X \sqsupseteq t$ *y* t *es de la forma* $h\bar{t}_m$ *con* $m < ar(h)$*.*

2.3.3. Almacenes de restricciones

Tanto en este capítulo como en los siguientes, trabajaremos con *almacenes de restricciones* de la forma $\Pi \,\square\, \sigma$, donde $\Pi \subseteq APCon_{\mathcal{D}}$ y σ es una sustitución idempotente tal que $vdom(\sigma) \cap var(\Pi) = \emptyset$. Análogamente, con frecuencia necesitaremos trabajar con soluciones de almacenes de restricciones, posiblemente afectadas por un prefijo existencial. Para este propósito, interpretaremos el símbolo separador $\square$ como conjunción lógica, y definiremos el conjunto de soluciones $Sol_{\mathcal{D}}(\Pi \,\square\, \sigma)$ en el modo siguiente:

- $Sol_{\mathcal{D}}(\exists \overline{Y}.\ (\Pi \,\square\, \sigma)) = \{\eta \in Val_{\mathcal{D}} \mid \text{existe } \eta' \in Sol_{\mathcal{D}}(\Pi \,\square\, \sigma) \text{ tal que } \eta' =_{\setminus \overline{Y}} \eta\}$.
- $Sol_{\mathcal{D}}(\Pi \,\square\, \sigma) = Sol_{\mathcal{D}}(\Pi) \cap Sol(\sigma)$.
- $Sol(\sigma) = \{\eta \in Val_{\mathcal{D}} \mid \eta = \sigma\eta\}$.

Observamos que $\eta = \sigma\eta$ se cumple si y sólo si $X\eta = X\sigma\eta$ para todo $X \in vdom(\sigma)$.

2.4. Resolutores de restricciones sobre un dominio

Con el fin de que los dominios de restricciones puedan ser de utilidad práctica desde el punto de vista de la programación en *CFLP*, necesitamos proporcionar para cada dominio concreto un *resolutor de restricciones*. Mediante los resolutores de restricciones será posible procesar, simplificar y en ocasiones resolver, todas aquellas restricciones que vayan surgiendo en el transcurso de la computación.

Para algunos propósitos teóricos, es suficiente con formalizar un resolutor como una función matemática que envía cualquier restricción dada a uno de los siguientes tres valores: *true*, $false$ o *unknown*; véase en este sentido, por ejemplo, la publicación [JMMS98]. Sin embargo, en la práctica, se espera que los resolutores tengan la habilidad de reducir restricciones primitivas a lo que se denominan *formas resueltas*, las cuales son restricciones más simples que pueden mostrarse como *respuestas computadas* a los usuarios.

En el resto de esta subsección vamos a dar una formalización matemática de la noción de resolutor para un dominio de restricciones $\mathcal{D}$ en el esquema $CFLP(\mathcal{D})$, así como a justificar la existencia de resolutores adecuados para los dominios $\mathcal{H}$, $\mathcal{R}$ y $\mathcal{FD}$ que hemos introducido en las secciones anteriores. Previamente, daremos algunas nociones que resultarán imprescindibles para poder adaptar la noción de resolutor a las necesidades particulares del esquema *CFLP*, como es el uso de *variables demandadas* y de *variables críticas*.

2.4.1. Variables demandadas y críticas

En algunas ocasiones, es posible observar que una solución dada $\eta \in Sol_{\mathcal{D}}(\Pi)$ puede vincular alguna variable X que aparezca en alguna restricción de Π al valor indefinido

$\bot$. Intuitivamente, esto sucede cuando el valor de X no es necesario para comprobar si se satisface o no la restricción en Π. Formalmente, diremos que una variable X es *demandada* por un conjunto de restricciones $\Pi \subseteq PCon_{\mathcal{D}}$ si y sólo si $\eta(X) \neq \bot$ para toda $\eta \in Sol_{\mathcal{D}}(\Pi)$. Escribiremos $dvar_{\mathcal{D}}(\Pi)$ para denotar el conjunto de todas las variables $X \in fvar(\Pi)$ tales que X es demandada por Π.

En la práctica, la programación *CFLP* requiere de procedimientos efectivos que permitan reconocer apariciones 'obvias' de variables demandadas en el caso en el que Π sea un conjunto de restricciones atómicas. Por este motivo, asumimos que para cualquier dominio práctico de restricciones $\mathcal{D}$ y para cualquier restricción atómica primitiva $\pi \in APCon_{\mathcal{D}}$, existe un mecanismo efectivo para computar un subconjunto $odvar_{\mathcal{D}}(\pi) \subseteq dvar_{\mathcal{D}}(\pi)$. Diremos que las variables $X \in odvar_{\mathcal{D}}(\pi)$ son *obviamente demandadas* por π. Análogamente, extendemos esta noción a un conjunto de restricciones finitas $\Pi \subseteq APCon_{\mathcal{D}}$ definiendo el conjunto de todas las variables *obviamente demandadas* por Π como $odvar_{\mathcal{D}}(\Pi) = \bigcup_{\pi \in \Pi} odvar_{\mathcal{D}}(\pi)$. De este modo, es claro que $odvar_{\mathcal{D}}(\Pi) \subseteq dvar_{\mathcal{D}}(\Pi)$ se cumple para cualquier $\Pi \subseteq APCon_{\mathcal{D}}$; es decir, las variables obviamente demandadas son siempre variables demandadas. Es más, la inclusión es estricta en general.

En particular, para cualquier dominio de restricciones $\mathcal{D}$ cuya signatura específica incluya la primitiva de igualdad estricta $==$ y una restricción atómica primitiva de la forma $\pi = (t_1 == t_2 \rightarrow! \ r)$, el conjunto de variables obviamente demandadas $odvar(\pi)$ se define como el conjunto más pequeño de variables que satisfaga la siguiente distinción de casos:

- $odvar(t_1 == t_2 \rightarrow! \, R) = \{R\}$, si $R \in \mathcal{V}ar$.
- $odvar(X == Y) = \{X, Y\}$, si $X, Y \in \mathcal{V}ar$.
- $odvar(X == t) = odvar(t == X) = \{X\}$, si $X \in \mathcal{V}ar$ y $t \notin \mathcal{V}ar$.
- $odvar(t_1 == t_2) = \emptyset$, en otro caso.
- $odvar(X \ /= Y) = \{X, Y\}$, si $X, Y \in \mathcal{V}ar$, y X e Y no son idénticas.
- $odvar(X \ /= t) = odvar(t \ /= X) = \{X\}$, si $X \in \mathcal{V}ar$ y $t \notin \mathcal{V}ar$.
- $odvar(t_1 \ /= t_2) = \emptyset$, en otro caso.

En este caso, la inclusión $odvar_{\mathcal{D}}(\pi) \subseteq dvar_{\mathcal{D}}(\pi)$ es fácil de comprobar, considerando el comportamiento de la interpretación de la operación de igualdad estricta $==^{\mathcal{D}}$. El método para poder computar $odvar_{\mathcal{D}}(\pi)$ en el caso de restricciones atómicas primitivas basadas en funciones primitivas distintas de la igualdad $==$ debe darse como parte integrante de la presentación práctica de cada dominio $\mathcal{D}$ que vaya a utilizarse.

En lo sucesivo, llamaremos *variables criticas* a todas aquellas variables que aparecen en Π que no sean obviamente demandadas, y escribiremos

$$cvar_{\mathcal{D}}(\Pi) = var(\Pi) \setminus odvar_{\mathcal{D}}(\Pi)$$

para representar el conjunto de todas aquellas variables que son críticas en Π. Se observa así que una variable puede ser crítica bien porque es demandada pero no obviamente demandada, o bien porque no era de por sí demandada.

Como ejemplo concreto, podemos observar que las variables A y B son variables demandadas, pero no variables obviamente demandadas, por la restricción de igualdad estricta $(A, 2) == (1, B)$, y por tanto, son variables críticas. Con el fin de ilustrar el caso de variables que no son demandadas, consideramos, por ejemplo, la restricción primitiva de desigualdad estricta $\pi = L /= (X : Xs)$. La variable L es obviamente demandada por π, debido a la propia definición de 'obviamente demandada' para la primitiva $==$. Sin embargo, ni X ni Xs son variables demandadas por π. De manera más precisa, X no es una variable demandada porque existen soluciones $\eta \in Sol_{\mathcal{D}}(\pi)$ tales que $\eta(X) = \bot$ (bien con $\eta(L) = [\,]$ o bien con $\eta(L) = (t : ts)$ tal que $\eta(Xs)$ es diferente de ts) y Xs no es una variable demandada debido a que se pueden argumentar razones similares. Por tanto, ambas variables X y Xs son variables críticas.

Como se verá en el Capítulo 4, los métodos de resolución de objetivos para el esquema de programación lógico funcional con restricciones $CFLP(\mathcal{D})$ se basan en el reconocimiento efectivo de variables críticas. Por tanto, el comportamiento apropiado de la resolución de objetivos dependerá de los métodos que se escojan para la computación de variables obviamente demandadas.

2.4.2. Resolutores de restricciones

Para un dominio de restricciones $\mathcal{D}$ cualquiera, vamos a postular la existencia de un *resolutor de restricciones* mediante el cual sea posible reducir cualquier conjunto finito Π de restricciones atómicas primitivas a una forma equivalente más simple. Más aún, en la operación de resolución necesitaremos tener en cuenta las apariciones de variables críticas en Π. Puesto que el valor de una variable crítica X puede ser necesario para alguna solución de Π e irrelevante para alguna otra solución, requeriremos a los resolutores que tengan la capacidad de computar una distinción de casos adecuada con el fin de poder discriminar tales situaciones.

Definición 5 (Resolutores de Restricciones) *Un resolutor de restricciones para el dominio $\mathcal{D}$ viene dado por una función $solve^{\mathcal{D}}$ aplicada a parejas de la forma $(\Pi, \mathcal{X})$, donde $\Pi \subseteq APCon(\mathcal{D})$ es un conjunto finito de restricciones primitivas y $\mathcal{X} \subseteq cvar_{\mathcal{D}}(\Pi)$ es un conjunto finito que incluye algunas de las variables críticas de Π,*

donde los dos casos extremos $\mathcal{X} = \emptyset$ *y* $\mathcal{X} = cvar_{\mathcal{D}}(\Pi)$ *están permitidos. Por convenio, abreviaremos* $solve^{\mathcal{D}}(\Pi, \emptyset)$ *como* $solve^{\mathcal{D}}(\Pi)$*. Adicionalmente, requeriremos que cualquier invocación de un resolutor* $solve^{\mathcal{D}}(\Pi, \mathcal{X})$ *devuelva una disyunción finita* $\bigvee_{j=1}^{k} \exists \overline{Y}_j.\ (\Pi_j \sqcap \sigma_j)$ *de almacenes de restricciones cuantificados existencialmente, satisfaciendo las siguientes condiciones:*

(1) Variables locales frescas: *Para todo* $1 \leq j \leq k$, $\Pi_j \sqcap \sigma_j$ *es un almacén de restricciones tal que* $\overline{Y}_j = var(\Pi_j \sqcap \sigma_j) \setminus var(\Pi)$ *son variables locales frescas, y* $vdom(\sigma_j) \cup vran(\sigma_j) \subseteq var(\Pi) \cup \overline{Y}_j$.

(2) Formas resueltas: *Para todo* $1 \leq j \leq k$, $\Pi_j \sqcap \sigma_j$ *es un almacén de restricciones en* forma resuelta *con respecto a* $\mathcal{X}$*. Por definición, esto significará que* $solve^{\mathcal{D}}(\Pi_j, \mathcal{X}) = \Pi_j \sqcap \varepsilon$.

(3) Vinculación segura de variables críticas: *Para todo* $1 \leq j \leq k$ *y para todo* $X \in \mathcal{X} \cap vdom(\sigma_j)$ *se satisface que* $\sigma_j(X)$ *es una constante.*

(4) Discriminación: *Cada forma resuelta computada* $\Pi_j \sqcap \sigma_j$ $(1 \leq j \leq k)$ *debe satisfacer que, o bien* $\mathcal{X} \cap odvar(\Pi_j) \neq \emptyset$ *o bien* $\mathcal{X} \cap var(\Pi_j) = \emptyset$ *(es decir, o bien alguna variable crítica se convierte en obviamente demandada o bien todas las variables críticas desaparecen).*

(5) Corrección: $Sol_{\mathcal{D}}(\Pi) \supseteq \bigcup_{j=1}^{k} Sol_{\mathcal{D}}(\exists \overline{Y}_j.\ (\Pi_j \sqcap \sigma_j))$.

(6) Completitud: $WTSol_{\mathcal{D}}(\Pi) \subseteq \bigcup_{j=1}^{k} WTSol_{\mathcal{D}}(\exists \overline{Y}_j.\ (\Pi_j \sqcap \sigma_j))$.

La presentación de cálculos concretos de resolución de objetivos en el Capítulo 4 nos permitirá motivar el modo más adecuado de elegir un conjunto $\mathcal{X}$ de variables críticas para cada invocación particular del resolutor. Por el momento, tan sólo indicaremos que la idea que se va a seguir para llevar a cabo esta elección es la de que $\mathcal{X}$ incluya todas aquellas variables críticas que están esperando a ser vinculadas al resultado que se obtendría al evaluar alguna expresión en algún otro lugar dentro del objetivo. Esta idea también motiva la condición de *vinculación segura de variables críticas* dada en la definición de resolutor de restricciones.

Operacionalmente, las alternativas correspondientes a las disyunciones que son calculadas por las invocaciones al resolutor se exploran usualmente siguiendo algún tipo de orden secuencial con la ayuda de un mecanismo de vuelta atrás o "backtracking". Asumiendo que $solve^{\mathcal{D}}(\Pi, \mathcal{X}) = \bigvee_{j=1}^{k} \exists \overline{Y}_j.\ (\Pi_j \sqcap \sigma_j)$, en ocasiones haremos uso de las siguiente notaciones:

- $\Pi \vdash_{solve^{\mathcal{D}}_{\mathcal{X}}} \exists \overline{Y'}.\ (\Pi' \sqcap \sigma')$, para indicar que $\exists\ \overline{Y'}.(\Pi' \sqcap \sigma')$ es $\exists \overline{Y}_j.\ (\Pi_j \sqcap \sigma_j)$ para algún $1 \leq j \leq k$. En este caso, hablaremos de una *invocación con éxito* del resolutor.

- $\Pi \Vdash_{solve^{\mathcal{D}}_{\mathcal{X}}} \blacksquare$, para indicar que $k = 0$. En este caso, hablaremos de una *invocación fallida* del resolutor, dando lugar a un almacén insatisfactible de la forma $\blacksquare = \blacklozenge \Box \varepsilon$.

Tal y como se ha definido previamente, un almacén de restricciones $\Pi \Box \sigma$ se dice que está en *forma resuelta* con respecto a un conjunto de variables críticas $\mathcal{X}$ (o simplemente en forma resuelta si $\mathcal{X} = \emptyset$) si y sólo si $solve^{\mathcal{D}}(\Pi, \mathcal{X}) = \Pi \Box \varepsilon$. En la práctica, las formas resueltas pueden reconocerse mediante criterios puramente sintácticos, de forma que una invocación del resolutor $solve^{\mathcal{D}}(\Pi, \mathcal{X})$ es ejecutada sólo en el caso en el que $\Pi \Box \sigma$ no esté ya en forma resuelta con respecto a $\mathcal{X}$.

Siempre que un resolutor es invocado, la condición de *corrección* requiere que ninguna solución nueva (bien tipada o no) sea introducida por el resolutor, mientras que la condición de *completitud* requiere que ninguna solución bien tipada se pierda. En la práctica, se espera que cualquier resolutor sea correcto. Sin embargo, la completitud podría satisfacerse sólo para algunas elecciones concretas del conjunto de restricciones Π que se va a resolver. Pedir la propiedad de completitud para soluciones arbitrarias (más allá de las que sean bien tipadas) podría ser un requerimiento todavía menos realista en la práctica. En el caso de resolutores que además tengan que trabajar con restricciones de igualdad y de desigualdad estricta, la completitud está sujeta a ciertas limitaciones relacionadas con la aparición de patrones opacos en los cómputos que se discutirán más adelante para cada uno de los dominios concretos que se van a considerar en este trabajo.

En el Capítulo 4 de la tesis nos interesará demostrar resultados teóricos sobre la completitud de procedimientos de resolución de objetivos para el esquema $CFLP(\mathcal{D})$, haciendo abstracción de la dificultad práctica de implementar resolutores completos. Para ello, optaremos por suponer que se dispone de un *resolutor idealmente completo* en el que la condición (6) pedida en la Definición 5 se cumple si cambiamos $WTSol_{\mathcal{D}}$ por $Sol_{\mathcal{D}}$. De esta forma, se consigue identificar claramente cuáles son las condiciones que garantizan la completitud de la semántica operacional del esquema $CFLP(\mathcal{D})$ con respecto a todas las posibles soluciones, sean estas bien tipadas o no, de manera análoga a como se realiza en los trabajos [LRV04b, Vad05]. En el resto de este capítulo seguiremos suponiendo la condición de completitud de un resolutor tal como aparece enunciada en la condición (6) más arriba.

2.4.3. Reglas de transformación de almacenes

Desde el punto de vista del usuario, un resolutor puede comportarse como una "*caja negra*" o como una "*caja transparente*". Los resolutores de "caja negra" pueden ser invocados para computar disyunciones de formas resueltas, pero los usuarios no pueden observar sus cómputos internos, en contraste con el caso de los resolutores de "caja transparente". En particular, los resolutores de "caja transparente" pueden ser definidos por el usuario mediante el uso de herramientas apropiadas, como las

denominadas *Constraint Handling Rules* [Fru98] de Früwirth. En el resto de este capítulo usaremos *reglas de transformación de almacenes* como técnica abstracta útil para especificar el comportamiento de resolutores de caja transparente.

Sistemas de transformación de almacenes

Un *sistema de transformación de almacenes* (abreviadamente *sts*, del inglés *store transformation system*) sobre un dominio de restricciones $\mathcal{D}$ se especifica mediante un conjunto de *reglas de transformación de almacenes* (abreviadamente *str*, del inglés *store transformation rule*) **RL** por medio de las cuales es posible describir los diferentes modos de transformar un almacén dado $\Pi \square \sigma$ con respecto a un conjunto dado $\mathcal{X}$ de variables críticas. Las nociones y notaciones que se definen a continuación son útiles para describir transformaciones de almacenes. Algunas de ellas hacen referencia a un conjunto seleccionado de *str*s denotado por $\mathcal{RS}$.

- $\Pi \square \sigma \vdash_{\mathcal{D},\mathcal{X}} \Pi' \square \sigma'$ indica que el almacén de restricciones $\Pi \square \sigma$ puede ser transformado en un nuevo almacén $\Pi' \square \sigma'$ en un solo paso, usando una de las *str*s disponibles. Esta notación puede también usarse para indicar un paso de transformación fallido, sin más que considerar el almacén de restricciones inconsistente $\blacksquare = \blacklozenge \square \varepsilon$ en lugar de $\Pi' \square \sigma'$.

- $\Pi \square \sigma \vdash^*_{\mathcal{D},\mathcal{X}} \Pi' \square \sigma'$ indica que el almacén $\Pi \square \sigma$ puede ser transformado en el nuevo almacén $\Pi' \square \sigma'$ en una cantidad finita de pasos.

- El almacén $\Pi \square \sigma$ se denomina $\mathcal{RS}$-*irreducible* si y sólo si no existe ninguna *str* $\mathbf{RL} \in \mathcal{RS}$ que pueda ser aplicada con el fin de transformar el almacén de restricciones $\Pi \square \sigma$. Se observa así que trivialmente esto es cierto en el caso particular en el que $\mathcal{RS}$ sea el conjunto vacío. Si $\mathcal{RS}$ es el conjunto de todas las *str*s disponibles, entonces el almacén $\Pi \square \sigma$ se denomina simplemente *irreducible* (o también una $\mathcal{X}$-*forma resuelta*).

- $\Pi \square \sigma \vdash^*_{\mathcal{D},\mathcal{X}}! \Pi' \square \sigma'$ indica que el paso de transformación $\Pi \square \sigma \vdash^*_{\mathcal{D},\mathcal{X}} \Pi' \square \sigma'$ se cumple, y además, que el almacén de restricciones final $\Pi' \square \sigma'$ es irreducible.

Las reglas de transformación de almacenes deben ser cuidadosamente especificadas. Asumamos un *sts* dado sobre un dominio de restricciones $\mathcal{D}$ tal que para cualquier conjunto finito de restricciones atómicas primitivas $\Pi \subseteq APCon_{\mathcal{D}}$, y para cualquier conjunto de variables críticas $\mathcal{X} \subseteq cvar_{\mathcal{D}}(\Pi)$, el conjunto

$$\mathcal{SF}_{\mathcal{D}}(\Pi, \mathcal{X}) = \{\Pi' \square \sigma' \mid \Pi \square \varepsilon \vdash^*_{\mathcal{D},\mathcal{X}}! \Pi' \square \sigma'\}$$

es finito. Entonces, es posible especificar el comportamiento del resolutor $solve^{\mathcal{D}}$ mediante un *sts* en el modo siguiente:

$$solve^{\mathcal{D}}(\Pi, \mathcal{X}) = \bigvee \{\ \exists\overline{Y'}.\ (\Pi' \sqcap \sigma') \mid (\Pi' \sqcap \sigma') \in \mathcal{SF}_{\mathcal{D}}(\Pi,\ \mathcal{X}),\ \overline{Y'} = var(\Pi' \sqcap \sigma') \setminus var(\Pi)\ \}$$

Una vez que el resolutor $solve^{\mathcal{D}}$ ha sido definido de este modo, la notación $\Pi \Vdash_{solve^{\mathcal{D}}_{\mathcal{X}}} \exists\overline{Y'}.\ (\Pi' \sqcap \sigma')$ tiene el significado de que $\Pi \sqcap \varepsilon \Vdash^{*}_{\mathcal{D},\mathcal{X}}! \ \Pi' \sqcap \sigma'$ y $\overline{Y'} = var(\Pi' \sqcap \sigma') \setminus var(\Pi)$. En este sentido, los símbolos $\Vdash_{solve^{\mathcal{D}}_{\mathcal{X}}}$ y $\Vdash^{*}_{\mathcal{D},\mathcal{X}}!$ no deberían prestarse a confusión, aunque tengan significados relacionados.

La siguiente definición especifica diferentes propiedades de los sistemas de transformación de almacenes que son de utilidad a la hora de comprobar si los resolutores definidos por medio de ellos satisfacen o no las condiciones que se han fijado en la Definición 5.

Definición 6 (Propiedades de los Sistemas de Transformación) *Asumamos un sistema de transformación de almacenes sts sobre $\mathcal{D}$ cuya relación de transición es $\Vdash_{\mathcal{D},\mathcal{X}}$ y un conjunto seleccionado $\mathcal{RS}$ de strs. Entonces, el sts se dice que satisface:*

(1) La propiedad de **variables locales frescas** *si $\Pi \sqcap \sigma \Vdash_{\mathcal{D},\mathcal{X}} \Pi' \sqcap \sigma'$ implica que $\Pi' \sqcap \sigma'$ es un almacén de restricciones, $\overline{Y'} = var(\Pi' \sqcap \sigma') \setminus var(\Pi \sqcap \sigma)$ son variables locales frescas y $\sigma' = \sigma\sigma_1$.*

(2) La propiedad de **vinculación segura de variables críticas** *si $\Pi \sqcap \sigma \Vdash_{\mathcal{D},\mathcal{X}} \Pi' \sqcap \sigma'$ implica que $\sigma'(X)$ es una constante para todo $X \in \mathcal{X} \cap vdom(\sigma')$.*

(3) La propiedad de **ramificación finita** *si para cualquier almacén dado $\Pi \sqcap \sigma$ hay una cantidad finita de almacenes $\Pi' \sqcap \sigma'$ tal que $\Pi \sqcap \sigma \Vdash_{\mathcal{D},\mathcal{X}} \Pi' \sqcap \sigma'$.*

(4) La propiedad de **terminación** *si no existe una secuencia infinita de la forma $\{\Pi_i \sqcap \sigma_i \mid i \in \mathbb{N}\}$ tal que $\Pi_i \sqcap \sigma_i \Vdash_{\mathcal{D},\mathcal{X}} \Pi_{i+1} \sqcap \sigma_{i+1}$ para todo $i \in \mathbb{N}$.*

(5) La propiedad de **corrección local** *si para cualquier almacén de restricciones $\Pi \sqcap \sigma$ en $\mathcal{D}$, la unión*

$$\bigcup \{\ Sol_{\mathcal{D}}(\exists\overline{Y'}.\ (\Pi' \sqcap \sigma')) \mid \Pi \sqcap \sigma \Vdash_{\mathcal{D},\mathcal{X}} \Pi' \sqcap \sigma',\ \overline{Y'} = var(\Pi' \sqcap \sigma') \setminus var(\Pi \sqcap \sigma)\ \}$$

es un subconjunto de $Sol_{\mathcal{D}}(\Pi \sqcap \sigma)$.

(6) La propiedad de **completitud local para pasos $\mathcal{RS}$-libres** *si para cualquier almacén de restricciones $\Pi \sqcap \sigma$ en $\mathcal{D}$ que sea $\mathcal{RS}$-irreducible pero que no esté en $\mathcal{X}$-forma resuelta, $WTSol_{\mathcal{D}}(\Pi \sqcap \sigma)$ es un subconjunto de la unión*

$$\bigcup \{\ WTSol_{\mathcal{D}}(\exists\overline{Y'}.\ (\Pi' \sqcap \sigma')) \mid \Pi \sqcap \sigma \Vdash_{\mathcal{D},\mathcal{X}} \Pi' \sqcap \sigma',\ \overline{Y'} = var(\Pi' \sqcap \sigma') \setminus var(\Pi \sqcap \sigma)\ \}$$

Si $\mathcal{RS}$ es un conjunto vacío (en cuyo caso todos los almacenes son trivialmente $\mathcal{RS}$-irreducibles) esta propiedad se denomina simplemente completitud local.

Asumamos ahora un resolutor $solve^{\mathcal{D}}$ que haya sido definido mediante un *sts* dado cuya relación de transición sea $\vdash_{\mathcal{D},\mathcal{X}}$ y un conjunto seleccionado $\mathcal{RS}$ de *str*s. Si el *sts* satisface la propiedad de terminación, la siguiente definición recursiva tiene sentido: Un almacén de restricciones dado $\Pi \square \sigma$ es *hereditariamente $\mathcal{RS}$-irreducible* si y sólo si $\Pi \square \sigma$ es *$\mathcal{RS}$-irreducible* y todos los almacenes $\Pi' \square \sigma'$ tales que $\Pi \square \sigma \vdash_{\mathcal{D},\mathcal{X}} \Pi' \square \sigma'$ (si es que hay alguno) son también *hereditariamente $\mathcal{RS}$-irreducibles*. Una invocación del resolutor $solve^{\mathcal{D}}(\Pi, \mathcal{X})$ se denomina *$\mathcal{RS}$-libre* si y sólo si el almacén de restricciones $\Pi \square \varepsilon$ es hereditariamente $\mathcal{RS}$-irreducible. Este concepto juega un papel importante en el siguiente lema técnico (probado en el Apéndice A), el cual puede ser aplicado a la hora de demostrar que $solve^{\mathcal{D}}$ satisface los requerimientos para resolutores especificados en la Definición 5.

Lema 4 (Resolutores Definidos Mediante Sistemas de Transformaciones) *Cualquier sistema de transformación de almacenes de restricciones en el dominio $\mathcal{D}$ que sea finitamente ramificado y terminante verifica las siguientes propiedades:*

(1) $\mathcal{SF}_{\mathcal{D}}(\Pi, \mathcal{X})$ es siempre finito, y por tanto, $solve^{\mathcal{D}}$ está bien definido y satisface trivialmente la propiedad de formas resueltas.

(2) $solve^{\mathcal{D}}$ posee la propiedad de variables locales frescas (respectivamente, la propiedad de vinculación segura) si el sistema de transformación de almacenes posee la propiedad correspondiente del mismo nombre.

(3) $solve^{\mathcal{D}}$ es correcto si el sistema de transformación de almacenes es localmente correcto.

(4) $solve^{\mathcal{D}}$ es completo para invocaciones $\mathcal{RS}$-libres si el sistema de transformación de almacenes es localmente completo para pasos $\mathcal{RS}$-libres. En el caso en el que $\mathcal{RS}$ sea vacío, esto quiere decir que $solve^{\mathcal{D}}$ es completo si el sistema de transformación de almacenes es localmente completo.

2.5. Algunas instancias de resolutores en el esquema $CFLP(\mathcal{D})$

En esta sección vamos a mostrar la existencia de resolutores adecuados para los tres dominios de restricciones $\mathcal{H}$, $\mathcal{FD}$ y $\mathcal{R}$ presentados en la Sección 2.2 y de los que haremos uso en el resto de capítulos de esta tesis para ilustrar instancias concretas del esquema $CFLP(\mathcal{D})$. Haremos especial énfasis en la presentación de un resolutor de "caja transparente" para $\mathcal{H}$ usando la técnica expuesta de sistemas de transformación de almacenes, nos remitiremos al artículo [FHSV07] como referencia que

muestra una posibilidad de especificar un resolutor similar para el dominio $\mathcal{FD}$ y mencionaremos por último los resolutores de "caja negra" para los dominios $\mathcal{R}$ y $\mathcal{FD}$ implementados en el sistema $\mathcal{TOY}$ a partir de los recursos suministrados por SICStus Prolog [SIC03].

2.5.1. Resolutores en el dominio de Herbrand $\mathcal{H}$

De acuerdo con la presentación del dominio de Herbrand $\mathcal{H}$ que se ha dado en la Subsección 2.2.1, las restricciones atómicas de este dominio tienen la forma $e_1 == e_2 \rightarrow! \; t$. Como ya se ha explicado también, las restricciones de igualdad estricta $e_1 == e_2$ y las restricciones de desigualdad estricta $e_1 /= e_2$ se utilizarán como abreviaturas de las restricciones de la forma $e_1 == e_2 \rightarrow! \; true$ y $e_1 == e_2 \rightarrow! \; false$, respectivamente.

Las variables obviamente demandadas (y de este modo también las variables críticas) para restricciones primitivas de Herbrand se computan como se ha explicado en la Sección 2.4.1. Un resolutor de Herbrand debe ser entonces capaz de resolver un conjunto dado $\Pi \subseteq APCon_{\mathcal{H}}$ de restricciones atómicas primitivas de $\mathcal{H}$ con respecto a un conjunto dado $\mathcal{X} \subseteq cvar_{\mathcal{H}}(\Pi)$ de variables críticas. Para ello, el resolutor va a proceder mediante la ejecución de varias transformaciones de almacenes de restricciones de $\mathcal{H}$, las cuales se van a encargar de realizar descomposiciones simbólicas y propagaciones de vínculos, de forma similar a las técnicas clásicas que se utilizan en los algoritmos de unificación y desunificación sintáctica presentados en trabajos tales como [LMM88, Com91], pero ahora además, teniendo en cuenta un tratamiento apropiado de las variables críticas. De manera más precisa, vamos a definir un resolutor de "caja transparente" $solve^{\mathcal{H}}$ para el dominio de restricciones de Herbrand $\mathcal{H}$ usando la técnica de transformación de almacenes explicada en la Subsección 2.4.2 mediante la aplicación de las reglas de transformación para $\mathcal{H}$-almacenes que se muestran en la Figura 2.1.

Cada una de estas reglas tiene la forma $\pi, \Pi \; \square \; \sigma \vdash_{\mathcal{H},\mathcal{X}} \Pi' \; \square \; \sigma'$ e indica la transformación de cualquier almacén de restricciones $\pi, \Pi \; \square \; \sigma$ que incluya la restricción atómica primitiva π más otras restricciones Π y donde no se presupone ningún orden secuencial. En esta situación, diremos que π es la restricción atómica seleccionada para este paso de transformación. La notación $\overline{t_m == s_m}$ en la transformación **H3** abrevia $t_1 == s_1, \ldots, t_m == s_m$.

Las transformaciones **H3** y **H7** involucran descomposiciones. Una aplicación de **H3** o de **H7** se denomina *opaca* si y sólo si h es m-opaca en el sentido explicado en la Subsección 2.1.2, en cuyo caso las nuevas restricciones resultantes de la descomposición podrían llegar a ser mal tipadas. Se observa también que una aplicación de la transformación **H13** podría perder soluciones. En este sentido, diremos que una invocación $solve^{\mathcal{H}}(\Pi, \mathcal{X})$ del resolutor del dominio de Herbrand $\mathcal{H}$ es *se-*

H1 $(t == s) \rightarrow_! R,\ \Pi \square \sigma \vdash_{\mathcal{H},\mathcal{X}} (t == s,\ \Pi)\sigma_1 \square \sigma\sigma_1$ siendo $\sigma_1 = \{R \mapsto true\}$.

H2 $(t == s) \rightarrow_! R,\ \Pi \square \sigma \vdash_{\mathcal{H},\mathcal{X}} (t /= s,\ \Pi)\sigma_1 \square \sigma\sigma_1$ siendo $\sigma_1 = \{R \mapsto false\}$.

H3 $h\,\overline{t}_m == h\,\overline{s}_m,\ \Pi \square \sigma \vdash_{\mathcal{H},\mathcal{X}} \overline{t_m == s_m},\ \Pi \square \sigma$

H4 $t == X,\ \Pi \square \sigma \vdash_{\mathcal{H},\mathcal{X}} X == t,\ \Pi \square \sigma$ si t no es una variable.

H5 $X == t,\ \Pi \square \sigma \vdash_{\mathcal{H},\mathcal{X}} tot(t),\ \Pi\sigma_1 \square \sigma\sigma_1$ si $X \notin \mathcal{X}$, $X \notin var(t)$, $X \neq t$, $\sigma_1 = \{X \mapsto t\}$ y $tot(t)$ abrevia $\bigwedge_{Y \in var(t)}(Y == Y)$.

H6 $X == t,\ \Pi \square \sigma \vdash_{\mathcal{H},\mathcal{X}} \blacksquare$ si $X \in var(t)$, $X \neq t$.

H7 $h\,\overline{t}_m /= h\,\overline{s}_m,\ \Pi \square \sigma \vdash_{\mathcal{H},\mathcal{X}} (t_i /= s_i,\ \Pi \square \sigma)$ para cada $1 \leq i \leq m$.

H8 $h\,\overline{t}_n /= h'\,\overline{s}_m,\ \Pi \square \sigma \vdash_{\mathcal{H},\mathcal{X}} \Pi \square \sigma$ si $h \neq h'$ o $n \neq m$.

H9 $t /= t,\ \Pi \square \sigma \vdash_{\mathcal{H},\mathcal{X}} \blacksquare$ si $t \in \mathcal{V}ar \cup DC \cup DF \cup SPF$.

H10 $t /= X,\ \Pi \square \sigma \vdash_{\mathcal{H},\mathcal{X}} X /= t,\ \Pi \square \sigma$ si t no es una variable.

H11 $X /= c\,\overline{t}_n, \Pi \square \sigma \vdash_{\mathcal{H},\mathcal{X}} (Z_i /= t_i, \Pi)\sigma_1 \square \sigma\sigma_1$ si $X \notin \mathcal{X}$, $c \in DC^n$ y $\mathcal{X} \cap var(c\,\overline{t}_n) \neq \emptyset$, donde $1 \leq i \leq n$ es una *elección indeterminista*, $\sigma_1 = \{X \mapsto c\,\overline{Z}_n\}$ y $\overline{Z}_n$ son variables nuevas.

H12 $X /= c\,\overline{t}_n,\ \Pi \square \sigma \vdash_{\mathcal{H},\mathcal{X}} \Pi\sigma_1 \square \sigma\sigma_1$ si $X \notin \mathcal{X}$, $c \in DC^n$ y $\mathcal{X} \cap var(c\,\overline{t}_n) \neq \emptyset$, donde $\sigma_1 = \{X \mapsto d\,\overline{Z}_m\}$, $c \in DC^n$, $d \in DC^m$, $d \neq c$, d pertenece al mismo tipo de datos que c y $\overline{Z}_m$ son variables nuevas.

H13 $X /= h\,\overline{t}_m,\ \Pi \square \sigma \vdash_{\mathcal{H},\mathcal{X}} \blacksquare$ si $X \notin \mathcal{X}$, $\mathcal{X} \cap var(h\,\overline{t}_m) \neq \emptyset$ y $h \notin DC^m$.

Figura 2.1: Reglas de transformación de almacenes para $solve^{\mathcal{H}}$

gura si y sólo si ha sido computada sin ninguna aplicación opaca de las reglas de transformación de almacenes **H3** y **H7**, y sin ninguna aplicación de la regla de transformación de almacenes **H13**. Más formalmente, diremos que $solve^{\mathcal{H}}(\Pi, \mathcal{X})$ es una invocación segura del $\mathcal{H}$-resolutor si y sólo si es $\mathcal{URS}$-libre, donde $\mathcal{URS}$ es el conjunto $\{\mathbf{OH3}, \mathbf{OH7}, \mathbf{H13}\}$ constituido por la regla **H13** y por las instancias no seguras **OH3** y **OH7** que corresponden, respectivamente, a aplicaciones opacas de las reglas **H3** y **H7**.

La idea de utilizar restricciones de igualdad y de desigualdad en Programación Lógica procede de los trabajos de Colmerauer [Col84, Col90]. El problema de resolver estas restricciones, así como también problemas de decisión relacionados que hacen uso de teorías que involucran ecuaciones e inecuaciones, ha sido ampliamente investigado en trabajos como [LMM88, Mah88, CL89, Com91, Fer92, BB94], entre otros. En ellos se asume una semántica algebraica clásica para la relación de igualdad, y se proponen métodos para resolver lo que se denominan *problemas de unificación y desunificación*, los cuales guardan bastantes analogías con las reglas de transformación que se muestran en la Figura 2.1. Sin embargo, también existen algunas diferencias importantes, debido a que la igualdad estricta en *CFLP* ha sido diseñada para poder trabajar con funciones perezosas y posiblemente indeterministas, cuyo comportamiento no se corresponde con la semántica de la igualdad en la lógica ecuacional y en el álgebra clásica, como se argumenta en [Rod01]. Una aproximación a las restricciones de desigualdad próxima a la dada en nuestro marco semántico puede consultarse en [AGL94]. Sin embargo, en esta referencia no se desarrolla ninguna formalización de un resolutor de Herbrand.

Propiedades formales del resolutor de Herbrand

El siguiente teorema afirma que el *sts* especificado en la Figura 2.1 define un resolutor correcto para el dominio $\mathcal{H}$ que además es completo para invocaciones seguras. Es importante observar que las *str*s de la Figura 2.1 no solo tienen sentido para la signatura específica del dominio $\mathcal{H}$, sino también para cualquier signatura específica que incluya la primitiva ==. Por ello, la Figura 2.1 se puede utilizar como parte de la especificación del resolutor de cualquier dominio de restricciones que disponga de la primitiva ==, como por ejemplo $\mathcal{R}$ y $\mathcal{FD}$.

Teorema 1 (Propiedades Formales de $solve^{\mathcal{H}}$**)** *El sts con relación de transición* $\Vdash_{\mathcal{H},\mathcal{X}}$ *definido en la Figura 2.1 es finitamente ramificado y terminante, y por tanto*

$$solve^{\mathcal{H}}(\Pi, \mathcal{X}) = \bigvee \{ \ \exists \overline{Y'}. \ (\Pi' \ \Box \ \sigma') \mid \Pi' \ \Box \ \sigma' \in \mathcal{SF}_{\mathcal{H}}(\Pi, \mathcal{X}), \ \overline{Y'} = var(\Pi' \ \Box \ \sigma') \setminus var(\Pi) \ \}$$

está bien definido para cualquier conjunto finito $\Pi \subseteq APCon_{\mathcal{H}}$ *y cualquier* $\mathcal{X} \subseteq cvar_{\mathcal{H}}(\Pi)$. *Más aún,* $solve^{\mathcal{H}}$ *satisface todos los requerimientos y requisitos que se enumeran para los resolutores de restricciones en la Definición 5, excepto la propiedad de* completitud, *la cual podría fallar para algunas elecciones del conjunto de restricciones* $\Pi \subseteq APCon_{\mathcal{H}}$ *que se va a resolver, y está garantizado que se cumpla sólo si la invocación del resolutor* $solve^{\mathcal{H}}(\Pi, \mathcal{X})$ *es segura (es decir, si es* $\mathcal{URS}$*-libre).*

La demostración del teorema anterior es bastante técnica y puede consultarse en el Apéndice A de esta memoria. Aquí tan sólo nos limitaremos a dar algunas indicaciones al respecto. Puesto que los primeros tres requerimientos de la Definición 5

(los que hemos denominado, respectivamente, *variables locales frescas*, *formas resueltas* y *vinculación segura de variables críticas*) son triviales, debido a que las transformaciones de almacenes introducen siempre nuevas variables frescas y cada una de las alternativas computadas por $solve^{\mathcal{H}}$ es una forma normal con respecto a estas transformaciones, vamos tan solo a dar unas pocas observaciones relativas a las propiedades de *discriminación* y de *completitud*. Estas observaciones pueden servir de ayuda para entender algunas de las diferencias entre nuestro $\mathcal{H}$-resolutor y algunos otros de los métodos clásicos para resolver problemas de unificación y desunificación que se han indicado anteriormente.

Con el fin de demostrar los requisitos pedidos para la propiedad de *discriminación*, se han de considerar los pasos de transformación de la forma $\pi, \Pi \square \sigma \vdash_{\mathcal{H},\mathcal{X}} \pi', \Pi' \square \sigma'$ basados en la selección de una restricción de desigualdad $\pi = X \not= t$, donde t no es una variable y $\mathcal{X} \cap var(t) \neq \emptyset$. Los pasos de transformación deben corresponder a alguno de los siguientes tres casos:

(1) $t \not= c\,\bar{t}_n$, donde $c \in DC^n$ es una constructora de datos n-aria, el paso ha usado la transformación **H11**, y por tanto $\sigma' = \sigma\ \{X \mapsto c\,\overline{Z}_n\}$, $\pi' = (Z_i \not= t_i)\sigma'$ y $\Pi' = \Pi\sigma'$, para alguna elección indeterminista de $1 \leq i \leq n$, siendo $\overline{Z}_n$ variables frescas. Después de este paso, cada aparición de una variable crítica perteneciente a $\mathcal{X} \cap var(c\,\bar{t}_n)$ que todavía se encuentre presente en π' ha de aparecer dentro de un patrón t_i cuyo tamaño sea estrictamente más pequeño que el de $c\,\bar{t}_n$.

(2) $t \not= c\,\bar{t}_n$, donde $c \in DC^n$ es una constructora de datos n-aria, el paso ha usado la transformación **H12**, y por tanto $\sigma' = \sigma\ \{X \mapsto d\,\overline{Z}_m\}$, $\pi' = \Diamond$, y $\Pi' = \Pi\sigma'$ para alguna elección de una constructora de datos m-aria $d \in DC^m$ perteneciente al mismo tipo de datos que la constructora c (pero diferente de c), y siendo $\overline{Z}_m$ variables frescas. Después de este paso, cada aparición de una variable crítica en la intersección $\mathcal{X} \cap var(c\,\bar{t}_n)$ no va a estar presente en π', puesto que $var(\pi') = var(\Diamond) = \emptyset$.

(3) $t \not= h\,\bar{t}_m$, donde $h \notin DC^m$ no es una constructora de datos m-aria, y el paso ha usado la transformación **H13**. En este caso, tenemos un paso de transformación fallido $\pi, \Pi \square \sigma \vdash_{\mathcal{H},\mathcal{X}} \blacksquare$, el cual conduce a devolver una disyunción vacía de alternativas en forma resuelta. En esta ocasión, se observa trivialmente que cada variable crítica perteneciente a $\mathcal{X}$ o bien desaparece o bien se convierte en una variable obviamente demandada en cada forma resuelta perteneciente a esta disyunción vacía.

Se observa así que, por una parte, las transformaciones **H11** y **H12** se han diseñado para asegurar la propiedad de *discriminación*, así como la preservación de la *completitud* con respecto a soluciones bien tipadas. Por otra parte, la transformación **H13** asegura trivialmente la discriminación, pero sacrifica la completitud debido a

que falla sin estar segura de que no existan soluciones bien tipadas. Esto correspondería a situaciones infrecuentes en la práctica y para las que no se dispone de ningún otro modo de preservar la completitud manteniendo la propiedad de *ramificación finita*. Las otras dos transformaciones de fallo dadas en la Figura 2.1 (las reglas **H6** y **H9**) sí que respetan la completitud, debido a que se aplican sobre almacenes de restricciones insatisfactibles. Finalmente, los otros dos casos en los que la completitud podría perderse corresponden a pasos de descomposición no seguros realizados con las instancias opacas **OH3** y **OH7** de las *str*s **H3** y **H7**. Debido a la propiedad de terminación de los $\mathcal{H}$-*sts*, es un problema decidible saber cuándo un $\mathcal{H}$-almacén dado $\Pi \sqcap \sigma$ es hereditariamente $\mathcal{URS}$-irreducible, en cuyo caso ninguna descomposición opaca tendrá lugar cuando se resuelva este almacén. Sin embargo, los cómputos que realizaremos en los cálculos de resolución de objetivos que se presentarán en el Capítulo 4 pueden dar lugar, en ocasiones, a $\mathcal{H}$-almacenes cuya resolución involucre pasos de descomposición opaca. Debido a los resultados teóricos que se demuestran en [GHR01], la aparición eventual de estos pasos de descomposición opaca durante la resolución de objetivos es un problema indecidible. En el caso en el que aparezcan descomposiciones opacas, éstas deberían ser señaladas de algún modo como advertencias para el usuario.

Ejemplo 3 (Comportamiento de $solve^{\mathcal{H}}$) *Con el fin de ilustrar el comportamiento del resolutor de restricciones de $\mathcal{H}$, vamos a considerar la restricción de desigualdad $L /= X : Xs$ discutida en la Subsección 2.4.1. Recordemos que la variable L es una variable obviamente demandada, mientras que las variables X y Xs son ambas críticas. Por tanto, hay cuatro posibles elecciones para el conjunto $\mathcal{X}$ de variables críticas que va a ser usado dentro de las llamadas al resolutor: $\emptyset$, $\{X\}$, $\{Xs\}$ y $\{X, Xs\}$. Vamos a discutir por separado cada uno de estos casos.*

- *Elegir $\mathcal{X} = \emptyset$ significa que al resolutor no se le pide discriminar con respeto a ninguna variable crítica. En este caso, $solve^{\mathcal{H}}(L /= X : Xs, \emptyset)$ devuelve como resultado $L /= X : Xs \sqcap \varepsilon$, mostrando así que la restricción $L /= X : Xs$ es considerada como una forma resuelta con respecto al conjunto vacío de variables críticas.*

- *Elegir $\mathcal{X} = \{X\}$ exige al resolutor discriminar con respecto a la variable crítica X. La llamada al resolutor $solve^{\mathcal{H}}(L /= X : Xs, \{X\})$ devuelve entonces una disyunción de alternativas:*

$$(\Diamond \sqcap \{L \mapsto []\}) \vee (X' /= X \sqcap \{L \mapsto X' : Xs'\}) \vee (Xs' /= Xs \sqcap \{L \mapsto X' : Xs'\})$$

cuyos miembros corresponden a los tres diferentes almacenes $(\Pi' \sqcap \sigma')$ tales que el paso $L /= X : Xs \sqcap \varepsilon \vdash_{\mathcal{H},\{X\}} \Pi' \sqcap \sigma'$ puede ser realizado con las

transformaciones **H11** *y* **H12**. *Puesto que estos almacenes están resueltos con respecto a* $\{X\}$, *ninguna otra transformación es requerida. Se observa, por tanto, que* X *no aparece en la primera y la tercera alternativa, mientras que en la segunda ésta ha llegado a ser una variable obviamente demandada. De este modo, se satisface la propiedad de discriminación requerida a los resolutores.*

- *Para cada una de las dos elecciones* $\mathcal{X} = \{Xs\}$ *y* $\mathcal{X} = \{X, Xs\}$, *es fácil comprobar que la llamada al resolutor* $solve^{\mathcal{H}}(L \mathrel{/=} X : Xs, \mathcal{X})$ *devuelve la misma disyunción de tres alternativas que en el apartado anterior, y la propiedad de discriminación para cada una de las variables pertenecientes a* $\mathcal{X}$ *se cumple también en estos dos casos con respecto al conjunto* $\mathcal{X}$ *elegido.*

Implementación del resolutor de Herbrand en el sistema $\mathcal{TOY}$

El comportamiento de las *str*s de la Figura 2.1 está empotrado en la implementación del propio sistema $\mathcal{TOY}$, del modo descrito en [AGL94]. Por ello, la funcionalidad del resolutor de Herbrand extendido a cualquier signatura que incluya la primitiva == está disponible en $\mathcal{TOY}$. Desgraciadamente, el sistema en su versión actual no genera advertencias para el usuario en el caso de que se produzcan descomposiciones opacas.

2.5.2. Resolutores en el dominio real $\mathcal{R}$

El dominio $\mathcal{R}$, tal y como se ha definido en la Subsección 2.2.2, soporta la computación con restricciones aritméticas sobre números reales. Este es uno de los dominios de restricciones que ha gozado de una mayor popularidad en el contexto de la programación con restricciones, y en especial en el contexto de la programación lógica con restricciones y el esquema *CLP* a través de su bien conocida instancia $CLP(\mathcal{R})$ [JMSY92].

Las restricciones en el dominio $\mathcal{R}$ tienen la forma general $t_1 \odot t_2 \rightarrow! \ t$, donde t_1, t_2, t son patrones y $\odot$ es una de las primitivas aritméticas o bien la primitiva de igualdad estricta ==. Además de las restricciones de igualdad y de desigualdad estricta entre términos construidos con variables, constructoras simbólicas y números reales, existen varios tipos de *restricciones de desigualdad numérica* que pueden definirse mediante abreviaturas del modo siguiente:

- $t_1 < t_2 =_{def} t_2 \leq t_1 \rightarrow! \ false$
- $t_1 \leq t_2 =_{def} t_1 \leq t_2 \rightarrow! \ true$
- $t_1 > t_2 =_{def} t_1 \leq t_2 \rightarrow! \ false$
- $t_1 \geq t_2 =_{def} t_2 \leq t_1 \rightarrow! \ true$

Para cualquier restricción atómica primitiva π de la forma $t_1 \odot t_2 \rightarrow! \ t$ donde $\odot$ es una de estas primitivas aritméticas se verifica que $odvar_{\mathcal{R}}(\pi) = \emptyset$, suponiendo que π esté bien tipada. En el caso en el que $\odot$ sea la primitiva $==$, el conjunto de variables obviamente demandadas $odvar_{\mathcal{R}}(\pi)$ se calcula del modo ya explicado anteriormente en la Subsección 2.4.1. Consecuentemente, y en relación al resolutor $solve^{\mathcal{R}}$, la implementación de este resolutor para $\mathcal{R}$ en $\mathcal{TOY}$ se compone de código empotrado en el sistema $\mathcal{TOY}$ que se hace cargo de las restricciones de la forma $t_1 == t_2 \rightarrow! \ t$ mediante *str*s como las presentadas en la Figura 2.1 y llamadas al resolutor de reales de SICStus Prolog que se encargan de resolver las restricciones correspondientes a primitivas aritméticas cuando se ha detectado que se trata de una restricción entre valores reales. El siguiente postulado puede ser entonces asumido como un resultado razonable en nuestro contexto:

Postulado 1 (Suposiciones sobre el Resolutor de $\mathcal{R}$) *El resolutor $solve^{\mathcal{R}}$ para $\mathcal{R}$ implementado en $\mathcal{TOY}$ es correcto, y además completo para llamadas que sean seguras en un doble sentido:*

(a) No dan lugar a descomposiciones opacas asociadas a la primitiva $==$.

(b) No dan lugar a restricciones aritméticas que no puedan ser resueltas por los procedimientos de resolución ofrecidos por SICStus Prolog (por ejemplo, restricciones aritméticas no lineales).

Ejemplo 4 (Comportamiento del Resolutor de $\mathcal{R}$) *Vamos ahora a ilustrar el comportamiento del resolutor $solve^{\mathcal{R}}$ considerando el siguiente conjunto de restricciones atómicas primitivas:*

$$\Pi = \{RY \geq d - 0.5,\ RY - RX \leq 0.5,\ RY + RX \leq 2 * d + 0.5\}$$

Estas tres restricciones son similares a las que ya aparecían en el programa descrito en el Ejemplo 1. La llamada al resolutor $solve^{\mathcal{R}}(\Pi)$ devuelve una sola alternativa Π' $\square$ ε con $\Pi' = \Pi \cup \{RY \leq d + 0.5\}$. En este caso, la nueva restricción $RY \leq d + 0.5$ ha sido inferida a partir de las dos restricciones $RY - RX \leq 0.5$ y $RY + RX \leq 2 * d + 0.5$. En otros casos, el resolutor de $\mathcal{R}$ puede realizar otras inferencias por medio de razonamientos aritméticos válidos en la teoría matemática de los números reales. En general, las formas resueltas que han sido computadas por los resolutores pueden ayudar a hacer más explícitos los requerimientos sobre los valores de las variables que ya estaban implícitos en las restricciones antes de su resolución (como por ejemplo, la cota superior $RY \leq d + 0.5$ en el conjunto de restricciones Π).

2.5.3. Resolutores en el dominio finito $\mathcal{FD}$

Como se definió en la Subsección 2.2.3, el dominio de restricciones $\mathcal{FD}$ permite soportar la computación con restricciones aritméticas sobre los números enteros y

restricciones de *dominio finito*. Las restricciones atómicas en $\mathcal{FD}$ incluyen así todas aquellas restricciones que son de la forma $t_1 \odot t_2 \rightarrow! \ t$, donde t_1, t_2, t son patrones y $\odot$ es una de las primitivas aritméticas o bien la primitiva de igualdad estricta $==$. Además de las restricciones de igualdad y desigualdad estricta entre términos construidos con variables, constructoras simbólicas y números enteros, encontramos las siguientes restricciones particulares:

- *Restricciones de dominio*: $domain \ t_1 \ t_2 \ t_3 \rightarrow! \ t$.
- *Restricciones de pertenencia*: $belongs \ t_1 \ t_2 \rightarrow! \ t$.
- *Restricciones de etiquetado*: $labeling \ t_1 \ t_2 \rightarrow! \ t$.

Al igual que en el dominio de los reales $\mathcal{R}$, además de las restricciones de igualdad y de desigualdad estricta, expresadas en la forma abreviada usual $t_1 == t_2$ y $t_1 /= t_2$, consideraremos también varios tipos de restricciones relacionales (precedidas siempre por el símbolo # para diferenciarlas de sus homónimas sobre los reales), las cuales pueden ser también definidas mediante abreviaturas como sigue:

- $t_1 \ \#< \ t_2 =_{def} t_2 \ \#<= t_1 \rightarrow! \ false$
- $t_1 \ \#<= t_2 =_{def} t_1 \ \#<= t_2 \rightarrow! \ true$
- $t_1 \ \#> \ t_2 =_{def} t_1 \ \#<= t_2 \rightarrow! \ false$
- $t_1 \ \#>= t_2 =_{def} t_2 \ \#<= t_1 \rightarrow! \ true$

Al igual que en el caso de $\mathcal{R}$, para cualquier restricción atómica primitiva π de $\mathcal{FD}$ de la forma $t_1 \odot t_2 \rightarrow! \ t$, donde $\odot$ es una de las primitivas aritméticas anteriores, se verifica que $odvar_{\mathcal{FD}}(\pi) = \emptyset$, suponiendo que π esté bien tipada. En el caso en el que $\odot$ sea la primitiva $==$, el conjunto de variables obviamente demandadas $odvar_{\mathcal{FD}}(\pi)$ se calcularía de un modo análogo al ya explicado en la Subsección 2.4.1. En este sentido, y en lo que tiene que ver con el resolutor $solve^{\mathcal{FD}}$, la implementación de este resolutor para $\mathcal{FD}$ en $\mathcal{TOY}(\mathcal{FD})$ se compone de código empotrado en el sistema $\mathcal{TOY}$ que se hace cargo de las restricciones de la forma $t_1 == t_2 \rightarrow! \ t$ mediante *str*s como las presentadas en la Figura 2.1 y llamadas al resolutor de dominios finitos de SICStus Prolog que se encargan de resolver las restricciones correspondientes a primitivas aritméticas cuando se ha detectado que se trata de una restricción entre valores enteros. El siguiente postulado puede ser así asumido como un resultado razonable en nuestro esquema:

Postulado 2 (Suposiciones sobre el Resolutor de $\mathcal{FD}$) *El resolutor $solve^{\mathcal{FD}}$ implementado en $\mathcal{TOY}(\mathcal{FD})$ es correcto, y además completo para llamadas que sean seguras en un doble sentido:*

(a) No dan lugar a descomposiciones opacas asociadas a la primitiva ==.

(b) No dan lugar a restricciones aritméticas que no puedan ser resueltas por los procedimientos de resolución ofrecidos por SICStus Prolog sobre $\mathcal{FD}$.

De manera particular, las restricciones de *etiquetado* se resuelven mediante una enumeración sistemática de todos los posibles valores que pueden tomar ciertas variables enteras. Es por ello que el resolutor $solve^{\mathcal{FD}}$ no es capaz de resolver una restricción de etiquetado π a menos que en el almacén de restricciones actual se incluyan restricciones de dominio o de pertenencia para todas las variables que aparezcan en π. El siguiente ejemplo muestra una situación típica de este tipo en $\mathcal{FD}$.

Ejemplo 5 (Comportamiento del Resolutor de $\mathcal{FD}$) *Con el fin de ilustrar el comportamiento del resolutor $solve^{\mathcal{FD}}$, vamos a considerar el conjunto de restricciones atómicas primitivas:*

$$\Pi = \{domain\ [X, Y]\ 0\ n,\ labeling\ [\,]\ [X, Y]\}$$

con restricciones similares a las que ya aparecían en el Ejemplo 1. La llamada al resolutor $solve^{\mathcal{FD}}(\Pi)$ debe entonces resolver la conjunción de una restricción de dominio con una restricción de etiquetado, y ambas restricciones involucran las variables enteras X y Y. El resolutor procede enumerando todos los posibles valores de ambas variables, X e Y, dentro de sus respectivos dominios (determinados en este caso por la restricción de dominio domain $[X, Y]$ 0 n) y devuelve una disyunción de $(n+1)^2$ alternativas, cada una de las cuales describe una solución particular de la forma:

$$(\Diamond\ \Box\ \{X \mapsto 0,\ Y \mapsto 0\}) \vee \cdots \vee (\Diamond\ \Box\ \{X \mapsto n,\ Y \mapsto n\})$$

En general, resolver restricciones de etiquetado puede dar lugar a enumeraciones de soluciones muy costosas, a menos que los dominios finitos de las variables enteras involucradas puedan ser acotados y podados por alguna computación precedente.

Un resolutor de "caja transparente" en el dominio $\mathcal{FD}$

Utilizando las ideas que hemos aplicado en el diseño de un resolutor de "caja transparente" en el dominio $\mathcal{H}$, es también posible especificar un resolutor similar sobre el dominio $\mathcal{FD}$. Este enfoque se desarrolla en profundidad en la publicación [FHSV07], donde se propone un sistema concreto de transformación de almacenes de restricciones para el que es posible demostrar propiedades semejantes a las que hemos probado para el resolutor de Herbrand. Las reglas de transformación de almacenes de este sistema son realmente una extensión bastante natural de las ya dadas en la Figura 2.1, aunque también incorporan reglas especiales para el tratamiento de restricciones propias del dominio $\mathcal{FD}$ (pertenencia, dominio y etiquetado) en función de un conjunto de variables críticas elegido en la resolución.

Remitimos al lector interesado a [FHSV07] para conocer aspectos más detallados sobre la formalización de cada una de estas reglas de transformación de almacenes en $\mathcal{FD}$, así como de los resultados teóricos sobre su corrección y completitud. En lo que queda de esta subsección vamos tan solo a mostrar cuál sería el efecto de aplicar tales reglas sobre un ejemplo concreto.

Ejemplo 6 *Vamos a ilustrar el comportamiento operacional de un resolutor de restricciones de dominio finito de "caja transparente" a través de una derivación particular. Consideramos un almacén inicial de restricciones $\mathcal{FD}$ de la forma*

$$\Pi = \{X == (s\ K) \rightarrow! \ R,\ A \ \#+ \ B \ \#< \ Z\}$$

para el que elegimos un conjunto de variables críticas $\mathcal{X} = \{Z\}$. Describimos a continuación los pasos de transformación que aplica el resolutor en su invocación $solve^{\mathcal{FD}}(\Pi, \mathcal{X})$, subrayando en cada paso la restricción del almacén que ha sido seleccionada. En el caso de este ejemplo:

$$\begin{array}{l} \underline{X == (s\ K) \rightarrow! \ R},\ A \ \#+ \ B < Z \ \square \ \varepsilon \Vdash_{\mathcal{FD},\{Z\}} \\ \qquad (\{X == s\ K,\ A \ \#+ \ B \ \#< \ Z\} \ \square \ \{R \mapsto true\}) \vee \\ \qquad (\{X \ /= s\ K,\ A \ \#+ \ B \ \#< \ Z\} \ \square \ \{R \mapsto false\}) \Vdash_{\mathcal{FD},\{Z\}} \end{array}$$

- $\begin{array}{l} (\{\underline{X == s\ K},\ A \ \#+ \ B \ \#< \ Z\} \ \square \ \{R \mapsto true\}) \Vdash_{\mathcal{FD},\{Z\}} \\ (\{\underline{s\ K == s\ K},\ A \ \#+ \ B \ \#< \ Z\} \ \square \ \{R \mapsto true, X \mapsto s\ K\}) \Vdash_{\mathcal{FD},\{Z\}} \\ (\{K == K,\ A \ \#+ \ B \ \#< \ Z\} \ \square \ \{R \mapsto true, X \mapsto s\ K\}) \not\Vdash_{\mathcal{FD},\{Z\}} \end{array}$

- $\begin{array}{l} (\{\underline{X \ /= s\ K},\ A \ \#+ \ B \ \#< \ Z\} \ \square \ \{R \mapsto false\}) \Vdash_{\mathcal{FD},\{Z\}} \\ \qquad (\{A \ \#+ \ B \ \#< \ Z\} \ \square \ \{R \mapsto false,\ X \mapsto 0\}) \vee \\ \qquad (\{A \ \#+ \ B \ \#< \ Z,\ M \ /= K\} \ \square \ \{R \mapsto false,\ X \mapsto s\ M\}) \not\Vdash_{\mathcal{FD},\{Z\}} \end{array}$

Por tanto, el resolutor de restricciones devuelve las siguientes formas resueltas:

$$\begin{array}{l} solve^{\mathcal{FD}}(\{X == (s\ K) \rightarrow! \ R,\ A \ \#+ \ B \ \#< \ Z\ \},\ \{Z\}) = \\ \qquad (\{A \ \#+ \ B \ \#< \ Z,\ tot(K)\} \ \square \ \{R \mapsto true,\ X \mapsto s\ K\}) \vee \\ \qquad (\{A \ \#+ \ B \ \#< \ Z\} \ \square \ \{R \mapsto false,\ X \mapsto 0\}) \vee \\ \qquad (\{A \ \#+ \ B \ \#< \ Z,\ M \ /= K\} \ \square \ \{R \mapsto false,\ X \mapsto s\ M\}) \end{array}$$

2.6. $CFLP(\mathcal{D})$-programas y objetivos

El esquema *CLP* [JL87, JMMS98] permite definir toda una familia de lenguajes de programación lógica con restricciones $CLP(\mathcal{D})$ parametrizados por un dominio de restricciones $\mathcal{D}$, de tal forma que los resultados clásicos relativos a la semántica de programas lógicos puedan ser cómodamente extendidos a todos los lenguajes

CLP($\mathcal{D}$) de un modo elegante y uniforme. Con este mismo propósito, vamos a completar en esta última sección la presentación del nuevo esquema genérico de programación *CFLP*($\mathcal{D}$) introduciendo una noción adecuada de *CFLP*($\mathcal{D}$)*-programa* y de *objetivo*, y asumiendo para ello que tanto el dominio de restricciones $\mathcal{D}$ como su correspondiente resolutor de "caja negra" o de "caja transparente", han sido definidos como se ha discutido en las secciones precedentes. Como en otros trabajos previos sobre programación lógico funcional con restricciones [Lop92, Lop94, MIS99, Mar00, MIS00, KMI01], introduciremos los programas como conjuntos de reglas de reescritura con restricciones para símbolos de función definidos por el usuario.

En lo sucesivo, asumiremos un dominio de restricciones arbitrariamente fijado $\mathcal{D}$, con universo de valores $\mathcal{U}_\mathcal{D}$ y signatura específica Σ. Consideraremos *CFLP*($\mathcal{D}$)*-programas* como conjuntos de reglas de reescritura con restricciones que permiten definir el comportamiento de funciones perezosas indeterministas (y posiblemente de orden superior) sobre $\mathcal{D}$, denominadas *reglas de programa*. De manera más precisa, una regla de programa Rl para un símbolo de función definida $f \in DF^n_\Sigma$ con tipo principal $f :: \overline{\tau}_n \to \tau$ tiene la forma

$$Rl : f\,\overline{t}_n \to r \Leftarrow P \,\square\, \Delta$$

y se le pide que satisfaga las condiciones listadas a continuación:

(1) El *lado izquierdo* $f\,\overline{t}_n$ es una expresión con una secuencia lineal de patrones totales $t_i \in Pat_\mathcal{D}$ para todo $1 \leq i \leq n$.

(2) El *lado derecho* $r \in Exp_\mathcal{D}$ es una expresión total.

(3) $\Delta \subseteq ACon_\mathcal{D}$ es una conjunción finita $\delta_1, \ldots, \delta_m$ de restricciones atómicas totales δ_i en $\mathcal{D}$ para cada $1 \leq i \leq m$, posiblemente incluyendo apariciones de símbolos de función definida.

(4) P es un conjunto finito de *producciones* de la forma $e_i \to s_i$ $(1 \leq i \leq k)$, también entendida como conjunción, y tal que las tres *condiciones de admisibilidad* siguientes se satisfacen para alguna ordenación (i.e., asignación de subíndices) de las producciones:

   a) Para todo $1 \leq i \leq k$, $e_i \in Exp_\mathcal{D}$ es una expresión total, $s_i \in Pat_\mathcal{D}$ es un patrón lineal total, y $var(s_i) \cap var(f\,\overline{t}_n) = \emptyset$.
   b) Para todo $1 \leq i \leq j \leq k$, $var(e_i) \cap var(s_j) = \emptyset$.
   c) Para todo $1 \leq i < j \leq k$, $var(s_i) \cap var(s_j) = \emptyset$.

   De manera equivalente, es posible reorganizar las producciones de P en la forma $P \equiv e_1 \to s_1, \ldots, e_k \to s_k$, donde $var(e_i) \cap var(s_j) = \emptyset$ para todo $1 \leq i \leq j \leq k$.

Adicionalmente, se requiere que las reglas de programa sean bien tipadas, es decir, que exista algún contexto de tipos Γ para las variables que aparecen en Rl que permita tipar adecuadamente la regla de programa en el siguiente sentido:

1. Para cada $1 \leq i \leq n : \Sigma,\ \Gamma \vdash_{WT} t_i :: \tau_i$,
2. $\Sigma,\ \Gamma \vdash_{WT} r :: \tau$,
3. Para cada $1 \leq i \leq m$: $\Sigma,\ \Gamma \vdash_{WT} \delta_i$.

La condición requerida de linealidad por la izquierda en el primer apartado es bastante común en programación lógica y funcional. Como ocurre en la programación lógica con restricciones, la parte condicional de una regla de programa no necesita la aparición explícita de cuantificadores existenciales, porque una regla de programa como la regla Rl que se ha descrito es lógicamente equivalente a

$$Rl' :\ f\,\bar{t}_n \to r \Leftarrow \exists \overline{Y}.\ (P \square \Delta)$$

donde $\overline{Y} = var(P \square \Delta) \setminus var(f\,\bar{t}_n \to r)$. Otra característica distinguida de nuestro lenguaje es que no se requiere ninguna propiedad de *confluencia* en los programas, y por tanto, las funciones pueden ser *indeterministas*, es decir, pueden devolver diversos valores para los mismos argumentos, incluso aunque estos sean cerrados.

Las condiciones de admisibilidad (4) a), b) y c) se pueden motivar pensando en cada una de las producciones $e_i \to s_i$ como una *definición local*, de forma que se espera que trabajen para obtener valores para las variables en el patrón s_i ajustando el resultado de evaluar e_i a s_i. La admisibilidad significa que las variables localmente definidas han de ser nuevas con respecto al lado izquierdo de la regla de programa, y también que las definiciones locales no sean recursivas. Tomar $P \square \Delta$ como parte condicional en la regla de programa significa que las producciones en P, y también las restricciones en Δ, deben tener éxito para que la regla de reescritura sea finalmente aplicable.

Una regla de programa tal que P y Δ sean ambas vacías puede ser abreviada en la forma $f\,\bar{t}_n \to r$. Observamos que, una formulación equivalente para la condición de admisibilidad (4) b) puede ser obtenida definiendo la *relación de producción* $\gg_P$ del modo siguiente: $X \gg_P Y$ si y sólo si existe algún $1 \leq i \leq k$ tal que $X \in var(e_i)$ e $Y \in var(s_i)$. De esta forma, basta con pedir que la clausura transitiva de $\gg_P^+$ sea irreflexiva, o equivalentemente, que sea un orden parcial estricto.

Como se ha mostrado en el Ejemplo 1, los *predicados* pueden verse como funciones booleanas, y las *cláusulas* pueden entenderse entonces como abreviaturas de reglas de programa de la forma $f\,\bar{t}_n \to true \Leftarrow P \square \Delta$, cuyo lado derecho es la constante booleana $true$. Además, las restricciones de la forma $e == true$ (siendo e una expresión booleana) se abrevian como e.

Como ejemplo concreto de *CFLP*($\mathcal{D}$)-programa, podemos considerar las reglas de programa presentadas en el Ejemplo 1 escritas en la sintaxis de $\mathcal{TOY}$. Otros ejemplos concretos de *CFLP*($\mathcal{D}$)-programas en los dominios $\mathcal{H}$, $\mathcal{R}$ y $\mathcal{FD}$ pueden encontrarse al final de esta sección y en el Apéndice B.3.

Un *objetivo* para un *CFLP*($\mathcal{D}$)-programa tiene la misma forma que la parte condicional de las reglas de programa, es decir, un objetivo G es de la forma $G : P \square \Delta$. Así, las *respuestas computadas* para un objetivo G se espera que sean parejas de la forma $\Pi \square \sigma$, donde

- σ es una sustitución idempotente,
- Π es un conjunto de restricciones primitivas verificando la condición de que $vdom(\sigma) \cap var(\Pi) = \emptyset$,
- para cualquier valoración η que sea solución de Π se tiene que es también una solución de $G\sigma$ con respecto al programa $\mathcal{P}$ (en símbolos, $Sol_{\mathcal{D}}(\Pi) \subseteq Sol_{\mathcal{P}}(G\sigma)$). La definición concreta de $Sol_{\mathcal{P}}(G\sigma)$ será presentada en el Capítulo 4, una vez que se haya introducido la semántica de los *CFLP*($\mathcal{D}$)-programas en el Capítulo 3.

Si bien en principio la formalización de un objetivo que hemos dado es satisfactoria para poder describir en este capítulo el tipo de cuestiones que se pueden plantear a partir de un *CFLP*($\mathcal{D}$)-programa, como se verá en capítulos posteriores, esta formalización puede resultar en ocasiones insuficiente. Dependiendo del propósito y del contexto en el que utilicemos los objetivos en el esquema *CFLP*($\mathcal{D}$), nos puede interesar representar con mayor o menor detalle la estructura de estos. Así por ejemplo, en el Capítulo 4, los objetivos han de ser capaces de reflejar cualquier estado de la computación al que se llega aplicando sucesivas reglas de transformación de objetivos, las cuales pueden actuar sobre producciones o sobre restricciones (primitivas o no), introduciendo nuevas variables locales e incluso generando vínculos de valores para las variables ya existentes. Para este propósito, resulta más adecuado distinguir separadamente la parte C de las restricciones de Δ en el objetivo que no son primitivas de la parte Π que sí que lo son. Además de la parte P de producciones, con el fin de almacenar todos aquellos vínculos de variables cuyo valor ha sido calculado hasta ese momento en la resolución del objetivo (incluidos aquellos que han sido generados por llamadas a un resolutor), resultará de utilidad hacer explícita una sustitución idempotente σ que permite recoger tales vínculos y que junto con Π represente un almacén de restricciones $S = \Pi \square \sigma$ del objetivo. Del mismo modo, interesará cuantificar existencialmente todas aquellas variables $\overline{U}$ que no figuraban inicialmente en el objetivo pero que han ido apareciendo durante el cómputo como variables intermedias. Como consecuencia, la formulación inicial de un objetivo $G : P \square \Delta$ se expresará con mayor detalle en la forma $G : \exists\overline{U}.\ (P \square C \square \Pi \square \sigma)$. Para el

caso del Capítulo 5, cuyo propósito es el de realizar la depuración de una respuesta computada incorrectamente, la formulación de los objetivos puede ser simplificada en la forma $\exists\overline{U}.\ (P \sqcap \Delta)$, eliminando así la sustitución idempotente σ (pues ésta se supondrá que ha sido aplicada a todo el objetivo) y la distinción explícita entre restricciones primitivas en Δ. Finalmente, en el Capítulo 6, el propósito de depurar respuestas perdidas hará necesario volver a considerar objetivos en la forma general $\exists\overline{U}.\ (R \sqcap S)$, donde R recoja la conjunción de P y de C, y S represente de nuevo el almacén de restricciones asociado al objetivo.

Como ejemplo concreto de programa y de objetivo, podemos considerar el siguiente $CFLP(\mathcal{R})$-programa que permite particionar, mediante la invocación de una función denominada *split*, una lista de números reales en una pareja de listas, la primera de ellas formada por todos los números positivos de la lista original y la segunda por todos los que son negativos o iguales a cero. La función *case* permite realizar esta discriminación de valores en función de las dos posibles alternativas que resultan de evaluar, para cada elemento X de la lista de partida, la restricción $X > 0 \rightarrow\!!\ R$. Usaremos pares ordenados (e_1, e_2) (los detalles referentes a tuplas son los ya explicados en la Subsección 2.1.2) y una sintaxis del estilo de Prolog para las constructoras de listas ($[\,]$ denotará la lista vacía y $[X|Xs]$ denotará una lista no vacía formada por un primer elemento X y una lista restante Xs).

$$split :: [real] \rightarrow ([real], [real])$$

$$\begin{array}{lcl} split\ \ [\,] & \rightarrow & ([\,], [\,]) \\ split\ \ [X|Xs\,] & \rightarrow & case\ R\ X\ Ys\ Zs \Leftarrow split\ Xs \rightarrow (Ys,\ Zs) \\ & & \qquad\qquad \sqcap\ X > 0 \rightarrow\!!\ R \end{array}$$

$$\begin{array}{l} case :: bool \rightarrow \alpha \rightarrow [\alpha] \rightarrow [\alpha] \rightarrow ([\alpha], [\alpha]) \\ case\ \ true\ \ X\ \ Ys\ \ Zs\ \ \rightarrow\ \ ([X|Ys],\ Zs) \\ case\ \ false\ \ X\ \ Ys\ \ Zs\ \ \rightarrow\ \ (Ys,\ [X|Zs]) \end{array}$$

Para este programa, las respuestas computadas que se esperarían para el objetivo $G : split\ [1.2,\ X,\ -0.25] == (Ys,\ Zs)$ serían las siguientes dos:

$$S_1 \sqcap \sigma_1 = \{X > 0\} \sqcap \{Ys \mapsto [1.2,\ X],\ Zs \mapsto [-0.25]\}$$
$$S_2 \sqcap \sigma_2 = \{X \leq 0\} \sqcap \{Ys \mapsto [1.2], Zs \mapsto [X,\ -0.25]\}$$

Obsérvese que, en este caso, $Sol_{\mathcal{R}}(S_1) \subseteq Sol_{\mathcal{P}}(G\sigma_1)$ da lugar a que $Sol_{\mathcal{R}}(X > 0) \subseteq Sol_{\mathcal{P}}(split\ [1.2,\ X,\ -0.25] == ([1.2,\ X],\ [-0.25]))$, que es intuitivamente cierto; y análogamente para la segunda respuesta computada.

En general, se espera que las respuestas que hayan sido computadas correctamente satisfagan el requerimiento de que $Sol_{\mathcal{D}}(S) \subseteq Sol_{\mathcal{P}}(G\sigma)$, cuyo significado depende, como ya se ha mencionado, de la *semántica declarativa* de los programas. En el Capítulo 3 proporcionaremos un marco formal para la *semántica declarativa* de los $CFLP(\mathcal{D})$-programas, dejando así para el Capítulo 4 la especificación formal de métodos de *resolución de objetivos*, los cuales se encargan de computar respuestas.

En el resto de esta sección, presentamos algunos otros ejemplos de programas concretos que nos permiten ilustrar otras características interesantes de la $CFLP(\mathcal{D})$-programación.

2.6.1. Ejemplos en el lenguaje de programación $CFLP(\mathcal{H})$

El siguiente $CFLP(\mathcal{D})$-programa puede usarse sobre el dominio de restricciones de Herbrand $\mathcal{H}$. En este programa, haremos uso de los símbolos de constructoras $0 \in DC^0$ y $s \in DC^1$ para representar simbólicamente el tipo de datos de los números naturales.

Ejemplo 7 *Listas infinitas y particionado de listas en* $CFLP(\mathcal{H})$*:*

$$data\ nat \ = \ 0 \mid s\ nat$$

$$from :: nat \rightarrow [nat]$$
$$from \quad N \quad \rightarrow \quad [N \mid from(s\ N)]$$

$$null :: [\alpha] \rightarrow nat$$
$$null \quad [\,] \quad \rightarrow \quad s\ 0$$
$$null \quad [X|Xs] \quad \rightarrow \quad 0$$

$$split :: [nat] \rightarrow ([nat], [nat])$$
$$split \quad [\,] \quad \rightarrow \quad ([\,], [\,])$$
$$split \quad [X|Xs] \quad \rightarrow \quad case\ R\ X\ Ys\ Zs \Leftarrow split\ Xs \rightarrow (Ys, Zs)$$
$$\square\ X == s\ 0 \rightarrow!\ R$$

La regla de programa para la función definida $from$ permite generar una lista potencialmente infinita de números naturales estrictamente creciente a partir de un argumento dado N. La función *null* devuelve 0 si se aplica sobre una lista no vacía y $s\ 0$ en caso contrario, y finalmente, la función principal *split* recibe como argumento una lista de números naturales representados mediante las constructoras 0 y s y

devuelve una pareja de listas, donde los miembros de la primera lista son todos los números naturales de la forma $s\ 0$, y los miembros de la segunda lista son todos los restantes números naturales. La función *case* permite de nuevo realizar la discriminación de valores en función de las dos posibles alternativas que resultan de evaluar para cada elemento X la restricción $X == s\ 0 \rightarrow\!!\ R$.

2.6.2. Ejemplos en el lenguaje de programación $CFLP(\mathcal{R})$

El siguiente ejemplo ilustra el uso de estructuras de datos infinitas y el uso de facilidades de programación de orden superior en $CFLP(\mathcal{R})$:

Ejemplo 8 *Programación de orden superior en* $CFLP(\mathcal{R})$*:*

$$
\begin{array}{l}
from :: real \rightarrow real \rightarrow [real] \\
from \quad X \quad D \quad \rightarrow \quad [\ X \mid from\ (X + D)\ D\ ]
\end{array}
$$

$$
\begin{array}{l}
takeWhile :: (\alpha \rightarrow bool) \rightarrow [\alpha] \rightarrow [\alpha] \\
takeWhile \quad P \quad [\,] \qquad \rightarrow \quad [\,] \\
takeWhile \quad P \quad [X|\mathrm{Xs}] \quad \rightarrow \quad if\ (P\,X)\ [\ X \mid takeWhile\ P\ \mathrm{Xs}\ ]\ [\,]
\end{array}
$$

$$
\begin{array}{l}
if :: bool \rightarrow \alpha \rightarrow \alpha \rightarrow \alpha \\
if \quad true \quad L \quad R \quad \rightarrow \quad L \\
if \quad false \quad L \quad R \quad \rightarrow \quad R
\end{array}
$$

$$
\begin{array}{l}
inInterval :: real \rightarrow real \rightarrow real \rightarrow bool \\
inInterval \quad A \quad B \quad X \quad \rightarrow \quad and \quad (A \leq X) \quad (X \leq B)
\end{array}
$$

$$
\begin{array}{l}
and :: bool \rightarrow bool \rightarrow bool \\
and \quad true \quad X \quad \rightarrow \quad X \\
and \quad false \quad X \quad \rightarrow \quad false
\end{array}
$$

En este ejemplo, la función $from$ permite generar una lista infinita de números reales cuyos miembros forman una secuencia aritmética, y la función de orden superior $takeWhile$ puede utilizarse para computar un prefijo de una lista dada, hasta el punto en el que algún elemento falla a la hora de satisfacer una propiedad también dada. La función $inInterval$ se define por medio de las primitivas de comparación y puede ser usada para construir restricciones definidas determinando las pertenencias entre números e intervalos.

Un posible objetivo para un programa que incluya las funciones definidas en el Ejemplo 8 sería el siguiente:

$$takeWhile\ (inInterval\ 0\ 1)\ (from\ X\ 0.5) == [X]$$

Se observa así el uso del patrón $(inInterval\ 0\ 1)$ dentro del objetivo para expresar un valor funcional, en este caso la función booleana que decide la pertenencia en el intervalo de números reales entre 0 y 1. La respuesta computada que se esperaría para este objetivo sería la siguiente:

$$\Pi \square \sigma = \{\ X > 0.5,\ X \leq 1\ \} \square \{\ \}$$

la cual puede ser escrita de manera más simple como el conjunto de restricciones $\{X > 0.5,\ X \leq 1\}$. La computación de esta respuesta de $CFLP(\mathcal{R})$ en un sistema práctico como $\mathcal{TOY}$ requiere la combinación de evaluación perezosa y resolución de restricciones, como se explicará detalladamente en el Capítulo 4.

2.6.3. Ejemplos en el lenguaje de programación $CFLP(\mathcal{FD})$

El siguiente y último ejemplo ilustra varios $CFLP(\mathcal{FD})$-programas cuyas reglas con restricciones pueden ser utilizadas en la combinación de la evaluación perezosa con listas infinitas y la resolución de restricciones de dominio finito en $CFLP(\mathcal{FD})$.

Ejemplo 9 *Evaluación perezosa de listas infinitas en* $CFLP(\mathcal{FD})$*:*

$$\begin{array}{l}
take :: int \to [\alpha] \to [\alpha] \\
take\ \ N\ \ Xs\ \ \to\ \ [\,]\ \ \Leftarrow\ \ N\ \#<=\ 0 \\
take\ \ N\ \ [\,]\ \ \to\ \ [\,]\ \ \Leftarrow\ \ N\ \#>\ \ 0 \\
take\ \ N\ \ [X \mid Xs]\ \ \to\ \ [X \mid take\ (N\ \#-\ 1)\ Xs\ ]\ \ \Leftarrow\ \ N\ \#>\ \ 0
\end{array}$$

$$\begin{array}{l}
check_list :: [int] \to int \\
check_list\ [\,]\ \ = 0 \\
check_list\ [X \mid Xs]\ \ = 1\ \ \Leftarrow domain\ [X]\ 1\ 2 \\
check_list\ [X \mid Xs]\ \ = 2\ \ \Leftarrow domain\ [X]\ 3\ 4 \\
check_list\ [X \mid Xs]\ \ = 4\ \ \Leftarrow domain\ [X]\ 5\ 7
\end{array}$$

$$\begin{array}{l}
generateFD :: int \to [int] \\
generateFD\ N\ \ \to\ \ [\ ]\ \ \Leftarrow N\ \#<=\ 0 \\
generateFD\ N\ \ \to\ \ [X \mid generateFD\ N]\ \ \Leftarrow \\
\qquad\qquad N\ \#>\ 0,\ belongs\ X\ (interval\ 0\ (N\ \#-\ 1))
\end{array}$$

$$interval :: int \to int \to [int]$$
$$interval\ L\ U \quad \to \quad if\ R\ [\ L \mid interval\ (L\ \#+\ 1)\ U\ ]\ [\,] \quad \Leftarrow \quad L\ \#<=\ U \to!\ R$$

$$from :: int \to [int]$$
$$from\ N \quad = \quad [\ N \mid from\ (N\ \#+\ 1)\ ]$$

Un posible objetivo para un programa que incluya las funciones definidas en este ejemplo, constituido por una sola igualdad estricta definida por el usuario en el dominio $\mathcal{FD}$, sería el siguiente:

$$take\ 3\ (generateFD\ 10) == List$$

Este objetivo pregunta por los tres primeros elementos de una lista infinita de enteros, generada bajo la restricción de que cada uno de sus elementos sea un valor entero perteneciente al intervalo de números enteros entre 0 y 9. Esto explica la aparición de vínculos de variable y de las restricciones de dominio en la siguiente respuesta computada para este objetivo:

$$\Pi \square \sigma = \{\ domain\ [X_1, X_2, X_3]\ 0\ 9\ \} \square \{\ List \mapsto [X_1, X_2, X_3]\ \}$$

Análogamente, $domain\ [M]\ 1\ 2$ y $domain\ [M]\ 3\ 4$ son también respuestas computadas para el objetivo admisible $check\_list\ (from\ M)\ \#< 3$, ambas acompañadas implícitamente de una sustitución vacía de vínculos de variables.

La computación de todas estas respuestas en el esquema genérico de programación $CFLP(\mathcal{D})$ requiere una combinación de evaluación perezosa y de resolución de restricciones que será detallada en el Capítulo 4 a través de la utilización de cálculos de estrechamiento perezoso con restricciones sobre un dominio $\mathcal{D}$ paramétricamente dado. Otros ejemplos más prácticos y aplicados de $CFLP(\mathcal{D})$-programación en el sistema $\mathcal{TOY}$ [ALR07] serán comentados en el Apéndice B de esta tesis. También se pueden consultar las publicaciones [LRV07] y [FHSV07], en las que se hace uso de la sintaxis y de las facilidades concretas que ofrece este sistema lógico funcional con restricciones sobre los dominios $\mathcal{H}$, $\mathcal{R}$ y $\mathcal{FD}$ presentados en este capítulo.

Capítulo 3

Una lógica para la reescritura en el esquema $CFLP(\mathcal{D})$

Presentamos en este capítulo una semántica declarativa para $CFLP(\mathcal{D})$-programas basada en el concepto de *c-interpretación* sobre un dominio de restricciones $\mathcal{D}$. Usaremos esta clase de interpretaciones para definir dos clases de semánticas de modelos en el esquema $CFLP(\mathcal{D})$ denominadas, respectivamente, *semántica débil* y *semántica fuerte*. En el Capítulo 4 usaremos estas semánticas para demostrar las propiedades de corrección y de completitud de varios procedimientos de resolución de objetivos que se han desarrollado en nuestro esquema. En concreto, mediante la semántica fuerte es posible proporcionar una caracterización del concepto de respuesta válida para todos aquellos objetivos que se plantean sobre la base de un $CFLP(\mathcal{D})$-programa, incluyendo las respuestas computadas como un caso particular. Demostraremos la existencia de un *modelo mínimo* para cada una de estas dos semánticas y lo caracterizaremos como el menor *punto fijo* calculado mediante la aplicación de un operador de transformación sobre c-interpretaciones, aprovechando la estructura de retículo de la familia de todas las c-interpretaciones sobre el dominio de restricciones $\mathcal{D}$. Asimismo, investigaremos la relación que existe entre los dos modelos mínimos obtenidos.

Por último, desarrollaremos una *lógica para la reescritura con restricciones* denominada $CRWL(\mathcal{D})$, la cual puede ser definida sobre un dominio de restricciones arbitrario $\mathcal{D}$ paramétricamente dado. Presentaremos esta lógica como una extensión natural de la lógica para la reescritura *CRWL* utilizada como marco teórico en lenguajes de programación lógico funcionales basados en funciones perezosas y posiblemente indeterministas (véase la panorámica presentada en [Rod01]). El propósito general de $CRWL(\mathcal{D})$ será el de proporcionar un marco lógico para la programación en el esquema $CFLP(\mathcal{D})$, y en consecuencia, una forma alternativa de caracterizar la semántica declarativa de los programas de una forma afín a como se ha realizado en trabajos anteriores sobre programación lógico funcional sin restricciones. Presentaremos un cálculo lógico para $CRWL(\mathcal{D})$, investigaremos sus principales propiedades

teóricas, y estudiaremos la relación que existe entre la derivación formal en este cálculo de prueba y las dos semánticas de modelos presentadas.

3.1. Interpretaciones y modelos para $CFLP(\mathcal{D})$-programas

El desarrollo de una semántica para $CFLP(\mathcal{D})$-programas podría realizarse utilizando como base el concepto de $\mathcal{D}$-*algebra* presentado en la Subsección 3.1.1. Esta aproximación resultaría análoga a la ya utilizada en la semántica tradicional de $CLP(\mathcal{D})$-programas mediante el uso del concepto de $\mathcal{D}$-*interpretación* [JMMS98]. Es más, sería formalmente muy similar a las estructuras que han sido utilizadas como interpretaciones de programas lógico funcionales en esquemas *CFLP* previos, como por ejemplo [Lop92, Lop94], y también en otros trabajos previos sobre la lógica para la reescritura *CRWL* [GHLR99, GHR01, AR01].

Sin embargo, hemos optado por reemplazar esta aproximación clásica basada en $\mathcal{D}$-álgebras por una aproximación más expresiva, la cuál va a estar motivada por el concepto de π-*interpretación* para $CLP(\mathcal{D})$-programas propuesto en los trabajos [GL91, GDL95]. De una manera informal, en el contexto *CLP* una π-interpretación es un conjunto de hechos de la forma $p\,\bar{t}_n \Leftarrow \Pi$, cuyo significado es el de que el átomo $p\,\bar{t}_n$ definido por el usuario sea válido en cualquier valoración que sea a su vez solución de las restricciones primitivas que hay en Π. Como se muestra en [GL91, GDL95], las π-interpretaciones pueden usarse como una base útil para fundamentar tres clases diferentes S_i $(i = 1, 2, 3)$ de semánticas de programas, mediante las cuales es posible caracterizar, respectivamente, *objetivos cerrados válidos*, *respuestas válidas para objetivos* y *respuestas computadas para objetivos*. De hecho, las semánticas S_i pueden verse como la contrapartida en *CLP* de semánticas previamente desarrolladas para la programación lógica, denominadas, respectivamente, *semántica de modelo mínimo cerrado de Herbrand* [Apt90, Llo87a], *semántica de modelo abierto de Herbrand* (también conocida como *C-semántica* [Cla79, FLMP93]), y por último, *S-semántica* [FLMP89, BGLM94]. Una panorámica suficientemente concisa y clarificadora de estas tres semánticas puede consultarse en [AG94].

Con el fin de generalizar las π-interpretaciones a los lenguajes $CFLP(\mathcal{D})$ descritos en el capítulo anterior, introducimos las *c-interpretaciones* en la Subsección 3.1.2, definiéndolas como conjuntos de hechos de la forma $f\,\bar{t}_n \to t \Leftarrow \Pi$, cuya intención es la de describir el comportamiento de funciones definidas por el usuario $f \in DF^n$. En el resto del capítulo usaremos esta clase de interpretaciones sobre un dominio de restricciones $\mathcal{D}$ dado para definir dos clases diferentes de semánticas en nuestro esquema $CFLP(\mathcal{D})$, las cuales se corresponderán, respectivamente, con S_1 y S_2, y a las cuales llamaremos, respectivamente, *semántica débil* y *semántica fuerte*. La semántica fuerte proporcionará una caracterización de respuestas válidas para los objetivos, incluyendo las respuestas computadas como un caso particular.

Como se verá a lo largo de este capítulo y también en los siguientes, las semánticas débil y fuerte tienen una caracterización declarativa muy natural, y son una herramienta suficientemente útil para poder demostrar la corrección y la completitud de las semánticas operacionales que se presentan en el Capítulo 4.

El planteamiento en nuestro esquema $CFLP(\mathcal{D})$ de un análogo de la semántica S_3 podría dar como resultado una caracterización de *exactamente* las respuestas computadas por la semántica operacional. En el contexto *CLP* de la programación lógica con restricciones, existe un acuerdo amplio sobre la utilización de una única formalización de esta semántica operacional, lo que da lugar a una clase bien definida de respuestas computadas [JMMS98]. Sin embargo, en el contexto *CFLP*, se ha propuesto una amplia variedad de *estrategias de estrechamiento* como base para poder describir la semántica operacional, dando así lugar a diferentes clases de respuestas computadas. La investigación de una semántica de modelos estilo S_3 se tendría que plantear eligiendo previamente una semántica operacional concreta. En esta tesis se ha optado por investigar solamente semánticas declarativas cuya formulación no dependa de detalles operacionales.

3.1.1. Álgebras sobre un dominio de restricciones

Con el fin de poder interpretar $CFLP(\mathcal{D})$-programas, necesitamos extender el dominio de restricciones $\mathcal{D}$ mediante interpretaciones concretas de los símbolos de función definida. Como primera aproximación a este objetivo, vamos a comenzar definiendo el concepto de $\mathcal{D}$-*algebra*:

Definición 7 ($\mathcal{D}$-álgebras) *Asumamos un dominio de restricciones $\mathcal{D}$ con universo de valores $\mathcal{U}_\mathcal{D}$. Una $\mathcal{D}$-algebra es cualquier estructura de la forma*

$$\mathcal{A} = \langle \mathcal{D}, \{f^\mathcal{A} \mid f \in DF\} \rangle$$

que extiende conservativamente $\mathcal{D}$ con una interpretación $f^\mathcal{A}$ para cada símbolo de función definida $f \in DF^n$, la cual debe satisfacer los dos siguientes requisitos:

(1) $f^\mathcal{A} \subseteq \mathcal{U}_\mathcal{D}^n \times \mathcal{U}_\mathcal{D}$, que da lugar a $f^\mathcal{A} \subseteq \mathcal{U}_\mathcal{D}$ en el caso particular en el que $n = 0$. La notación $f^\mathcal{A}\,\overline{t}_n \to t$ indica que $(\overline{t}_n, t) \in f^\mathcal{A}$. En el caso en el que $n = 0$, esta notación da lugar a $f^\mathcal{A} \to t$.

(2) $f^\mathcal{A}$ se comporta de forma monótona en sus argumentos y de forma antimonótona en su resultado; es decir, siempre que $f^\mathcal{A}\,\overline{t}_n \to t$, $\overline{t}_n \sqsubseteq \overline{t'}_n$ y $t \sqsupseteq t'$ se tiene que $f^\mathcal{A}\,\overline{t'}_n \to t'$.

De manera similar a como se comenta en la Definición 1 de dominio de restricciones, las condiciones de *polaridad* que se dan en el apartado (2) se emplean para capturar el comportamiento de una posible función indeterminista sobre valores finitos. Sin embargo, la condición de *radicalidad* dada en la Definición 1 se omite aquí, debido

a que las funciones definidas por el usuario que devuelven estructuras de datos potencialmente infinitas como resultado son de utilidad para la programación, y son obviamente no radicales.

3.1.2. c-interpretaciones

Nuestro siguiente paso consiste en definir un análogo para $CFLP(\mathcal{D})$-programas del concepto de π-interpretación. Para este propósito, necesitamos algunas notaciones preliminares.

Definición 8 (Sentencias con Restricciones y $\mathcal{D}$-implicación) *Sea $\mathcal{D}$ un dominio de restricciones arbitrariamente fijado. En lo que sigue, asumiremos patrones parciales $t, t_i \in Pat_{\mathcal{D}}$, expresiones parciales $e, e_i \in Exp_{\mathcal{D}}$, y un conjunto finito $\Pi \subseteq APCon_{\mathcal{D}}$ de restricciones atómicas primitivas que sea admisible (en el sentido precisado en el apartado (4) de la Definición 4 en la Subsección 2.3.2).*

(1) Consideramos tres tipos posibles de sentencias con restricciones *(abreviadamente,* c-sentencias*):*

(a) c-producciones $e \to t \Leftarrow \Pi$, *con $e \in Exp_{\mathcal{D}}$. En el caso en el que Π sea vacío darían lugar a producciones sin restricciones escritas en la forma $e \to t$. Una c-producción se denomina* trivial *si y sólo si $t = \bot$ o $Insat_{\mathcal{D}}(\Pi)$.*

(b) c-hechos $f\,\overline{t}_n \to t \Leftarrow \Pi$, *con $f \in DF^n$. Son un caso particular de c-producciones. En el caso en el que Π sea vacío dan lugar a hechos sin restricciones, escritos en la forma $f\,\overline{t}_n \to t$. Un c-hecho se denomina* trivial *si y sólo si $t = \bot$ o $Insat_{\mathcal{D}}(\Pi)$.*

(c) c-átomos $p\,\overline{e}_n \to\!! t \Leftarrow \Pi$, *con $p \in PF^n$ y t total. En el caso en el que Π sea vacío, dan lugar a átomos sin restricciones escritos en la forma $p\,\overline{e}_n \to\!! t$. Un c-átomo se denomina* trivial *si y sólo si $Insat_{\mathcal{D}}(\Pi)$.*

En lo sucesivo, usaremos φ y otros símbolos similares para denotar cualquier c-sentencia de la forma $e \to^? t \Leftarrow \Pi$, donde el símbolo $\to^?$ ha de entenderse como $\to\!!$ en el caso en el que φ sea un átomo; en otro caso, $\to^?$ ha de ser entendido como $\to$.

(2) Dadas dos c-sentencias φ y φ', decimos que φ $\mathcal{D}$-implica *φ' (en símbolos, $\varphi \succcurlyeq_{\mathcal{D}} \varphi'$) si y sólo si se cumple uno de los dos siguientes casos:*

(a) $\varphi = e \to t \Leftarrow \Pi$, $\varphi' = e' \to t' \Leftarrow \Pi'$, y existe algún $\sigma \in Sub_{\mathcal{D}}$ tal que $\Pi' \models_{\mathcal{D}} \Pi\sigma$, $\Pi' \models_{\mathcal{D}} e' \sqsupseteq e\sigma$ y $\Pi' \models_{\mathcal{D}} t' \sqsubseteq t\sigma$.

(b) $\varphi = p\,\overline{e}_n \to\!! t \Leftarrow \Pi$, $\varphi' = p\,\overline{e'}_n \to\!! t' \Leftarrow \Pi'$, y existe algún $\sigma \in Sub_{\mathcal{D}}$ tal que $\Pi' \models_{\mathcal{D}} \Pi\sigma$, $\Pi' \models_{\mathcal{D}} p\,\overline{e'}_n \sqsupseteq (p\,\overline{e}_n)\sigma$ y $\Pi' \models_{\mathcal{D}} t' \sqsupseteq t\sigma$.

La petición de que el conjunto Π utilizado en las c-sentencias sea *admisible* resulta natural debido a que los conjuntos de restricciones de los que haremos uso en adelante estarán en forma resuelta y habrán sido definidos sobre dominios de interés práctico como los presentados en el Capítulo 2, los cuales tienen siempre esta propiedad. Por este motivo, en el resto de la tesis se dará por supuesto implícitamente que los conjuntos Π manejados son siempre admisibles. Por otra parte, la idea intuitiva que hay detrás del concepto de $\mathcal{D}$-*implicación* es la de que, siempre que $\varphi \succcurlyeq_{\mathcal{D}} \varphi'$, la c-sentencia φ' puede aceptarse como una consecuencia de φ para cualquier posible interpretación de los símbolos de función definida. Esto es razonable debido a que la definición de la relación de $\mathcal{D}$-implicación depende solo de suposiciones relativas al comportamiento monótono tanto de funciones definidas como de funciones primitivas, y también del comportamiento radical de las funciones primitivas.

La siguiente definición generaliza la idea de π-interpretación dada en [GL91, GDL95] en el esquema $CFLP(\mathcal{D})$:

Definición 9 (c-interpretación) *Para cualquier dominio de restricciones $\mathcal{D}$:*

(1) Una c-interpretación *sobre $\mathcal{D}$ es cualquier conjunto $\mathcal{I}$ de c-hechos que incluya a todos los c-hechos triviales y que sea cerrado bajo $\mathcal{D}$-implicación. De manera equivalente, una c-interpretación es cualquier conjunto $\mathcal{I}$ de c-hechos tal que $cl_{\mathcal{D}}(\mathcal{I}) \subseteq \mathcal{I}$, donde $cl_{\mathcal{D}}(\mathcal{I})$ se define como sigue:*

$$cl_{\mathcal{D}}(\mathcal{I}) = \{\ \varphi' \mid \varphi' \text{ es un c-hecho trivial o existe } \varphi \in \mathcal{I} \text{ tal que } (\varphi \succcurlyeq_{\mathcal{D}} \varphi')\ \}$$

(2) El núcleo básico *de una c-interpretación $\mathcal{I}$ se define como*

$$gk_{\mathcal{D}}(\mathcal{I}) = \{\ \varphi \in \mathcal{I} \mid \varphi \text{ es un c-hecho cerrado}\ \}$$

(3) El $\mathcal{D}$-cierre *de una c-interpretación $\mathcal{I}$ se define como la clausura de su núcleo básico con respecto a los c-hechos triviales y a la $\mathcal{D}$-implicación, es decir:*

$$[\mathcal{I}]_{\mathcal{D}} = cl_{\mathcal{D}}(gk_{\mathcal{D}}(\mathcal{I}))$$

De la definición anterior se desprende que el núcleo básico $gk_{\mathcal{D}}(\mathcal{I})$ no es técnicamente una c-interpretación, aunque represente la información básica dada por $\mathcal{I}$. En su lugar, el cierre $[\mathcal{I}]_{\mathcal{D}}$ sí que es la menor c-interpretación que incluye la información básica dada por $\mathcal{I}$. Claramente, dos c-interpretaciones diferentes pueden tener el mismo núcleo, y de este modo, el mismo cierre.

Se puede demostrar fácilmente que el conjunto de los c-hechos cerrados de la forma $f\bar{t}_n \to t$ pertenecientes a una c-interpretación dada $\mathcal{I}$ determinan una $\mathcal{D}$-álgebra. Gracias a este hecho, es posible motivar nuestra afirmación de que las c-interpretaciones generalizan a las $\mathcal{D}$-álgebras y que, por tanto, estas últimas pueden ser abandonadas en favor de las primeras. En el resto de este capítulo trabajaremos sólo con c-interpretaciones.

3.1.3. Un cálculo semántico sobre c-interpretaciones

La siguiente definición asume un dominio de restricciones $\mathcal{D}$ con universo de valores $\mathcal{U}_\mathcal{D}$ y familia de valores básicos $\mathcal{B}^\mathcal{D}$, y una c-interpretación dada $\mathcal{I}$ sobre $\mathcal{D}$. El propósito del siguiente *cálculo semántico* es el de inferir la validez semántica de una c-sentencia arbitraria en $\mathcal{I}$.

Definición 10 (Cálculo Semántico sobre c-interpretaciones) *En esta definición, escribimos $\mathcal{I} \Vdash_\mathcal{D} \varphi$ para indicar que la c-sentencia φ puede ser derivada a partir de la c-interpretación $\mathcal{I}$ usando las siguientes reglas de inferencia:*

TI *Inferencia Trivial:*

$$\frac{}{\varphi}$$

Si φ es una c-sentencia trivial.

RR *Reflexividad Restringida:*

$$\frac{}{t \to t \Leftarrow \Pi}$$

Si $t \in \mathcal{B}^\mathcal{D} \cup \mathcal{V}ar$.

SP *Producción Simple:*

$$\frac{}{s \to t \Leftarrow \Pi}$$

Si $s \in Pat_\mathcal{D}$, $s \in \mathcal{V}ar$ o bien $t \in \mathcal{V}ar$, y $\Pi \models_\mathcal{D} s \sqsupseteq t$.

DC *Descomposición:*

$$\frac{e_1 \to t_1 \Leftarrow \Pi, \cdots, e_m \to t_m \Leftarrow \Pi}{h\,\overline{e}_m \to h\,\overline{t}_m \Leftarrow \Pi}$$

Si $h\,\overline{e}_m$ *es pasivo.*

IR *Reducción Interna:*

$$\frac{e_1 \to t_1 \Leftarrow \Pi, \cdots, e_m \to t_m \Leftarrow \Pi}{h\,\overline{e}_m \to X \Leftarrow \Pi}$$

Si $h\,\overline{e}_m$ *es pasivo pero no un patrón,* $X \in \mathcal{V}ar$ *y* $\Pi \models_{\mathcal{D}} h\,\overline{t}_m \sqsupseteq X$.

DF$_{\mathcal{I}}$ $\mathcal{I}$*-Función Definida:*

$$\frac{e_1 \to t_1 \Leftarrow \Pi, \cdots, e_n \to t_n \Leftarrow \Pi}{f\,\overline{e}_n \to t \Leftarrow \Pi}$$

Si $f \in DF^n$ *y* $(f\,\overline{t}_n \to t \Leftarrow \Pi) \in \mathcal{I}$.

$$\frac{e_1 \to t_1 \Leftarrow \Pi, \cdots, e_n \to t_n \Leftarrow \Pi, s\,\overline{a}_k \to t \Leftarrow \Pi}{f\,\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi}$$

Si $f \in DF^n$, $k > 0$, $(f\,\overline{t}_n \to s \Leftarrow \Pi) \in \mathcal{I}$ *y* $s \in Pat_{\mathcal{D}}$.

PF *Función Primitiva:*

$$\frac{e_1 \to t_1 \Leftarrow \Pi, \cdots, e_n \to t_n \Leftarrow \Pi}{p\,\overline{e}_n \to t \Leftarrow \Pi}$$

Si $p \in PF^n$, $t_i \in Pat_{\mathcal{D}}$ *para cada* $1 \leq i \leq n$ *y* $\Pi \models_{\mathcal{D}} p\,\overline{t}_n \to t$.

AC *Restricción Atómica:*

$$\frac{e_1 \to t_1 \Leftarrow \Pi, \cdots, e_n \to t_n \Leftarrow \Pi}{p\,\overline{e}_n \to! \; t \Leftarrow \Pi}$$

Si $p \in PF^n$, $t_i \in Pat_{\mathcal{D}}$ *para cada* $1 \leq i \leq n$ *y* $\Pi \models_{\mathcal{D}} p\,\overline{t}_n \to! \; t$.

Por convenio, ninguna regla de inferencia del cálculo semántico será aplicada en el caso de que alguna regla previa en el orden textual pueda ser aplicada. En particular, ninguna regla, excepto **TI**, puede ser usada para inferir una c-sentencia trivial, y la regla **SP** no es aplicable si **RR** es aplicable.

Árboles de prueba en el cálculo semántico

Cualquier derivación en el cálculo semántico puede ser representada como un *árbol de prueba* cuyos nodos están etiquetados mediante c-sentencias, y donde cada nodo ha sido inferido a partir de sus hijos por medio de las reglas de inferencia. En lo sucesivo, haremos uso de las siguientes notaciones:

1) Para cada regla $\mathbf{RL} \in \{\mathbf{TI}, \mathbf{RR}, \mathbf{SP}, \ldots, \mathbf{PF}, \mathbf{AC}\}$, escribiremos $T = \mathbf{RL}(\varphi, [T_1, \ldots, T_p])$ para representar un árbol de prueba cuya c-sentencia raíz φ se infiera mediante la regla de inferencia etiquetada **RL**, tomando como premisas las p c-sentencias previamente derivadas con árboles de prueba T_i $(1 \leq i \leq p)$.

2) $T : \mathcal{I} \Vdash_{\mathcal{D}} \varphi$ indica que $\mathcal{I} \Vdash_{\mathcal{D}} \varphi$ tiene como *testigo* el árbol de prueba T.

3) T se denomina un *árbol de prueba simple* si y sólo si T no hace uso de las reglas de inferencia $\mathbf{DF}_{\mathcal{I}}$, **PF** y **AC**.

4) $\|T\|$ denota el *tamaño total* del árbol de prueba T, definido como el número total de nodos que hay en T.

5) $|\ T\ |$ denota el *tamaño restringido* del árbol de prueba T, definido como el número de nodos que hay en T que son inferidos mediante alguna de las siguientes reglas: $\mathbf{DF}_{\mathcal{I}}$, **PF** o **AC**. Obviamente, $|\ T\ | \leq \|T\|$ y $|\ T\ | = 0$ si y sólo si T es un árbol de prueba simple.

Propiedades del cálculo semántico

El siguiente lema recoge todas aquellas propiedades del cálculo semántico que serán de utilidad tanto en este capítulo como en los siguientes. La demostración es bastante técnica y puede ser consultada en el Apéndice A.

Lema 5 (Propiedades del Cálculo Semántico) *El cálculo semántico sobre c-interpretaciones satisface las siguientes propiedades:*

(1) **Propiedad de Compacidad***: $\mathcal{I} \Vdash_{\mathcal{D}} \varphi$ implica que $cl_{\mathcal{D}}(\mathcal{I}_0) \Vdash_{\mathcal{D}} \varphi$ para algún subconjunto finito $\mathcal{I}_0 \subseteq \mathcal{I}$.*

(2) **Propiedad de Extensión***: $\mathcal{I} \Vdash_{\mathcal{D}} \varphi$ e $\mathcal{I} \subseteq \mathcal{I}'$ implican que $\mathcal{I}' \Vdash_{\mathcal{D}} \varphi$.*

(3) **Propiedad de Aproximación***: Para cualquier $e \in Exp_{\mathcal{D}}$ y $t \in Pat_{\mathcal{D}}$, se verifica que $\Pi \models_{\mathcal{D}} e \sqsupseteq t$ si y sólo si existe algún árbol de prueba simple T tal que $T :\Vdash_{\mathcal{D}} e \rightarrow t \Leftarrow \Pi$ (entendido como una derivación a partir de la c-interpretación trivial $\perp\!\!\!\perp = cl_{\mathcal{D}}(\emptyset)$).*

(4) **c-Átomos Primitivos***: Para cualquier átomo primitivo $p\,\bar{t}_n \rightarrow\!! t$, se verifica que $\mathcal{I} \Vdash_{\mathcal{D}} p\,\bar{t}_n \rightarrow\!! t \Leftarrow \Pi$ si y sólo si $\Pi \models_{\mathcal{D}} p\,\bar{t}_n \rightarrow\!! t$.*

(5) **Propiedad de Implicación***: $T : \mathcal{I} \Vdash_{\mathcal{D}} \varphi$ y $\varphi \succcurlyeq_{\mathcal{D}} \varphi'$ implican $T' : \mathcal{I} \Vdash_{\mathcal{D}} \varphi'$ con árbol de prueba T' tal que $\mid T' \mid \leq \mid T \mid$.*

(6) **Propiedad de Conservación***: Para cualquier c-hecho φ, se verifica $\mathcal{I} \Vdash_{\mathcal{D}} \varphi$ si y sólo si $\varphi \in \mathcal{I}$.*

(7) **Propiedad de Cierre***: Para cualquier c-sentencia cerrada φ, se verifica que $\mathcal{I} \Vdash_{\mathcal{D}} \varphi$ si y sólo si $[\mathcal{I}]_{\mathcal{D}} \Vdash_{\mathcal{D}} \varphi$.*

3.1.4. Semántica de restricciones definidas por el usuario

Usando el cálculo semántico, es posible definir fácilmente el concepto de solución de una restricción definida por el usuario. La siguiente definición permite generalizar la Definición 3 dada en el Capítulo 2 del concepto de solución de una restricción primitiva, esta vez asumiendo una c-interpretación $\mathcal{I}$ dada sobre un dominio de restricciones $\mathcal{D}$.

Definición 11 (Soluciones de Restricciones Definidas por el Usuario) *Se definen las siguientes nociones semánticas sobre restricciones:*

(1) El conjunto de soluciones de una restricción $\delta \in ACon_{\mathcal{D}}$ es un subconjunto $Sol_{\mathcal{I}}(\delta) \subseteq Val_{\mathcal{D}}$ definido recursivamente como sigue:

- *(a) $Sol_{\mathcal{I}}(\Diamond) = Val_{\mathcal{D}}$.*
- *(b) $Sol_{\mathcal{I}}(\blacklozenge) = \emptyset$.*
- *(c) $Sol_{\mathcal{I}}(\delta) = \{\ \eta \in Val_{\mathcal{D}} \mid \mathcal{I} \Vdash_{\mathcal{D}} \delta\eta\ \}$, para cualquier restricción atómica $\delta \in ACon_{\mathcal{D}} \setminus \{\Diamond, \blacklozenge\}$.*
- *(d) $Sol_{\mathcal{I}}(\delta_1 \wedge \delta_2) = Sol_{\mathcal{I}}(\delta_1) \cap Sol_{\mathcal{I}}(\delta_2)$.*
- *(e) $Sol_{\mathcal{I}}(\exists X.\, \delta) = \{\ \eta \in Val_{\mathcal{D}} \mid \eta' \in Sol_{\mathcal{I}}(\delta)$, para algún $\eta' =_{\setminus\{X\}} \eta\}$.*

(2) El conjunto de soluciones de un conjunto de restricciones $\Delta \subseteq ACon_{\mathcal{D}}$ *se define como* $Sol_{\mathcal{I}}(\Delta) = \bigcap_{\delta \in \Delta} Sol_{\mathcal{I}}(\delta)$*, correspondiendo a una lectura lógica de* Δ *como la conjunción de sus miembros. En particular,* $Sol_{\mathcal{I}}(\emptyset) = Val_{\mathcal{D}}$*, correspondiendo a la lectura lógica de una conjunción vacía como la restricción idénticamente cierta* $\Diamond$*.*

Para las restricciones primitivas, es fácil comprobar que $Sol_{\mathcal{I}}(\pi) = Sol_{\mathcal{D}}(\pi)$ y que $Sol_{\mathcal{I}}(\Pi) = Sol_{\mathcal{D}}(\Pi)$, usando la correspondencia lógica que existe entre la Definición 3 y la Definición 11.

Al igual que en el caso de las restricciones primitivas, vamos a considerar, para un uso posterior, un *Lema de Sustitución* análogo al dado en la Subsección 2.3.2, el cual también puede ser demostrado fácilmente por inducción sobre la estructura sintáctica del conjunto de restricciones Δ definidas por el usuario:

Lema 6 (Lema de Sustitución) *Para cualquier conjunto* $\Delta \subseteq Con_{\mathcal{D}}$*, cualquier sustitución* $\sigma \in Sub_{\mathcal{D}}$*, cualquier c-interpretación* $\mathcal{I}$ *sobre* $\mathcal{D}$*, y cualquier solución* $\eta \in Val_{\mathcal{D}}$*, se cumple la siguiente equivalencia:*

$$\eta \in Sol_{\mathcal{I}}(\Delta\sigma) \Leftrightarrow \sigma\eta \in Sol_{\mathcal{I}}(\Delta)$$

Por último, es posible definir el conjunto de *soluciones bien tipadas* $WTSol_{\mathcal{I}}(\Delta) \subseteq Sol_{\mathcal{I}}(\Delta)$ de una forma análoga a la ya utilizada para restricciones primitivas ($WTSol_{\mathcal{D}}(\Pi) \subseteq Sol_{\mathcal{D}}(\Pi)$) en la Subsección 2.3.2.

3.1.5. Denotación de expresiones

El cálculo semántico también permite definir la *denotación* de expresiones arbitrarias en una c-interpretación dada como sigue:

Definición 12 (Denotación de Expresiones) *Asumimos una c-interpretación dada* $\mathcal{I}$ *sobre un dominio de restricciones* $\mathcal{D}$*. La* denotación *de una expresión* $e \in Exp_{\mathcal{D}}$ *en* $\mathcal{I}$ *bajo una valoración* $\eta \in Val_{\mathcal{D}}$ *se define como el conjunto* $[\![e]\!]^{\mathcal{I}}_{\eta} = \{t \in \mathcal{U}_{\mathcal{D}} \mid \mathcal{I} \Vdash_{\mathcal{D}} e\eta \rightarrow t\}$*. Para el caso de expresiones cerradas* $e \in GExp_{\mathcal{D}}$*, abreviaremos* $[\![e]\!]^{\mathcal{I}}_{\varepsilon}$ *como* $[\![e]\!]^{\mathcal{I}}$*.*

Usando el Lema 5, es fácil demostrar que $[\![e]\!]^{\mathcal{I}}_{\eta} \subseteq \mathcal{U}_{\mathcal{D}}$ incluye el elemento indefinido $\bot$ y es cerrado (por abajo) con respecto al orden de información $\sqsubseteq$; es decir, $t' \in [\![e]\!]^{\mathcal{I}}_{\eta}$ se cumple siempre que $t \in [\![e]\!]^{\mathcal{I}}_{\eta}$ para algún $t \sqsupseteq t'$. Debido a estas propiedades, $[\![e]\!]^{\mathcal{I}}_{\eta}$ se convierte en un elemento del *dominio potencia de Hoare* $\mathcal{HP}(\overline{\mathcal{U}_{\mathcal{D}}})$ [Sco82, Win85, GS90], correspondiendo así con la denominada semántica de *call-time choice* para el indeterminismo. Este tipo de semántica está inspirada por los

trabajos de Hussmann sobre *especificaciones algebraicas* y programas indeterministas [Hus88, Hus92, Hus93] y se ha argumentado que es la más conveniente para la programación lógico funcional en trabajos previos sobre *CRWL* (ver, por ejemplo, [GHLR99, Rod01]).

La Definición 11 y la Definición 12 solo dependen de la información proporcionada por los núcleos básicos de las c-interpretaciones. Por el contrario, el primer apartado de la siguiente definición realmente aprovecha la información no básica proporcionada por las c-interpretaciones.

3.1.6. Modelos fuertes y débiles. Consecuencia lógica

En esta subsección definimos las dos clases de semánticas de modelos que vamos a utilizar en el esquema $CFLP(\mathcal{D})$, denominadas, respectivamente, *semántica débil* y *semántica fuerte*.

Definición 13 (Modelos Fuertes y Débiles) *Para cualquier $CFLP(\mathcal{D})$-programa $\mathcal{P}$ y para cualquier c-interpretación $\mathcal{I}$ diremos que:*

(1) $\mathcal{I}$ es un modelo fuerte *de $\mathcal{P}$ (en símbolos, $\mathcal{I} \models^s_{\mathcal{D}} \mathcal{P}$), si y sólo si, para cualesquiera $(f\,\bar{t}_n \to r \Leftarrow P \,\square\, \Delta) \in \mathcal{P}$, $\theta \in Sub_{\mathcal{D}}$, $\Pi \subseteq APCon_{\mathcal{D}}$ y $t \in Pat_{\mathcal{D}}$ tales que*

- $\mathcal{I} \Vdash_{\mathcal{D}} (P \,\square\, \Delta)\theta \Leftarrow \Pi$,
- $\mathcal{I} \Vdash_{\mathcal{D}} r\theta \to t \Leftarrow \Pi$,

se tiene que $((f\,\bar{t}_n)\theta \to t \Leftarrow \Pi) \in \mathcal{I}$.

(2) $\mathcal{I}$ es un modelo débil *de $\mathcal{P}$ (en símbolos, $\mathcal{I} \models^w_{\mathcal{D}} \mathcal{P}$), si y sólo si, para cualesquiera $(f\,\bar{t}_n \to r \Leftarrow P \,\square\, \Delta) \in \mathcal{P}$, $\eta \in Val_{\mathcal{D}}$ y $t \in GPat_{\mathcal{D}}$ tales que*

- $(f\,\bar{t}_n \to r \Leftarrow P \,\square\, \Delta)\eta$ *es cerrado,*
- $\mathcal{I} \Vdash_{\mathcal{D}} (P \,\square\, \Delta)\eta$,
- $\mathcal{I} \Vdash_{\mathcal{D}} r\eta \to t$,

se tiene que $((f\,\bar{t}_n)\eta \to t) \in \mathcal{I}$.

De una manera informal, la *semántica de modelo débil* $\mathcal{I} \models^w_{\mathcal{D}} \mathcal{P}$ significa que todas las instancias individuales de reglas de programa de $\mathcal{P}$ deben ser válidas en la c-interpretación $\mathcal{I}$. Por consiguiente, una variante técnica de la semántica débil podría definirse usando las $\mathcal{D}$-álgebras introducidas en la Definición 7. En cambio, la *semántica de modelo fuerte* no tendría sentido para $\mathcal{D}$-álgebras. El significado informal de la relación de modelo fuerte $\mathcal{I} \models^s_{\mathcal{D}} \mathcal{P}$ dada en la Definición 13 es el de que todos aquellos c-hechos que son "consecuencias inmediatas" de c-hechos pertenecientes

a $\mathcal{I}$ por reglas de programa de $\mathcal{P}$ deben pertenecer a $\mathcal{I}$. En comparación con otros trabajos previos, la semántica de modelo débil es similar, desde un punto de vista informal, a la noción de *modelo* usada para *CRWL* en [GHLR99, GHR01, AR01], a la semántica del esquema $CFLP(\mathcal{D})$ en [Lop92, Lop94], y a la semántica más tradicional de $CLP(\mathcal{D})$ en [JL87, JM94, JMMS98]. La semántica de modelo fuerte resultaría análoga a la S_2-semántica para $CLP(\mathcal{D})$-programas propuesta en [GL91, GDL95].

La siguiente proposición permite establecer una relación natural entre los modelos débiles y fuertes:

Proposición 1 (Modelos Fuertes vs. Débiles) *Para cualquier $CFLP(\mathcal{D})$-programa $\mathcal{P}$ y para cualquier c-interpretación $\mathcal{I}$ se tiene que: $\mathcal{I} \models^s_{\mathcal{D}} \mathcal{P} \Rightarrow \mathcal{I} \models^w_{\mathcal{D}} \mathcal{P}$. El recíproco es falso en general.*

Demostración 1 *Cualquier modelo fuerte de un $CFLP(\mathcal{D})$-programa $\mathcal{P}$ dado es también modelo débil de $\mathcal{P}$, debido a que el apartado (2) de la Definición 13 es un caso particular del apartado (1), obtenido cuando θ es una sustitución cerrada (una valoración), Π es vacío y t es un patrón cerrado. Como contraejemplo para el recíproco, podemos considerar el siguiente $CFLP(\mathcal{R})$-programa $\mathcal{P}$ formado por una sola regla de programa*

$$notZero\ X \to true \Leftarrow X * X\ /= 0$$

y la siguiente c-interpretación sobre $\mathcal{R}$:

$$\mathcal{I} =_{def} cl_{\mathcal{R}}(\{\ notZero\ X \to true \Leftarrow X > 0,\ notZero\ X \to true \Leftarrow X < 0\ \})$$

Para este programa y esta c-interpretación se cumplen los siguientes resultados:

(a) $\mathcal{I} \models^w_{\mathcal{R}} \mathcal{P}$, debido a que el apartado (2) de la Definición 13 se satisface. De hecho, para cualquier $\eta \in Val_{\mathcal{R}}$ que proporcione una instancia cerrada de la regla de programa de $\mathcal{P}$, se verifica que $\mathcal{I} \Vdash_{\mathcal{R}} (X * X\ /= 0)\eta$ si y sólo si $\eta(X) \in \mathbb{R} \setminus \{0\}$. Por tanto, para cada η en estas condiciones, se tiene que $((notZero\ X)\eta \to true) \in \mathcal{I}$, puesto que $\mathcal{I}$ es cerrada bajo $\mathcal{R}$-implicación.

(b) $\mathcal{I} \not\models^s_{\mathcal{R}} \mathcal{P}$, debido a que el apartado (1) en la Definición 13 falla cuando se escoge ε como θ y $X * X/= 0$ como Π. De hecho, $\mathcal{I} \Vdash_{\mathcal{R}} X * X\ /= 0 \Leftarrow X * X/= 0$, $\mathcal{I} \Vdash_{\mathcal{R}} true \to true \Leftarrow X * X\ /= 0$ y $(notZero\ X \to true \Leftarrow X * X/= 0) \notin \mathcal{I}$, debido a que este c-hecho no se sigue por $\mathcal{R}$-implicación de los c-hechos usados para definir $\mathcal{I}$.

□

Los dos tipos de modelos proporcionados en la Definición 13 dan lugar también, de manera natural, a dos nociones diferentes de consecuencia lógica:

Definición 14 (Consecuencia Fuerte y Débil) *Para cualquier CFLP($\mathcal{D}$)-programa $\mathcal{P}$ dado y para cualquier c-sentencia φ, diremos que*

(1) φ es consecuencia fuerte *de $\mathcal{P}$ (en símbolos, $\mathcal{P} \models^s_{\mathcal{D}} \varphi$), si y sólo si $\mathcal{I} \Vdash_{\mathcal{D}} \varphi$ se cumple para cualquier modelo fuerte $\mathcal{I} \models^s_{\mathcal{D}} \mathcal{P}$.*

(2) φ es consecuencia débil *de $\mathcal{P}$ (en símbolos, $\mathcal{P} \models^w_{\mathcal{D}} \varphi$), si y sólo si $\mathcal{I} \Vdash_{\mathcal{D}} \varphi\eta$ se cumple para cualquier modelo débil $\mathcal{I} \models^w_{\mathcal{D}} \mathcal{P}$ y para cualquier valoración $\eta \in Val_{\mathcal{D}}$ tal que $\varphi\eta$ sea cerrada.*

Como se probará en la Subsección 3.4.3, la consecuencia fuerte siempre implica la consecuencia débil, pero el recíproco es falso en general.

3.2. Una semántica de punto fijo

En esta sección vamos a demostrar la existencia de *modelos mínimos* para *CFLP*($\mathcal{D}$)-programas y a caracterizarlos como *puntos fijos mínimos*, aprovechando la estructura de *retículo* de la familia de todas las c-interpretaciones sobre el dominio de restricciones $\mathcal{D}$. Resultados similares son bien conocidos en el contexto de la programación lógica [Apt90, Llo87a] y en el de la programación lógica con restricciones [JMMS98, GL91, GDL95], así como en el esquema *CFLP*($\mathcal{D}$) de [Lop92, Lop94].

3.2.1. El retículo de las c-interpretaciones sobre un dominio

En nuestro esquema *CFLP*($\mathcal{D}$), la estructura de *retículo* se pone de manifiesto a través del siguiente resultado:

Proposición 2 (Retículo de las c-interpretaciones) *La familia $\mathbb{I}_{\mathcal{D}}$, definida como el conjunto de todas las posibles c-interpretaciones $\mathcal{I}$ sobre el dominio de restricciones $\mathcal{D}$, es un retículo completo con respecto al orden de inclusión de conjuntos. Además, el elemento fondo $\perp\!\!\!\perp$ y el elemento tope $\top\!\!\!\top$ de este retículo pueden caracterizarse, respectivamente, como sigue:*

$$\perp\!\!\!\perp \;=\; cl_{\mathcal{D}}(\{\ \varphi \mid \varphi \text{ es un c-hecho trivial}\ \})$$

$$\top\!\!\!\top \;=\; \{\ \varphi \mid \varphi \text{ es cualquier c-hecho}\ \}$$

Demostración 2 *$\top\!\!\!\top$ es trivialmente el elemento tope de $\mathbb{I}_{\mathcal{D}}$ con respecto al orden de inclusión de conjuntos. Más aún, $\perp\!\!\!\perp$ es el elemento fondo debido a que para cualquier c-interpretación se requiere que incluya todos los c-hechos triviales y además que sea cerrada bajo $cl_{\mathcal{D}}$. Por tanto, sólo queda por demostrar que cualquier subconjunto $\mathfrak{I} \subseteq \mathbb{I}_{\mathcal{D}}$ tiene una menor cota superior $\sqcup\mathfrak{I}$ y una mayor cota inferior $\sqcap\mathfrak{I}$ con respecto al orden de inclusión de conjuntos. Veamos que en efecto esto es así:*

- $\sqcup\mathfrak{I} = cl_{\mathcal{D}}(\bigcup \mathfrak{I})$, *el cual es obviamente el conjunto más pequeño de c-hechos que es cerrado bajo* $cl_{\mathcal{D}}$ *y que incluye a todo* $\mathcal{I} \in \mathfrak{I}$ *como subconjunto. Se observa así que* $\sqcup\emptyset = \perp\!\!\!\perp$ *y que* $\sqcup\mathfrak{I} = \bigcup \mathfrak{I}$ *(el cual es ya cerrado bajo* $cl_{\mathcal{D}}$*) para un* $\mathfrak{I}$ *no vacío.*

- $\sqcap\mathfrak{I} = \bigcap \mathfrak{I}$ *(entendido como* $\top\!\!\!\top$ *si* $\mathfrak{I}$ *es vacío), el cual es cerrado bajo* $cl_{\mathcal{D}}$ *y es el conjunto más grande de c-hechos que está incluido como un subconjunto en todo* $\mathcal{I} \in \mathfrak{I}$.

□

3.2.2. Operadores de transformación sobre c-interpretaciones

Las transformaciones de c-interpretaciones que se definen a continuación, para el caso fuerte y para el caso débil, permiten formalizar, respectivamente, la computación de "consecuencias inmediatas" fuertes y débiles a partir de los c-hechos que pertenecen a una c-interpretación dada.

Definición 15 (Transformaciones de c-interpretaciones) *Para cualquier programa dado* $\mathcal{P}$ *y para cualquier c-interpretación* $\mathcal{I}$ *sobre un dominio* $\mathcal{D}$*, definimos:*

(1) $ST_{\mathcal{P}}(\mathcal{I}) =_{def} cl_{\mathcal{D}}(preST_{\mathcal{P}}(\mathcal{I}))$*, donde* $preST_{\mathcal{P}}(\mathcal{I})$ *es el conjunto de todos los c-hechos de la forma* $(f\,\bar{t}_n)\theta \to t \Leftarrow \Pi$ *tales que*

- $(f\,\bar{t}_n \to r \Leftarrow P \,\square\, \Delta) \in \mathcal{P}$,
- $\theta \in Sub_{\mathcal{D}}$, $\Pi \subseteq PCon_{\mathcal{D}}$, $t \in Pat_{\mathcal{D}}$,
- $\mathcal{I} \Vdash_{\mathcal{D}} (P \,\square\, \Delta)\theta \Leftarrow \Pi$,
- $\mathcal{I} \Vdash_{\mathcal{D}} r\theta \to t \Leftarrow \Pi$.

(2) $WT_{\mathcal{P}}(\mathcal{I}) =_{def} cl_{\mathcal{D}}(preWT_{\mathcal{P}}(\mathcal{I}))$*, donde* $preWT_{\mathcal{P}}(\mathcal{I})$ *es el conjunto de c-hechos cerrados de la forma* $(f\,\bar{t}_n)\eta \to t$ *tales que*

- $(f\,\bar{t}_n \to r \Leftarrow P \,\square\, \Delta) \in \mathcal{P}$,
- $\eta \in Val_{\mathcal{D}}$ *con* $(f\,\bar{t}_n \to r \Leftarrow P \,\square\, \Delta)\eta$ *una instancia cerrada,* $t \in GPat_{\mathcal{D}}$,
- $\mathcal{I} \Vdash_{\mathcal{D}} (P \,\square\, \Delta)\eta$,
- $\mathcal{I} \Vdash_{\mathcal{D}} r\eta \to t$.

3.2.3. Modelos mínimos fuertes y débiles

Las principales propiedades relativas a las transformaciones de c-interpretaciones de la Definición 15 se recogen en la siguiente proposición, cuya demostración puede encontrarse en el Apéndice A:

Proposición 3 (Propiedades de las Transformaciones de Interpretaciones) *Para cualquier CFLP($\mathcal{D}$)-programa $\mathcal{P}$ fijado, los operadores de transformación $ST_{\mathcal{P}}$, $WT_{\mathcal{P}} : \mathbb{I}_{\mathcal{D}} \to \mathbb{I}_{\mathcal{D}}$ están bien definidos como funciones continuas, cuyos pre-puntos fijos son los modelos fuertes, respectivamente los modelos débiles, de $\mathcal{P}$. De manera más precisa, para cualquier $\mathcal{I} \in \mathbb{I}_{\mathcal{D}}$ se tiene que*

(a) $ST_{\mathcal{P}}(\mathcal{I}) \subseteq \mathcal{I}$ *si y sólo si* $\mathcal{I} \models^{s}_{\mathcal{D}} \mathcal{P}$,

(b) $WT_{\mathcal{P}}(\mathcal{I}) \subseteq \mathcal{I}$ *si y sólo si* $\mathcal{I} \models^{w}_{\mathcal{D}} \mathcal{P}$.

Usando la anterior proposición, es fácil obtener la caracterización deseada de *modelo mínimo*:

Teorema 2 (Modelo Mínimo de un Programa) *Para cualquier CFLP($\mathcal{D}$)-programa $\mathcal{P}$, existe:*

(1) Un modelo mínimo fuerte $\mathcal{S}_{\mathcal{P}} = lfp(ST_{\mathcal{P}}) = \bigcup_{k\in\mathbb{N}} ST_{\mathcal{P}} \uparrow^{k} (\perp\!\!\!\perp)$.

(2) Un modelo mínimo débil $\mathcal{W}_{\mathcal{P}} = lfp(WT_{\mathcal{P}}) = \bigcup_{k\in\mathbb{N}} WT_{\mathcal{P}} \uparrow^{k} (\perp\!\!\!\perp)$.

Demostración 3 *Debido al bien conocido* Teorema de Knaster y Tarski *[Tar55], una función monótona de un retículo completo en sí mismo siempre tiene un menor punto fijo que es también su menor pre-punto fijo. En el caso en el que la función sea continua, su menor punto fijo puede caracterizarse como la menor cota superior de la secuencia de elementos del retículo obtenida mediante la aplicación reiterada de la función a partir del elemento fondo. Combinando estos resultados con los de la Proposición 3, el teorema se demuestra trivialmente.*

□

3.2.4. Relación entre modelos mínimos fuertes y débiles

En el resto de esta sección, vamos a investigar la relación que existe entre los dos modelos mínimos, $\mathcal{S}_{\mathcal{P}}$ y $\mathcal{W}_{\mathcal{P}}$, de un programa dado $\mathcal{P}$, cuya existencia ha sido probada en el Teorema 2. Obviamente, $\mathcal{W}_{\mathcal{P}} \subseteq \mathcal{S}_{\mathcal{P}}$, debido a que $\mathcal{S}_{\mathcal{P}}$ es un modelo débil de $\mathcal{P}$ por la Proposición 1, y por tanto, $\mathcal{S}_{\mathcal{P}}$ ha de incluir al modelo mínimo débil. Una caracterización más precisa de $\mathcal{W}_{\mathcal{P}}$ como un subconjunto de $\mathcal{S}_{\mathcal{P}}$ se sigue a partir del siguiente resultado, cuya demostración también se incluye en el Apéndice A.

Proposición 4 (Relación entre las dos Transformaciones) *Para cualquier CFLP($\mathcal{D}$)-programa dado $\mathcal{P}$ y para cualquier c-interpretación $\mathcal{I}$ sobre el dominio $\mathcal{D}$, se cumple la siguiente igualdad:*

$$WT_{\mathcal{P}}([\mathcal{I}]_{\mathcal{D}}) = [ST_{\mathcal{P}}(\mathcal{I})]_{\mathcal{D}}$$

Usando esta proposición, podemos probar el siguiente teorema, el cual nos permite establecer una relación entre modelos mínimos.

Teorema 3 (Relación entre Modelos Mínimos) *Para cualquier $CFLP(\mathcal{D})$-programa $\mathcal{P}$, el menor modelo débil de $\mathcal{P}$ es el $\mathcal{D}$-cierre del menor modelo fuerte de $\mathcal{P}$, es decir,*

$$\mathcal{W}_\mathcal{P} = [\mathcal{S}_\mathcal{P}]_\mathcal{D}$$

Demostración 4 *Debido al Teorema 2, es suficiente con demostrar que $WT_\mathcal{P} \uparrow^k (\perp\!\!\!\perp) = [ST_\mathcal{P} \uparrow^k (\perp\!\!\!\perp)]_\mathcal{D}$ se cumple para todo $k \in \mathbb{N}$. Razonamos por inducción sobre k:*

- Caso base*: La ecuación se cumple para $k = 0$, puesto que $WT_\mathcal{P} \uparrow^0 (\perp\!\!\!\perp) = \perp\!\!\!\perp = [\perp\!\!\!\perp]_\mathcal{D} = [ST_\mathcal{P} \uparrow^0 (\perp\!\!\!\perp)]_\mathcal{D}$.*
- Paso inductivo*: Asumamos como* hipótesis de inducción *que la ecuación que queremos demostrar se cumple para k. Entonces:*

$$\begin{array}{rcl} WT_\mathcal{P} \uparrow^{k+1} (\perp\!\!\!\perp) & = & WT_\mathcal{P}(WT_\mathcal{P} \uparrow^k (\perp\!\!\!\perp)) \\ & = & WT_\mathcal{P}([ST_\mathcal{P} \uparrow^k (\perp\!\!\!\perp)]_\mathcal{D}) \\ & = & [ST_\mathcal{P}(ST_\mathcal{P} \uparrow^k (\perp\!\!\!\perp))]_\mathcal{D} \\ & = & [ST_\mathcal{P} \uparrow^{k+1} (\perp\!\!\!\perp)]_\mathcal{D} \end{array}$$

donde la segunda ecuación se cumple por hipótesis de inducción*, y la tercera ecuación se cumple por la Proposición 4.*

□

En la Subsección 3.4.3 presentaremos un ejemplo en el que se muestra que la inclusión $\mathcal{W}_\mathcal{P} \subseteq \mathcal{S}_\mathcal{P}$ puede ser estricta para algunos $CFLP(\mathcal{D})$-programas.

3.3. La lógica de reescritura $CRWL(\mathcal{D})$

La *Lógica de Reescritura con Restricciones $CRWL(\mathcal{D})$* es una extensión natural de la lógica de reescritura *CRWL* (del inglés, *Constructor-based ReWriting Logic*). Esta lógica fue propuesta originalmente en [GHLR96, GHLR99] como un marco lógico para lenguajes de programación lógico funcional que estuvieran basados en funciones perezosas y posiblemente indeterministas, cuya semántica no puede ser directamente descrita en términos de la lógica ecuacional. Los primeros trabajos sobre *CRWL* fueron inspirados por el trabajo de Hussmann sobre indeterminismo en programas y especificaciones algebraicas [Hus88, Hus92, Hus93].

A pesar de su nombre, y en comparación con la Lógica de Reescritura de Meseguer [Mes92, MM02], propuesta inicialmente como un marco lógico y semántico unificado para lenguajes y sistemas concurrentes, la lógica para la reescritura *CRWL* muestra diferencias claras tanto en objetivos como en motivación. Una comparativa detallada de ambas aproximaciones puede encontrarse en [Pal02, Pal07]. En estos trabajos se demuestra cómo las semánticas de ambas lógicas, cuando son vistas como *instituciones*, son formalmente incomparables.

En los últimos años varias extensiones de *CRWL* han sido desarrolladas con el fin de tener en cuenta diferentes características de los lenguajes lógico funcionales, como las funciones de order-superior [GHR97], las lambda abstracciones [Vad07], los tipos polimórficos [GHR01], las constructoras de datos algebraicos [AR97a, AR97b, AR01], un tratamiento adecuado de ciertos tipos de restricciones [ALR98, ALR99] e incluso el fallo finito [LS03, LS04]. Una panorámica del trabajo previo que se ha desarrollado en los últimos años sobre *CRWL* puede encontrarse en [Rod01].

Una extensión genérica $CRWL(\mathcal{D})$ de *CRWL* sobre un dominio de restricciones $\mathcal{D}$ dado como parámetro y orientada a caracterizar la semántica declarativa de $CFLP(\mathcal{D})$-programas, se presentó por primera vez en [LRV04a, LRV07] y va a ser expuesta en esta sección. Para ello, comenzaremos presentando un cálculo lógico para $CRWL(\mathcal{D})$ e investigando sus principales propiedades teóricas. Seguidamente estudiaremos la relación que existe entre la derivación formal en $CRWL(\mathcal{D})$ y las semánticas de modelos presentadas en secciones anteriores. En particular, obtendremos resultados que caracterizan los modelos mínimos de los $CFLP(\mathcal{D})$-programas en términos de $CRWL(\mathcal{D})$-derivabilidad.

3.3.1. El cálculo de la lógica $CRWL(\mathcal{D})$

La siguiente definición asume un dominio de restricciones $\mathcal{D}$ y un $CFLP(\mathcal{D})$-programa $\mathcal{P}$ dados. El propósito del cálculo lógico $CRWL(\mathcal{D})$ es el de inferir la validez semántica de c-sentencias arbitrarias a partir de las reglas de programa de $\mathcal{P}$.

Definición 16 (Cálculo de la Reescritura con Restricciones) *Escribiremos $\mathcal{P} \vdash_{\mathcal{D}} \varphi$ para indicar que la c-sentencia φ puede ser derivada a partir de las reglas de programa de $\mathcal{P}$ en el* cálculo de reescritura con restricciones *$CRWL(\mathcal{D})$, el cual está formado por las reglas de inferencia* **TI**, **RR**, **SP**, **DC**, **IR**, **PF** *y* **AC** *ya presentadas en el* cálculo semántico *de la Definición 10, más la siguiente regla de inferencia:*

DF$_{\mathcal{P}}$ *$\mathcal{P}$-Función Definida:*

$$\frac{e_1 \to t_1 \Leftarrow \Pi, \cdots, e_n \to t_n \Leftarrow \Pi,\ P \,\square\, \Delta \Leftarrow \Pi,\ r \to t \Leftarrow \Pi}{f\,\overline{e}_n \to t \Leftarrow \Pi}$$

Si $f \in DF^n$ *y* $(f\,\overline{t}_n \to r \Leftarrow P \,\square\, \Delta) \in [\mathcal{P}]_{\perp}$.

$$\frac{e_1 \to t_1 \Leftarrow \Pi, \cdots, e_n \to t_n \Leftarrow \Pi,\ P \,\square\, \Delta \ \Leftarrow \Pi,\ r \to s \Leftarrow \Pi,\ s\,\overline{a}_k \to t \Leftarrow \Pi}{f\,\overline{e}_n\overline{a}_k \ \to\ t \ \Leftarrow \Pi}$$

Si $f \in DF^n$, $k > 0$, $(f\,\overline{t}_n \to r \Leftarrow P \,\square\, \Delta) \in [\mathcal{P}]_\perp$ *y* $s \in Pat_{\mathcal{D}}$.

La diferencia más importante entre el cálculo lógico $CRWL(\mathcal{D})$ y el cálculo semántico es que $CRWL(\mathcal{D})$ infiere el comportamiento de funciones definidas a partir de las reglas de un programa dado $\mathcal{P}$, en vez de a partir de una c-interpretación dada $\mathcal{I}$. Esto se ve claramente a partir de la formulación de la regla $\mathbf{DF}_{\mathcal{P}}$, donde $[\mathcal{P}]_\perp$ denota el conjunto $\{R\theta \mid R \in \mathcal{P}, \theta \in Sub_{\mathcal{D}}\}$ formado por todas las posibles instancias de las reglas de definición de función pertenecientes a $\mathcal{P}$.

Como en el caso de las reglas del cálculo semántico, convenimos en que ninguna regla de inferencia del cálculo $CRWL(\mathcal{D})$ pueda ser aplicada si existe alguna regla previa (en el orden textual en el que han sido presentadas) que pueda ser usada. Más aún, también convenimos en que la premisa $P \,\square\, \Delta \Leftarrow \Pi$ en la regla $\mathbf{DF}_{\mathcal{P}}$ se entienda como una abreviatura de varias premisas $\alpha \Leftarrow \Pi$, una para cada sentencia atómica α que aparezca en $P \,\square\, \Delta$. Esta convención sintáctica inofensiva permite prescindir de una regla de inferencia adicional.

3.3.2. Árboles de prueba en $CRWL(\mathcal{D})$

Las derivaciones obtenidas mediante la aplicación de las reglas del cálculo $CRWL(\mathcal{D})$ pueden ser representadas como *árboles de prueba*, cuyos nodos quedan etiquetados mediante c-sentencias, y donde cada nodo ha sido inferido a partir de sus hijos por medio de alguna regla de inferencia del cálculo $CRWL(\mathcal{D})$. En relación con los árboles de prueba y sus tamaños, usaremos la misma notación y terminología que se introdujo para el cálculo semántico en la Subsección 3.1.3, considerando tan solo el reemplazamiento de la regla $\mathbf{DF}_{\mathcal{I}}$ por la regla $\mathbf{DF}_{\mathcal{P}}$. En particular, $T : \mathcal{P} \vdash_{\mathcal{D}} \varphi$ indicará que $\mathcal{P} \vdash_{\mathcal{D}} \varphi$ tiene por *testigo* el árbol de prueba T.

Ejemplo 10 *El árbol de prueba T mostrado más abajo corresponde a la siguiente $CRWL(\mathcal{R})$-derivación:*

$$T : \mathcal{P} \vdash_{\mathcal{R}} takeWhile\ (inInterval\ 0\ 1)\ (from\ X\ 0.5) == [X] \Leftarrow X > 0.5,\ X \leq 1$$

que sirve de testigo de la corrección declarativa de la respuesta computada que se discutió en el Ejemplo 8. Con el fin de facilitar la comprensión de esta derivación, y por tanto de la construcción del árbol de prueba T asociado a ella, consideraremos separadamente un subárbol de T denominado T_0.

AC $takeWhile\ (inInterval\ 0\ 1)\ (from\ X\ 0.5) == [X] \Leftarrow X > 0.5,\ X \leq 1$
$X > 0.5,\ X \leq 1 \models_{\mathcal{R}} [X] == [X]$
DC $[X] \rightarrow [X] \Leftarrow X > 0.5,\ X \leq 1$
RR $X \rightarrow X \Leftarrow X > 0.5,\ X \leq 1$
DC $[\ ] \rightarrow [\ ] \Leftarrow X > 0.5,\ X \leq 1$
DF $takeWhile\ (inInterval\ 0\ 1)\ (from\ X\ 0.5) \rightarrow [X] \Leftarrow X > 0.5,\ X \leq 1$
DC $inInterval\ 0\ 1 \rightarrow inInterval\ 0\ 1 \Leftarrow X > 0.5,\ X \leq 1$
RR $0 \rightarrow 0 \Leftarrow X > 0.5,\ X \leq 1$
RR $1 \rightarrow 1 \Leftarrow X > 0.5,\ X \leq 1$
DF $from\ X\ 0.5 \rightarrow [\ X, X + 0.5 \mid \bot\ ] \Leftarrow X > 0.5,\ X \leq 1$
RR $X \rightarrow X \Leftarrow X > 0.5,\ X \leq 1$
RR $0.5 \rightarrow 0.5 \Leftarrow X > 0.5,\ X \leq 1$
DC $[\ X \mid from\ (X + 0.5)\ 0.5\ ] \rightarrow [\ X, X + 0.5 \mid \bot\ ] \Leftarrow X > 0.5,$
$X \leq 1$
RR $X \rightarrow X \Leftarrow X > 0.5,\ X \leq 1$
DF $from\ (X + 0.5)\ 0.5 \rightarrow [\ X + 0.5 \mid \bot\ ] \Leftarrow X > 0.5,$
$X \leq 1$
PF $X + 0.5 \rightarrow X + 0.5 \Leftarrow X > 0.5,\ X \leq 1$
RR $X \rightarrow X \Leftarrow X > 0.5,\ X \leq 1$
RR $0.5 \rightarrow 0.5 \Leftarrow X > 0.5,\ X \leq 1$
$X > 0.5,\ X \leq 1 \models_{\mathcal{R}} X + 0.5 \rightarrow X + 0.5$
RR $0.5 \rightarrow 0.5 \Leftarrow X > 0.5,\ X \leq 1$
DC $[\ X + 0.5 \mid from\ ((X + 0.5) + 0.5)\ 0.5\ ] \rightarrow$
$[\ X + 0.5 \mid \bot\ ] \Leftarrow X > 0.5,\ X \leq 1$
PF $X + 0.5 \rightarrow X + 0.5 \Leftarrow X > 0.5,\ X \leq 1$
TI $from\ ((X + 0.5) + 0.5)\ 0.5 \rightarrow \bot \Leftarrow X > 0.5,$
$X \leq 1$
DF $if\ (inInterval\ 0\ 1\ X)\ [\ X \mid takeWhile\ (inInterval\ 0\ 1)$
$[X + 0.5 \mid \bot]\ ]\ [\ ] \rightarrow [X] \Leftarrow X > 0.5,\ X \leq 1$
DF $inInterval\ 0\ 1\ X \rightarrow true \Leftarrow X > 0.5,\ X \leq 1$
RR $0 \rightarrow 0 \Leftarrow X > 0.5,\ X \leq 1$
RR $1 \rightarrow 1 \Leftarrow X > 0.5,\ X \leq 1$
RR $X \rightarrow X \Leftarrow X > 0.5,\ X \leq 1$
DF $and\ (0 \leq X)\ (X \leq 1) \rightarrow true \Leftarrow X > 0.5,\ X \leq 1$
PF $0 \leq X \rightarrow true \Leftarrow X > 0.5,\ X \leq 1$
RR $0 \rightarrow 0 \Leftarrow X > 0.5,\ X \leq 1$
RR $X \rightarrow X \Leftarrow X > 0.5,\ X \leq 1$
$X > 0.5,\ X \leq 1 \models_{\mathcal{R}} 0 \leq X \rightarrow true$
PF $X \leq 1 \rightarrow true \Leftarrow X > 0.5,\ X \leq 1$
RR $X \rightarrow X \Leftarrow X > 0.5,\ X \leq 1$
RR $1 \rightarrow 1 \Leftarrow X > 0.5,\ X \leq 1$
$X > 0.5,\ X \leq 1 \models_{\mathcal{R}} X \leq 1 \rightarrow true$

$$\begin{array}{l}
\qquad\qquad\quad \mathbf{DC}\ true \to true \Leftarrow X > 0.5,\ X \leq 1 \\
\mathbf{DC}\ [\ X \mid takeWhile\ (inInterval\ 0\ 1)\ [X + 0.5 \mid \bot]\ ] \to [X] \\
\qquad \Leftarrow X > 0.5,\ X \leq 1 \\
\qquad \mathbf{RR}\ X \to X \Leftarrow X > 0.5,\ X \leq 1 \\
\qquad \mathbf{DC}\ takeWhile\ (inInterval\ 0\ 1)\ [X + 0.5 \mid \bot] \to [] \\
\qquad\quad \Leftarrow X > 0.5,\ X \leq 1 \\
\\
\qquad\qquad\qquad\qquad \ldots\ T_0\ \ldots \\
\\
\mathbf{DC}\ [\,] \to [\,] \Leftarrow X > 0.5,\ X \leq 1 \\
\mathbf{DC}\ [X] \to [X] \Leftarrow X > 0.5,\ X \leq 1 \\
\qquad \mathbf{RR}\ X \to X \Leftarrow X > 0.5,\ X \leq 1 \\
\qquad \mathbf{DC}\ [\,] \to [\,] \Leftarrow X > 0.5,\ X \leq 1
\end{array}$$

El subárbol T_0 sirve de testigo a la derivación

$$T_0 : \mathcal{P} \vdash_{\mathcal{R}} takeWhile\ (inInterval\ 0\ 1)\ [\ X + 0.5 \mid \bot\ ] \to [\,] \Leftarrow X > 0.5,\ X \leq 1$$

La Figura 3.1 muestra una representación gráfica de T_0, donde la parte común de restricciones $X > 0.5,\ X \leq 1$ ha sido omitida para que la figura resulte más legible.

$$\begin{array}{l}
\mathbf{DF}\ takeWhile\ (inInterval\ 0\ 1)\ [\ X + 0.5 \mid \bot\ ] \to [\,] \Leftarrow X > 0.5,\ X \leq 1 \\
\quad \mathbf{DC}\ inInterval\ 0\ 1 \to inInterval\ 0\ 1 \Leftarrow X > 0.5,\ X \leq 1 \\
\quad \mathbf{DC}\ [\ X + 0.5 \mid \bot\ ] \to [\ X + 0.5 \mid \bot\ ] \Leftarrow X > 0.5,\ X \leq 1 \\
\quad \mathbf{DF}\ if\ (inInterval\ 0\ 1\ (X{+}0.5))\ [X{+}0.5 \mid takeWhile\ (inInterval\ 0\ 1)\ \bot]\ [\,] \\
\qquad \to [\,] \Leftarrow X > 0.5,\ X \leq 1 \\
\qquad \mathbf{DF}\ inInterval\ 0\ 1\ (X + 0.5) \to false \Leftarrow X > 0.5,\ X \leq 1 \\
\qquad\quad \mathbf{RR}\ 0 \to 0 \Leftarrow X > 0.5,\ X \leq 1 \\
\qquad\quad \mathbf{RR}\ 1 \to 1 \Leftarrow X > 0.5,\ X \leq 1 \\
\qquad\quad \mathbf{PF}\ X + 0.5 \to X + 0.5 \Leftarrow X > 0.5,\ X \leq 1 \\
\qquad\quad \mathbf{DF}\ and\ (0 \leq X + 0.5)\ (X + 0.5 \leq 1) \to false \Leftarrow X > 0.5,\ X \leq 1 \\
\qquad\qquad \mathbf{PF}\ 0 \leq X + 0.5 \to true \Leftarrow X > 0.5,\ X \leq 1 \\
\qquad\qquad\quad \mathbf{RR}\ 0 \to 0 \Leftarrow X > 0.5,\ X \leq 1 \\
\qquad\qquad\quad \mathbf{PF}\ X + 0.5 \to X + 0.5 \Leftarrow X > 0.5,\ X \leq 1 \\
\qquad\qquad\quad X > 0.5,\ X \leq 1 \models_{\mathcal{R}} 0 \leq X + 0.5 \to true \\
\qquad\qquad \mathbf{PF}\ X + 0.5 \leq 1 \to false \Leftarrow X > 0.5,\ X \leq 1 \\
\qquad\qquad\quad \mathbf{PF}\ X + 0.5 \to X + 0.5 \Leftarrow X > 0.5,\ X \leq 1 \\
\qquad\qquad\quad \mathbf{RR}\ 1 \to 1 \Leftarrow X > 0.5,\ X \leq 1 \\
\qquad\qquad\quad X > 0.5,\ X \leq 1 \models_{\mathcal{R}} X + 0.5 \leq 1 \to false \\
\qquad\qquad \mathbf{DC}\ false \to false \Leftarrow X > 0.5,\ X \leq 1 \\
\qquad \mathbf{TI}\ [\ X + 0.5 \mid takeWhile\ (inInterval\ 0\ 1)\ \bot\ ] \to \bot \Leftarrow X > 0.5,
\end{array}$$

$$X \leq 1$$
$$\mathbf{DC}\ [\,] \rightarrow [\,] \Leftarrow X > 0.5,\ X \leq 1$$
$$\mathbf{DC}\ [\,] \rightarrow [\,] \Leftarrow X > 0.5,\ X \leq 1$$

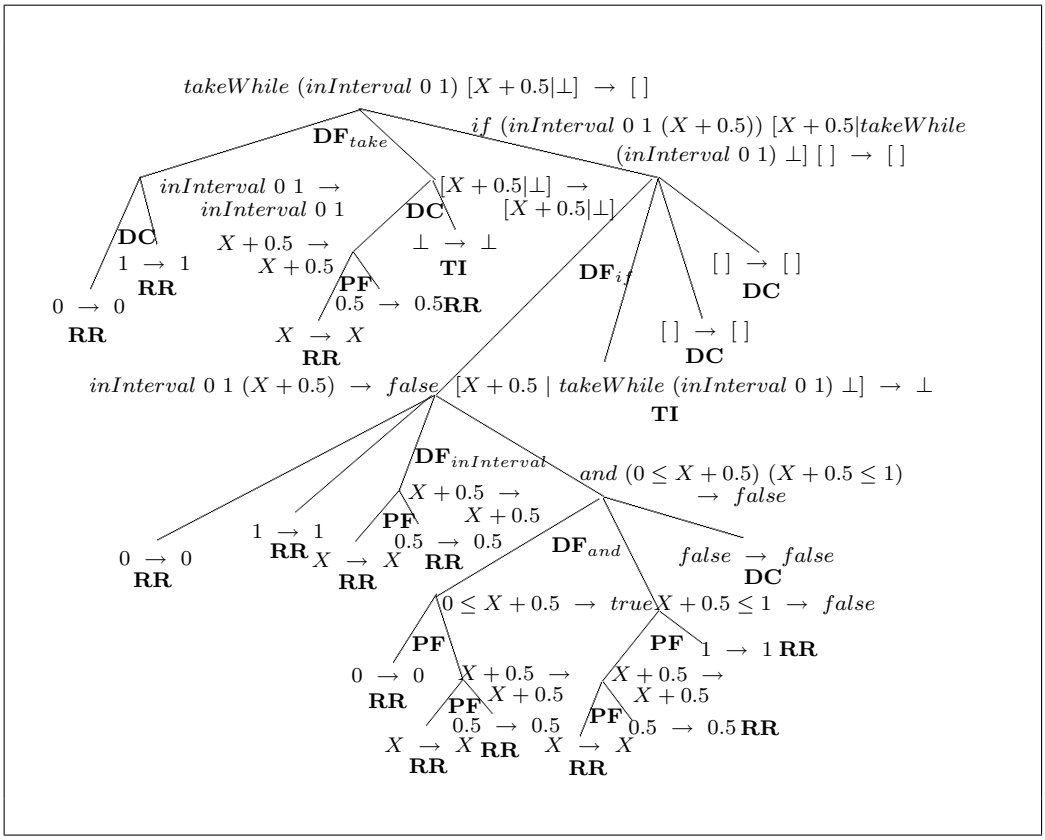


Figura 3.1: Un $CRWL(\mathcal{R})$-árbol de prueba para el Ejemplo 10

3.3.3. Propiedades del cálculo $CRWL(\mathcal{D})$

La mayoría de las propiedades que han sido enunciadas y demostradas a través del Lema 5 para el cálculo semántico sobre c-interpretaciones pueden ser transformadas directamente en propiedades análogas válidas para el cálculo de reescritura con restricciones $CRWL(\mathcal{D})$. La única excepción la constituyen los apartados (6) y (7) del Lema 5, los cuales no parecen tener una analogía natural en $CRWL(\mathcal{D})$. De manera más precisa, las propiedades del cálculo $CRWL(\mathcal{D})$ se establecen formalmente en el siguiente lema. Como en el caso del Lema 5, la demostración de este resultado es también bastante técnica y puede consultarse en el Apéndice A.

Lema 7 (Propiedades del Cálculo de Reescritura con Restricciones) *El cálculo de reescritura con restricciones $CRWL(\mathcal{D})$ satisface las siguientes propiedades:*

(1) **Propiedad de Compacidad***: $\mathcal{P} \vdash_{\mathcal{D}} \varphi$ implica que $\mathcal{P}_0 \vdash_{\mathcal{D}} \varphi$ para algún subconjunto finito $\mathcal{P}_0 \subseteq \mathcal{P}$.*

(2) **Propiedad de Extensión***: $\mathcal{P} \vdash_{\mathcal{D}} \varphi$ y $\mathcal{P} \subseteq \mathcal{P}'$ implican que $\mathcal{P}' \vdash_{\mathcal{D}} \varphi$.*

(3) **Propiedad de Aproximación***: Para cualquier $e \in Exp_{\mathcal{D}}$ y $t \in Pat_{\mathcal{D}}$, se verifica que $\Pi \models_{\mathcal{D}} e \sqsupseteq t$ si y sólo si existe algún árbol de prueba simple T tal que $T :: \vdash_{\mathcal{D}} e \rightarrow t \Leftarrow \Pi$ (entendido como una derivación a partir de un programa vacío).*

(4) **c-Átomos primitivos***: Para cualquier átomo primitivo $p\,\bar{t}_n \rightarrow! \, t$, se verifica que $\mathcal{P} \vdash_{\mathcal{D}} p\,\bar{t}_n \rightarrow! \, t \Leftarrow \Pi$ si y sólo si $\Pi \models_{\mathcal{D}} p\,\bar{t}_n \rightarrow! \, t$.*

(5) **Propiedad de Implicación***: $T :: \mathcal{P} \vdash_{\mathcal{D}} \varphi$ y $\varphi \succcurlyeq_{\mathcal{D}} \varphi'$ implican $T' :: \mathcal{P} \vdash_{\mathcal{D}} \varphi'$ con árbol de prueba T' tal que $\mid T' \mid \leq \mid T \mid$.*

3.4. Teoría de modelos

Los resultados de la presente sección tratan de mostrar una correspondencia natural entre las $CRWL(\mathcal{D})$-derivaciones, el concepto de consecuencia lógica a partir de un $CFLP(\mathcal{D})$-programa dado presentado en la Subsección 3.1.6 y los modelos mínimos de un $CFLP(\mathcal{D})$-programa presentados en la Subsección 3.2.3. A partir de estos resultados, puede esperarse que las $CRWL(\mathcal{D})$-derivaciones de la forma $T :: \mathcal{P} \vdash_{\mathcal{D}} G\sigma \Leftarrow \Pi$ existan si y sólo si la pareja $\Pi \square \sigma$ es una *respuesta correcta* (en el sentido del marco $CFLP(\mathcal{D})$) para el objetivo G con respecto al programa $\mathcal{P}$. Por esta razón, los árboles de prueba que sirven de testigos a tales derivaciones juegan un papel importante en las demostraciones matemáticas de corrección y completitud de los cálculos de resolución de objetivos que se presentarán en el Capítulo 4.

3.4.1. Resultados de adecuación para la semántica fuerte

En esta sección vamos a investigar la relación que existe entre la $CRWL(\mathcal{D})$-derivabilidad y las dos semánticas de modelos que han sido presentadas en las secciones precedentes. El primer resultado muestra una correspondencia entre la $CRWL(\mathcal{D})$-derivabilidad, el concepto de consecuencia fuerte y la validez en el menor modelo fuerte.

Teorema 4 (Resultados de Adecuación para la Semántica Fuerte) *Para cualquier $CFLP(\mathcal{D})$-programa $\mathcal{P}$ y para cualquier c-sentencia φ, las siguientes tres*

condiciones son equivalentes:

$(a)\ \mathcal{P} \vdash_{\mathcal{D}} \varphi \qquad (b)\ \mathcal{P} \models^{s}_{\mathcal{D}} \varphi \qquad (c)\ \mathcal{S}_{\mathcal{P}} \Vdash_{\mathcal{D}} \varphi$

Más aún, también se verifican las tres propiedades siguientes:

(1) **Corrección***: Para cualquier c-sentencia* φ, $\mathcal{P} \vdash_{\mathcal{D}} \varphi \Rightarrow \mathcal{P} \models^{s}_{\mathcal{D}} \varphi$.

(2) **Completitud***: Para cualquier c-sentencia* φ, $\mathcal{P} \models^{s}_{\mathcal{D}} \varphi \Rightarrow \mathcal{P} \vdash_{\mathcal{D}} \varphi$.

(3) **Canonicidad***:* $\mathcal{S}_{\mathcal{P}} = \{\ \varphi \mid \varphi$ es un c-hecho y $\mathcal{P} \vdash_{\mathcal{D}} \varphi\ \}$.

Demostración 5 *La demostración de la equivalencia entre las condiciones* (a), (b) *y* (c) *se da en el Apéndice A. La corrección y la completitud son una consecuencia trivial de esta equivalencia. Con el fin de poder probar la canonicidad, consideramos cualquier c-hecho* φ. *Sabemos que* $\varphi \in \mathcal{S}_{\mathcal{P}}$ *si y sólo si* $\mathcal{S}_{\mathcal{P}} \Vdash_{\mathcal{D}} \varphi$, *debido a la* Propiedad de Conservación *del Lema 5. Por otra parte,* $\mathcal{S}_{\mathcal{P}} \Vdash_{\mathcal{D}} \varphi$ *si y sólo si* $\mathcal{P} \vdash_{\mathcal{D}} \varphi$ *está garantizado por la equivalencia entre* (c) *y* (a).

□

3.4.2. Resultados de adecuación para la semántica débil

En cuanto a la relación entre la *CRWL*($\mathcal{D}$)-derivabilidad y la semántica débil, la mayor parte de los resultados de la subsección anterior (con la excepción de la *propiedad de corrección*) deben restringirse a c-sentencias cerradas:

Teorema 5 (Resultados de Adecuación para la Semántica Débil) *Para cualquier CFLP*($\mathcal{D}$)*-programa* $\mathcal{P}$ *y para cualquier c-sentencia cerrada* φ, *las siguientes tres condiciones son equivalentes:*

$(a)\ \mathcal{P} \vdash_{\mathcal{D}} \varphi \qquad (b)\ \mathcal{P} \models^{w}_{\mathcal{D}} \varphi \qquad (c)\ \mathcal{W}_{\mathcal{P}} \Vdash_{\mathcal{D}} \varphi$

Más aún, también se verifican las tres propiedades siguientes:

(1) **Corrección***: Para cualquier c-sentencia* φ, $\mathcal{P} \vdash_{\mathcal{D}} \varphi \Rightarrow \mathcal{P} \models^{w}_{\mathcal{D}} \varphi$.

(2) **Completitud Básica***: Para cualquier c-sentencia cerrada* φ, $\mathcal{P} \models^{w}_{\mathcal{D}} \varphi \Rightarrow \mathcal{P} \vdash_{\mathcal{D}} \varphi$. *Esto no se cumple, en general, para c-sentencias arbitrarias.*

(3) **Canonicidad Básica***:* $gk_{\mathcal{D}}(\mathcal{W}_{\mathcal{P}}) = \{\varphi \mid \varphi$ es un c-hecho cerrado y $\mathcal{P} \vdash_{\mathcal{D}} \varphi\}$.

Demostración 6 *Los resultados enunciados en este teorema pueden deducirse fácilmente del Teorema 4, usando las relaciones ya demostradas entre modelos fuertes y débiles. Las argumentaciones detalladas pueden consultarse en el Apéndice A.*

□

3.4.3. Semántica fuerte vs. semántica débil

Usando los Teoremas 4 y 5 es posible obtener fácilmente dos resultados que se anunciaron al final de las Subsecciones 3.1.6 y 3.2.4, respectivamente.

Proposición 5 (Consecuencia Fuerte vs. Consecuencia Débil) *Para cualquier $CFLP(\mathcal{D})$-programa $\mathcal{P}$ y para cualquier c-hecho φ se tiene que: $\mathcal{P} \models^s_{\mathcal{D}} \varphi \Rightarrow \mathcal{P} \models^w_{\mathcal{D}} \varphi$. El recíproco es falso en algunos casos.*

Demostración 7 *Asumamos que $\mathcal{P} \models^s_{\mathcal{D}} \varphi$. Por la* Propiedad de Completitud *del Teorema 4 podemos deducir que $\mathcal{P} \vdash_{\mathcal{D}} \varphi$, lo cual implica que $\mathcal{P} \models^w_{\mathcal{D}} \varphi$ por la* Propiedad de Corrección *del Teorema 5. Por otra parte, en la demostración del Teorema 5 dada en el Apéndice A se muestran un $CFLP(\mathcal{R})$-programa $\mathcal{P}$ y una c-sentencia no cerrada φ tales que $\mathcal{P} \models^w_{\mathcal{R}} \varphi$ y $\mathcal{P} \not\vdash_{\mathcal{R}} \varphi$, lo que es lo mismo que decir $\mathcal{P} \not\models^s_{\mathcal{R}} \varphi$ debido al Teorema 4.*

□

Proposición 6 (Modelos Mínimos Fuertes vs. Modelos Mínimos Débiles) *Para cualquier $CFLP(\mathcal{D})$-programa $\mathcal{P}$ se tiene que $\mathcal{W}_{\mathcal{P}} \subseteq \mathcal{S}_{\mathcal{P}}$. La inclusión es estricta en algunos casos.*

Demostración 8 *La inclusión $\mathcal{W}_{\mathcal{P}} \subseteq \mathcal{S}_{\mathcal{P}}$ ya se ha demostrado al final de la Subsección 3.2.4. Como un contraejemplo para la inclusión contraria, vamos a considerar un dominio de restricciones arbitrario $\mathcal{D}$ con universo de valores $\mathcal{U}_{\mathcal{D}}$, el $CFLP(\mathcal{D})$-programa $\mathcal{P}$ formado por una sola regla de programa $id\ X \rightarrow X$ definiendo la función identidad, y la c-interpretación $\mathcal{I} = cl_{\mathcal{D}}(\{\ id\ t \rightarrow t \mid t \in \mathcal{U}_{\mathcal{D}}\ \})$. Se tiene entonces que el c-hecho $\varphi = (id\ X \rightarrow X)$ no pertenece a $\mathcal{I}$, puesto que no es ni un c-hecho trivial ni se sigue por $\mathcal{D}$-implicación de los c-hechos cerrados usados para definir $\mathcal{I}$. Por otra parte, φ pertenece a $\mathcal{S}_{\mathcal{P}}$ por la* Propiedad de Canonicidad *del Teorema 4, debido a que $\mathcal{P} \vdash_{\mathcal{D}} \varphi$ es obviamente cierto. Por tanto, $\mathcal{S}_{\mathcal{P}} \not\subseteq \mathcal{I}$. Sin embargo, $\mathcal{W}_{\mathcal{P}} \subseteq \mathcal{I}$ se cumple por el Teorema 2, debido a que $\mathcal{I}$ es claramente un modelo débil de $\mathcal{P}$. A partir de $\mathcal{S}_{\mathcal{P}} \not\subseteq \mathcal{I}$ y de $\mathcal{W}_{\mathcal{P}} \subseteq \mathcal{I}$ se concluye que $\mathcal{S}_{\mathcal{P}} \not\subseteq \mathcal{W}_{\mathcal{P}}$.*

□

Capítulo 4

Resolución de objetivos en el esquema $CFLP(\mathcal{D})$

El propósito de este capítulo es el de proporcionar varios métodos de resolución de objetivos que puedan ser usados como base formal para describir la semántica operacional de los $CFLP(\mathcal{D})$-programas. La introducción en el capítulo anterior de una semántica declarativa precisa mediante la lógica para la reescritura con restricciones $CRWL(\mathcal{D})$ nos permite ahora formalizar de una manera clara y concisa las nociones necesarias de *objetivo*, *respuesta* y de *solución* en las que se van a apoyar nuestros métodos de resolución de objetivos y sus correspondientes propiedades. En primer lugar, presentaremos un cálculo formal de transformación de objetivos denominado $CLNC(\mathcal{D})$, basado en *estrechamiento perezoso con restricciones*, y que resulta ser *correcto* y *fuertemente completo* con respecto a $CRWL(\mathcal{D})$. En segundo lugar, presentaremos un refinamiento del cálculo $CLNC(\mathcal{D})$ que utiliza un mecanismo especial de control denominado *árbol definicional* con el que es posible asegurar que todos los pasos de estrechamiento perezoso que se ejecutan en las computaciones son realmente necesarios en el cómputo de respuestas. El cálculo así obtenido, denominado $CDNC(\mathcal{D})$, conserva las propiedades de corrección y de completitud fuerte con respecto a $CRWL(\mathcal{D})$, además de incorporar propiedades de *optimalidad* que han sido previamente demostradas para las estrategias de estrechamiento necesario y de estrechamiento dirigido por demanda con árboles definicionales. Más aún, demostramos que el cálculo $CDNC(\mathcal{D})$ puede ser aplicado a cualquier $CFLP(\mathcal{D})$-programa aunque éste no admita directamente un árbol definicional gracias a la aplicación de un algoritmo de transformación de programas con restricciones, el cual permite preservar la semántica declarativa de $CRWL(\mathcal{D})$. Por todas estas propiedades teóricas, el cálculo $CDNC(\mathcal{D})$ resulta ser de gran utilidad como especificación concreta de la semántica operacional y de su correspondiente implementación en sistemas *CFLP* existentes como $\mathcal{TOY}$ o *Curry*.

4.1. Resolución de objetivos basada en estrechamiento

Como se ha explicado en capítulos anteriores, el esquema de programación lógico funcional con restricciones $CFLP(\mathcal{D})$ trata de proporcionar una semántica declarativa y operacional, lo más clara y concisa posible, a la combinación de los paradigmas de programación lógico funcional y de programación lógica con restricciones. En los últimos 20 años se han investigado una gran variedad de métodos de resolución de objetivos que sirven de fundamento a la semántica operacional de la programación lógico funcional. Muchos de ellos están basados en la técnica denominada *estrechamiento* (del inglés, *narrowing*), una combinación de reescritura y de unificación que fue propuesta originalmente como una herramienta de *Demostración Automática de Teoremas* en [Sla74, Lan75, Fay79, Hul80]. La literatura habitual sobre el estrechamiento como método de resolución de objetivos para programación lógico funcional incluye resultados que establecen la corrección y la completitud de diversas variantes del estrechamiento, con respecto a diferentes formalizaciones de la semántica declarativa pretendida de los programas; una descripción detallada de las diferentes *estrategias de estrechamiento* que han sido propuestas en las últimas décadas puede ser consultada en las siguientes referencias [DO90, Han94b, MH94, Han07a].

En este capítulo vamos a presentar varios métodos de resolución de objetivos basados en estrechamiento que son correctos y completos con respecto a la lógica para la reescritura con restricciones $CRWL(\mathcal{D})$ del Capítulo 3, y que son relevantes para el esquema $CFLP(\mathcal{D})$, y particularmente para el paradigma lógico funcional, puesto que la semántica declarativa de programas lógico funcionales puede ser caracterizada mediante el uso de la lógica para la reescritura *CRWL*, la cual puede verse como un caso particular de $CRWL(\mathcal{D})$ donde las únicas restricciones existentes en el dominio $\mathcal{D}$ son de igualdad estricta.

Una variedad interesante de los métodos de resolución de objetivos basada en estrechamiento es la que se presenta en la forma de *cálculos formales*, constituidos por reglas de transformación de objetivos. Estas reglas se aplican sucesivamente a un objetivo inicial con el fin de transformarlo en varios objetivos en forma resuelta que representan diferentes respuestas computadas. Éste es el caso del *cálculo de estrechamiento perezoso CLNC* (del inglés, *Constructor-based Lazy Narrowing Calculus*) presentado en [GHLR99] y de su extensión de orden superior *HOLNC* presentada en [GHR97]. Ambos cálculos son correctos y completos con respecto a la semántica proporcionada por la lógica para la reescritura *CRWL*. El lector interesado puede referirse a [Rod01] para una presentación global de estos cálculos y una comparativa con otros métodos de estrechamiento relacionados.

Además de ser lógicamente correcto y completo, un método de estrechamiento debe satisfacer otras propiedades adicionales con el fin de que pueda ser considerado como una adecuada referencia y guía para la implementación eficiente de sistemas reales de resolución de objetivos. Los métodos de estrechamiento con la propiedad de ejecutar sólo *pasos necesarios* son particularmente interesantes en este sentido.

Intuitivamente, los pasos de estrechamiento necesarios son todos aquellos pasos de estrechamiento que son demandados, y que por tanto resultan imprescindibles, para poder completar satisfactoriamente el cómputo global. Esta idea ha sido investigada de un modo más pragmático y orientado a la implementación en [LLR93] y ha sido formalizada teóricamente en [AEH94, AEH00], donde se ha demostrado que la denominada *estrategia de estrechamiento necesario NN* (del inglés, *Needed Narrowing strategy*) posee un número significativo de propiedades, tales como la corrección, la completitud, y adicionalmente, *propiedades de optimalidad*.

Sin embargo, el estrechamiento necesario, tal y como se presenta en [AEH94, AEH00], no se adecúa a la semántica declarativa dada para *CRWL*. Como una alternativa eficiente, el *cálculo de estrechamiento dirigido por demanda DNC* (del inglés, *Demand-driven Narrowing Calculus*) presentado en [Vad03a, Vad03b], y su reciente extensión de orden superior *HOLN-DT* propuesta en [Vad07], sí permiten mantener las propiedades de optimalidad del estrechamiento necesario además de seguir siendo cálculos correctos y completos con respecto a la semántica de *CRWL*, como ya ocurría con los cálculos *CLNC* y *HOLNC*, respectivamente.

Los cálculos *CLNC* y *DCN* han sido tomados como un punto de partida para el diseño de cálculos de estrechamiento perezoso con restricciones que son correctos y completos con respeto a la semántica $CRWL(\mathcal{D})$, es decir, la semántica declarativa para $CFLP(\mathcal{D})$ programas que se ha descrito en el Capítulo 3. En primer lugar, *CLNC* ha sido extendido en [LRV04b] a través del *cálculo de estrechamiento perezoso con restricciones* $CLNC(\mathcal{D})$ (del inglés, *Constraint Lazy Narrowing Calculus*), el cual incorpora una estrategia de estrechamiento perezoso con restricciones; en segundo lugar, el cálculo *DNC* ha sido extendido en [Vad05] mediante el *cálculo de estrechamiento con restricciones dirigido por demanda* $CDNC(\mathcal{D})$ (del inglés, *Constraint Demand-driven Narrowing Calculus*), el cual disfruta de las propiedades de optimalidad del estrechamiento necesario además de las ya conocidas propiedades de corrección y de completitud con respecto a $CRWL(\mathcal{D})$.

Los cálculos $CLNC(\mathcal{D})$ y $CDNC(\mathcal{D})$ usan la lógica para la reescritura $CRWL(\mathcal{D})$ presentada en el Capítulo 3 para definir conjuntos de soluciones asociados a objetivos, los cuales resultan imprescindibles como base para la formulación adecuada de las propiedades de corrección y completitud. Ambos cálculos combinan además el estrechamiento perezoso (respectivamente el estrechamiento necesario) con la resolución de restricciones, gracias a las propiedades postuladas para un resolutor de restricciones sobre un dominio $\mathcal{D}$ en el Capítulo 2, y que resultan también necesarias para probar la corrección y la completitud de estos cálculos. Estas propiedades han sido obtenidas generalizando las ideas introducidas en trabajos previos como [Lop92, AGL94, LS03].

De entre los cálculos de resolución de objetivos que proponemos en este capítulo, el cálculo $CDNC(\mathcal{D})$ es el que resulta más adecuado para formalizar el comportamiento de sistemas de programación $CFLP(\mathcal{D})$ actuales como $\mathcal{TOY}$ o *Curry*. Para el caso particular de las restricciones de dominio finito, esta adecuación entre teoría

e implementación ha sido demostrada en la publicación [EV05]. En el Apéndice B mostraremos, mediante un ejemplo concreto, cómo las computaciones que se hacen mediante el cálculo $CDNC(\mathcal{FD})$ proporcionan un modelo adecuado para las computaciones actuales en el sistema $\mathcal{TOY}(\mathcal{FD})$ [FHS03b, FHS03c].

4.2. El cálculo de estrechamiento perezoso $CLNC(\mathcal{D})$

En esta sección vamos a presentar el *cálculo de estrechamiento perezoso con restricciones* $CLNC(\mathcal{D})$ sobre un dominio de restricciones $\mathcal{D}$ paramétricamente dado, con el fin de resolver objetivos formulados en base a $CFLP(\mathcal{D})$-programas, usando ideas y técnicas de cálculos de estrechamiento perezoso anteriores para *FLP* sin restricciones [GHLR99, GHR01, Vad03b] y lenguajes *CFLP* [Lop92, AGL94, ALR99]. Vamos a dar primero una definición precisa para la clase particular de objetivos admisibles, respuestas computadas y soluciones con las que vamos a trabajar en lo sucesivo.

4.2.1. Respuestas y soluciones de objetivos

La siguiente definición permite representar, con un mayor nivel de detalle, los objetivos para $CFLP(\mathcal{D})$-programas que han sido introducidos en la Sección 2.6, de forma que ahora sea posible describir mediante ellos cualquier posible estado del cómputo durante el proceso de resolución de objetivos. Más concretamente, en vez de representar simplemente los objetivos en la forma $G \equiv P \;\square\; \Delta$, optamos ahora por hacer explícitas las variables intermedias que pueden ser introducidas en algún momento de la computación como variables cuantificadas existencialmente $\exists\overline{U}$, mantenemos la parte P de las producciones con las mismas condiciones de admisibilidad que exigíamos en la Sección 2.6, distinguimos separadamente la parte C de las restricciones de Δ que no son restricciones primitivas de la parte Π que sí que lo son, e introducimos finalmente una sustitución idempotente σ para ir almacenando los vínculos de todas aquellas variables cuyo valor ya ha sido calculado y forma parte de las respuestas que se van a computar.

Definición 17 (Objetivos Admisibles para $CFLP(\mathcal{D})$-programas) *Un* objetivo *para un $CFLP(\mathcal{D})$-programa dado tiene la siguiente forma general $G \equiv \exists\overline{U}.\; P \;\square\; C \;\square\; \Pi \;\square\; \sigma$, donde el símbolo $\square$ se interpreta como conjunción, y:*

- $\overline{U} =_{def} evar(G)$ *es el conjunto de las denominadas* variables existenciales *del objetivo G. Se trata de variables intermedias que podrían vincularse a patrones parciales en una solución. Se define también el conjunto $fvar(G) =_{def} var(G) \backslash evar(G)$ de todas aquellas variables denominadas* variables libres *del objetivo G.*

- $P \equiv e_1 \rightarrow t_1, \ldots, e_n \rightarrow t_n$ *es una conjunción finita de producciones, donde* $e_i \in Exp_{\mathcal{D}}$ *y* $t_i \in Pat_{\mathcal{D}}$ *para todo* $1 \leq i \leq n$. *El conjunto de* variables producidas *de* G *se define como* $pvar(P) =_{def} var(t_1) \cup \ldots \cup var(t_n)$.

- $C \equiv \delta_1, \ldots, \delta_k$ *es una conjunción finita de restricciones atómicas (posiblemente incluyendo apariciones de símbolos de función definida).*

- $\Pi \equiv \pi_1, \ldots, \pi_l$ *es una conjunción finita de restricciones atómicas primitivas, denominada* restricción respuesta.

- σ *es una sustitución idempotente denominada* sustitución respuesta, *tal que* $vdom(\sigma) \cap var(P \ \square \ C \ \square \ \Pi) = \emptyset$. *Al par* $(\Pi \,\square\, \sigma)$ *se le denomina* almacén de restricciones *del objetivo* G.

Adicionalmente, cualquier objetivo admisible *debe satisfacer las siguientes condiciones de admisibilidad, denominadas* invariantes del objetivo*:*

LN *Cada variable producida es producida sólo una vez, es decir, la tupla* $t_1, \ldots, t_n$ *debe ser* lineal.

EX *Todas las variables producidas deben ser existenciales, es decir,* $pvar(P) \subseteq evar(G)$.

NC *El cierre transitivo de la relación de producción* $\gg_P$ *(definido en la Sección 2.6) debe ser irreflexivo, o equivalentemente, un orden parcial estricto.*

SL *Ninguna variable producida entra en la sustitución respuesta, es decir,* $var(\sigma) \cap pvar(P) = \emptyset$.

Un objetivo admisible $G \equiv \exists \overline{U}.\ P \ \square \ C \ \square \ \Pi \ \square \ \sigma$ se denomina *objetivo en forma resuelta* si y sólo si P y C son vacíos y $\Pi \ \square \ \sigma$ es un almacén de restricciones en forma resuelta con respecto a un conjunto vacío de variables críticas en el sentido explicado en la Definición 5 del Capítulo 2. Un *objetivo inicial* puede ser cualquier objetivo admisible.

Definición 18 (Respuestas) *Una* respuesta *para un objetivo admisible* $G \equiv \exists \overline{U}.\ P \ \square \ C \ \square \ S \ \square \ \sigma$ *y un CFLP*$(\mathcal{D})$*-programa dado* $\mathcal{P}$ *es un almacén de restricciones de la forma* $\Pi \ \square \ \theta$, *donde* $\Pi \subseteq PCon_{\mathcal{D}}$ *es una conjunción finita de restricciones atómicas primitivas,* $\theta \in Sub_{\mathcal{D}}$ *es una sustitución idempotente tal que* $vdom(\theta) \cap var(\Pi) = \emptyset$, *y existe una sustitución* $\theta' =_{\setminus evar(G)} \theta$ *satisfaciendo las siguientes condiciones:*

- $\mathcal{P} \vdash_{\mathcal{D}} (P \ \square \ C)\theta' \Leftarrow \Pi$,

- $\Pi \models_{\mathcal{D}} S\theta'$,

- $X\theta' \equiv t\theta'$ *para cada* $\{X \mapsto t\} \in \sigma$, *abreviado como* $\theta' \in Sol(\sigma)$.

Un testigo $\mathcal{M}$ *para el hecho de que el almacén* $\Pi \ \square \ \theta$ *sea una respuesta de* G *se define como un multiconjunto formado por todas las CRWL*$(\mathcal{D})$*-pruebas mencionadas más arriba. Escribiremos* $Ans_{\mathcal{P}}(G)$ *para designar el conjunto de todas las respuestas de* G. *Una respuesta* $(\Pi \ \square \ \theta) \in Ans_{\mathcal{P}}(G)$ *se denomina* trivial *si* $Insat_{\mathcal{D}}(\Pi)$ *y* no-trivial *en otro caso.*

Definición 19 (Soluciones) *Sea* $G \equiv \exists \overline{U}.\ P \ \square \ C \ \square \ S \ \square \ \sigma$ *un objetivo admisible para un CFLP*$(\mathcal{D})$*-programa* $\mathcal{P}$ *dado. Decimos que una valoración* $\mu \in Val_{\mathcal{D}}$ *es una* solución *del objetivo* G *si existe alguna valoración* $\mu' =_{\setminus evar(G)} \mu$ *que satisface las siguientes condiciones:*

- $\mathcal{P} \vdash_{\mathcal{D}} (P \ \square \ C)\mu'$,
- $\models_{\mathcal{D}} S\mu'$ *(es decir,* $\mu' \in Sol_{\mathcal{D}}(S)$*),*
- $X\mu' \equiv t\mu'$ *para cada* $\{X \mapsto t\} \in \sigma$, *abreviado como* $\mu' \in Sol(\sigma)$.

Escribimos $Sol_{\mathcal{P}}(G)$ *para representar el conjunto de todas las soluciones de* G. *Análogamente, definimos el conjunto de soluciones para una respuesta* $\Pi \ \square \ \theta$ *como:*

$$Sol_{\mathcal{D}}(\Pi \ \square \ \theta) =_{def} \{\ \mu \in Val_{\mathcal{D}} \mid \mu \in Sol_{\mathcal{D}}(\Pi) \cap Sol(\theta)\ \}.$$

A partir de las Definiciones 18 y 19 es fácil demostrar que la noción de solución es un caso particular de la noción de respuesta para un objetivo. Más formalmente, si G es un objetivo admisible y $\mu \in Val_{\mathcal{D}}$ entonces

$$\mu \in Sol_{\mathcal{P}}(G) \Leftrightarrow (\emptyset \ \square \ \mu) \in Ans_{\mathcal{P}}(G).$$

Razonando por inducción sobre la estructura sintáctica de G se puede demostrar también el siguiente resultado:

Lema 8 (Lema de Sustitución) *Asumamos un CFLP*$(\mathcal{D})$*-programa* $\mathcal{P}$, *un objetivo admisible* G *para* $\mathcal{P}$, *una sustitución* $\sigma \in Sub_{\mathcal{D}}$ *y una valoración* $\eta \in Val_{\mathcal{D}}$. *Se cumple entonces la siguiente equivalencia:*

$$\eta \in Sol_{\mathcal{P}}(G\sigma) \Leftrightarrow \sigma\eta \in Sol_{\mathcal{P}}(G)$$

En vista del Teorema 4 de la Subsección 3.4.1, el Lema 8 se puede considerar como una extensión del Lema 6 dado en la Subsección 3.1.4 para objetivos, enfocado al caso en el que $\mathcal{I}$ es $\mathcal{S}_{\mathcal{P}}$. Una consecuencia útil del Lema 8 es la siguiente:

Corolario 7 *Sean* $\mathcal{P}$, G, σ *y* η *como en el enunciado del Lema 8. Si se cumple que* $\eta \in Sol(\sigma)$ *y* $\eta \in Sol_{\mathcal{P}}(G\sigma)$, *entonces también se tiene que* $\eta \in Sol_{\mathcal{P}}(G)$.

Demostración 9 *Por definición, la hipótesis $\eta \in Sol(\sigma)$ significa que $\sigma\eta = \eta$. Por otro lado, la hipótesis $\eta \in Sol_{\mathcal{P}}(G\sigma)$ equivale a $\sigma\eta \in Sol_{\mathcal{P}}(G)$ por el Lema 8. Al ser $\sigma\eta = \eta$, se concluye que $\eta \in Sol_{\mathcal{P}}(G)$, como se quería demostrar.*

□

Otro resultado técnico que interesa enunciar para uso posterior es el siguiente lema, cuya demostración se reduce a una comprobación trivial a partir de la Definición 19:

Lema 9 (Lema de Coincidencia) *Sea G un objetivo admisible para un CFLP($\mathcal{D}$)-programa $\mathcal{P}$, y sean $\eta, \eta' \in Val_{\mathcal{D}}$ dos valoraciones tales que $\eta =_{fvar(G)} \eta'$. Se cumple entonces que $\eta \in Sol_{\mathcal{P}}(G)$ si y solo si $\eta' \in Sol_{\mathcal{P}}(G)$.*

El Lema 8, su Corolario 7 y el Lema 9 se utilizarán implícitamente en las demostraciones del Apéndice A.

Finalmente, otra relación útil entre respuestas y soluciones viene dada a través de la siguiente proposición.

Proposición 8 *Sea $G \equiv \exists\overline{U}.\ P \ \square\ C \ \square\ S \ \square\ \sigma$ un objetivo admisible para un CFLP($\mathcal{D}$)-programa $\mathcal{P}$ dado y $(\Pi \ \square\ \theta) \in Ans_{\mathcal{P}}(G)$ una respuesta para G. Entonces, $Sol_{\mathcal{D}}(\Pi \ \square\ \theta) \subseteq Sol_{\mathcal{P}}(G)$. Más aún, si G es una forma resuelta entonces $(S \ \square\ \sigma) \in Ans_{\mathcal{P}}(G)$.*

Demostración 10 *Puesto que $(\Pi \ \square\ \theta) \in Ans_{\mathcal{P}}(G)$, existe algún θ' satisfaciendo:*

(a) $\theta' =_{\backslash evar(G)} \theta$,

(b) $\mathcal{P} \vdash_{\mathcal{D}} (P \ \square\ C)\theta' \Leftarrow \Pi$,

(c) $\Pi \models_{\mathcal{D}} S\theta'$,

(d) $\theta' \in Sol(\sigma)$.

Para probar que $Sol_{\mathcal{D}}(\Pi \ \square\ \theta) \subseteq Sol_{\mathcal{P}}(G)$, asumimos cualquier valoración μ tal que

(1) $\mu \in Sol_{\mathcal{D}}(\Pi)$,

(2) $\mu \in Sol(\theta)$.

Entonces, $\mu \in Sol_{\mathcal{P}}(G)$ se cumple debido a que la valoración $\mu' = \theta'\mu$ verifica:

(a') $\theta'\mu =_{\backslash evar(G)} \mu$, *debido a (a) y $\theta\mu = \mu$ (que se sigue de (2)),*

(b') $\mathcal{P} \vdash_{\mathcal{D}} (P \ \square\ C)\theta'\mu$, *debido a (b) y a la* Propiedad de Implicación *del Lema 7 en el Capítulo 3 (obsérvese que $(P \ \square\ C)\theta' \Leftarrow \Pi \succcurlyeq_{\mathcal{D}} (P \ \square\ C)\theta'\mu$ se sigue de (1)),*

(c') $\theta'\mu \in Sol_{\mathcal{D}}(S)$*, o equivalentemente,* $\mu \in Sol_{\mathcal{D}}(S\theta')$*, debido a (1) y (c),*

(d') $\theta'\mu \in Sol(\sigma)$*, debido a (d).*

Para probar la segunda parte de la proposición, vamos a asumir que P *y* C *son vacíos. Entonces,* $(S \;\square\; \sigma) \in Ans_{\mathcal{P}}(G)$ *se cumple debido a que* $\sigma' = \sigma$ *verifica trivialmente* $\sigma' =_{\setminus evar(G)} \sigma$*, y también* $\mathcal{P} \vdash_{\mathcal{D}} (P \;\square\; C)\sigma \Leftarrow S$ *(puesto que* $P \;\square\; C$ *es vacío);* $S \models_{\mathcal{D}} S\sigma$ *(puesto que* $S\sigma = S$*); y* $\sigma \in Sol(\sigma)$ *(trivialmente).*

□

4.2.2. Variables demandadas en un objetivo

De manera similar a otros trabajos en programación lógico funcional sin restricciones [GHLR99, GHR01, Vad03a, Vad03b], el cálculo $CLNC(\mathcal{D})$ hace uso de una noción especial de *variable demandada* para poder realizar correctamente una *evaluación perezosa* de las expresiones involucradas en los objetivos, pero ahora con respecto a un almacén de restricciones sobre el dominio $\mathcal{D}$. Intuitivamente, las producciones $e \to X$ en G, donde e no es un patrón, no propagan el vínculo $\{X \mapsto e\}$. En su lugar, debe efectuarse la evaluación de e bajo el supuesto de que la variable X *esté demandada en* G. El resultado que se obtenga de la evaluación de e será entonces compartido por todas las apariciones de X en el objetivo G.

Definición 20 (Variables Demandadas en un $CFLP(\mathcal{D})$-objetivo) *Sea* $G \equiv \exists\overline{U}.\; P \;\square\; C \;\square\; S \;\square\; \sigma$ *un objetivo admisible para un* $CFLP(\mathcal{D})$*-programa dado y* $X \in var(G)$*. Decimos que* X *es una* variable demandada *en* G *si y sólo si*

(a) $X \in odvar_{\mathcal{D}}(S)$*, o bien,*

(b) existe alguna producción $(X\overline{a}_k \to t) \in P$ *tal que* $t \notin \mathcal{V}ar$*, o* $k > 0$ *y* t *es una variable demandada en* G*.*

Escribimos $dvar_{\mathcal{D}}(G)$ *(o más precisamente,* $dvar_{\mathcal{D}}(P \;\square\; S)$*) para representar el conjunto de variables demandadas en el objetivo* G*.*

4.2.3. Estrechamiento perezoso con restricciones

El cálculo $CLNC(\mathcal{D})$ está constituido por un conjunto de *reglas de transformación* de objetivos admisibles. Cada regla de transformación es de la forma $G \Vdash G'$, y especifica uno de los posibles modos de realizar un paso de resolución de objetivos. Escribimos $G \Vdash_{\mathbf{RL}} G'$ para indicar que $G \Vdash G'$ por medio de la aplicación de la regla de transformación **RL** del cálculo $CLNC(\mathcal{D})$. Las *derivaciones* son secuencias de $\Vdash$-pasos. Como en el caso de las *SLD*-derivaciones con restricciones para $CLP(\mathcal{D})$-programas [JMMS98], las *derivaciones con éxito* terminarán eventualmente con un

objetivo en forma resuelta. Las *derivaciones fallidas* (finalizadas con un *objetivo trivialmente inconsistente* representado mediante el símbolo ■) y las *derivaciones infinitas* también son permitidas.

Las reglas de transformación de objetivos correspondientes a producciones están diseñadas con el propósito de modelar el comportamiento del estrechamiento perezoso con restricciones y compartición (*sharing*), involucrando funciones primitivas, funciones definidas de order superior posiblemente indeterministas, y variables funcionales.

DC Descomposición

$$\exists\overline{U}.\ h\overline{e}_m \to h\overline{t}_m, P \sqcup C \sqcup S \sqcup \sigma \Vdash_{DC_1} \exists\overline{U}.\ \overline{e_m \to t_m}, P \sqcup C \sqcup S \sqcup \sigma$$

si $h\overline{e}_m$ es pasivo.

$$\exists\overline{U}.\ u \to u, P \sqcup C \sqcup S \sqcup \sigma \Vdash_{DC_2} \exists\overline{U}.\ P \sqcup C \sqcup S \sqcup \sigma \qquad \text{si } u \in \mathcal{B}^{\mathcal{D}}.$$

SP Producción Simple

$$\exists[X], \overline{U}.\ X \to t, P \sqcup C \sqcup S \sqcup \sigma \Vdash_{SP_{[1],2}} \exists\overline{U}.\ (P \sqcup C \sqcup S)\sigma_0 \sqcup \sigma[\sigma_0]$$

si $t \notin \mathcal{V}ar, [X \notin \overline{U}]$ y $\sigma_0 = \{X \mapsto t\}$.

$$\exists X, \overline{U}.\ t \to X, P \sqcup C \sqcup S \sqcup \sigma \Vdash_{SP_3} \exists\overline{U}.\ (P \sqcup C \sqcup S)\sigma_0 \sqcup \sigma$$

si $t \in Pat_{\mathcal{D}}$ y $\sigma_0 = \{X \mapsto t\}$.

IM Imitación

$$\exists X, \overline{U}.\ h\overline{e}_m \to X, P \sqcup C \sqcup S \sqcup \sigma \Vdash_{IM} \exists\overline{X}_m, \overline{U}.\ (\overline{e_m \to X_m}, P \sqcup C \sqcup S)\sigma_0 \sqcup \sigma$$

si $h\overline{e}_m \notin Pat_{\mathcal{D}}$ es pasivo, $X \in dvar_{\mathcal{D}}(P \sqcup S)$, y $\sigma_0 = \{X \mapsto h\overline{X}_m\}$ con $\overline{X}_m$ variables nuevas tales que $h\overline{X}_m \in Pat_{\mathcal{D}}$.

EL Eliminación

$$\exists X, \overline{U}.\ e \to X, P \sqcup C \sqcup S \sqcup \sigma \Vdash_{EL} \exists\overline{U}.\ P \sqcup C \sqcup S \sqcup \sigma$$

si $X \notin var(P \sqcup C \sqcup S \sqcup \sigma)$.

PF Función Primitiva

$$\exists\overline{U}.\ p\overline{e}_n \to t, P \,\square\, C \,\square\, S \,\square\, \sigma \Vdash_{PF} \exists\overline{X}_q, \overline{U}.\ \overline{e_q \to X_q}, P \,\square\, C \,\square\, p\overline{t}_n \to! \ t, S \,\square\, \sigma$$

si $p \in PF^n$, $t \notin \mathcal{V}ar$ o $t \in dvar_{\mathcal{D}}(P \,\square\, S)$, y $\overline{X}_q$ son nuevas variables ($0 \leq q \leq n$ es el número de $e_i \notin Pat_{\mathcal{D}}$) tal que $t_i \equiv X_j$ $(0 \leq j \leq q)$ si $e_i \notin Pat_{\mathcal{D}}$ y $t_i \equiv e_i$ en otro caso para cada $1 \leq i \leq n$.

DF Función Definida

$$\exists\overline{U}.\ f\overline{e}_n \to t, P \,\square\, C \,\square\, S \,\square\, \sigma \Vdash_{DF_1} \exists\overline{Y}, \overline{U}.\ \overline{e_n \to t_n}, r \to t, P', P \,\square\, C', C \,\square\, S \,\square\, \sigma$$

$$\exists\overline{U}.\ f\overline{e}_n\overline{a}_k \to t, P \,\square\, C \,\square\, S \,\square\, \sigma \Vdash_{DF_2} \exists X, \overline{Y}, \overline{U}.\ \overline{e_n \to t_n}, r \to X, X\overline{a}_k \to t, P', P \,\square\, C', C \,\square\, S \,\square\, \sigma$$

si $f \in DF^n$ $(k > 0), t \notin \mathcal{V}ar$ o $t \in dvar_{\mathcal{D}}(P \,\square\, S)$ y $R : f\overline{t}_n \to r \Leftarrow P' \,\square\, C'$ es una variante fresca de una regla de programa en $\mathcal{P}$, con $\overline{Y} = var(R)$ y X variables nuevas.

FV Variable Funcional

$$\exists[F], \overline{U}.\ F\overline{e}_q \to t, P \,\square\, C \,\square\, S \,\square\, \sigma \Vdash_{FV_{[1],2}} \exists\overline{X}_p, \overline{U}.\ (h\overline{X}_p\overline{e}_q \to t, P \,\square\, C \,\square\, S)\sigma_0 \,\square\, \sigma[\sigma_0]$$

si $[F \notin pvar(P)], q > 0, t \notin \mathcal{V}ar$ o $t \in dvar_{\mathcal{D}}(P \,\square\, S)$, $\sigma_0 = \{F \mapsto h\overline{X}_p\}$, y $\overline{X}_p$ son variables nuevas tales que $h\overline{X}_p \in Pat_{\mathcal{D}}$.

La notación $\overline{e_m \to t_m}$ permite abreviar la secuencia de producciones $e_1 \to t_1$, ..., $e_m \to t_m$. Algunas reglas del cálculo $CLNC(\mathcal{D})$ usan la notación especial "[...]" para designar una parte opcional del objetivo, presente sólo bajo ciertas condiciones. Por ejemplo, en la regla **Producción Simple**, si se tiene la condición $X \notin \overline{U}$ (y por tanto $X \notin pvar(P)$ por la condición de admisibilidad **EX**) entonces se tiene la regla

$$\exists\overline{U}.\ X \to t, P \,\square\, C \,\square\, S \,\square\, \sigma \Vdash_{SP_1} \exists\overline{U}.\ (P \,\square\, C \,\square\, S)\sigma_0 \,\square\, \sigma\sigma_0$$

y la regla

$$\exists X, \overline{U}.\ X \to t, P \,\square\, C \ \,\square\, S \,\square\, \sigma \Vdash_{SP_2} \exists\overline{U}.\ (P \,\square\, C \,\square\, S)\sigma_0 \,\square\, \sigma$$

en otro caso. Análogamente para las reglas de **Variable Funcional**.

Las reglas de transformación de objetivos del cálculo que permiten manipular restricciones están diseñadas con el propósito de poder combinar restricciones (primitivas o definidas por el usuario) con la acción de un resolutor de restricciones que satisfaga los requisitos dados en el Capítulo 2.

CS Resolución de Restricciones

$$\exists \overline{U}.\ P \,\square\, C \,\square\, S \,\square\, \sigma \Vdash_{CS\{\chi\}} \exists \overline{Y}_i, \overline{U}.\ (P \,\square\, C)\sigma_i \,\square\, S_i \,\square\, \sigma\sigma_i$$

si $\chi = pvar(P)$, S no está en χ-forma resuelta, $solve^{\mathcal{D}}(S, \chi) = \bigvee_{i=1}^{k}(S_i \,\square\, \sigma_i)$, y $\overline{Y}_i$ son las variables nuevas introducidas por el resolutor en $S_i \,\square\, \sigma_i$, para cada $1 \leq i \leq k$.

AC Restricción Atómica

$$\exists \overline{U}.\ P \,\square\ p\overline{e}_n \rightarrow! \ t, C \,\square\, S \,\square\, \sigma \Vdash_{AC} \exists \overline{X}_q, \overline{U}.\ \overline{e_q \rightarrow X_q}, P \,\square\, C \,\square\ p\overline{t}_n \rightarrow! \ t, S \,\square\, \sigma$$

si $p \in PF^n$, $p\overline{e}_n \rightarrow! \ t$ es una restricción atómica, $\overline{X}_q$ son variables nuevas ($0 \leq q \leq n$ es el número de $e_i \notin Pat_{\mathcal{D}}$) tales que $t_i \equiv X_j$ ($0 \leq j \leq q$) si $e_i \notin Pat_{\mathcal{D}}$ y $t_i \equiv e_i$ en otro caso, para cada $1 \leq i \leq n$.

Finalmente, las reglas de fallo se utilizan para la detección de fallos en la resolución de restricciones y la detección de fallos en el proceso de unificación sintáctica de la parte producida del objetivo.

CF Fallo por Conflicto

$$\exists \overline{U}.\ h\overline{e}_p \rightarrow h'\overline{t}_q, P \,\square\, C \,\square\, S \,\square\, \sigma \Vdash_{CF} \blacksquare$$

si $h\overline{e}_p$ es pasivo, y $h \neq h'$ o bien $p \neq q$ (análogamente si en vez de la producción $h\overline{e}_p \rightarrow h'\overline{t}_q$ tuviéramos una producción de la forma $u \rightarrow u'$, $h\overline{e}_p \rightarrow u'$ o bien $u \rightarrow h'\overline{t}_q$, siendo $u, u' \in \mathcal{B}^{\mathcal{D}}$ tales que $u \neq_{\mathcal{B}^{\mathcal{D}}} u'$).

SF Fallo en la Resolución

$$\exists \overline{U}.\ P \,\square\, C \,\square\, S \,\square\, \sigma \Vdash_{SF\{\chi\}} \blacksquare$$

si $\chi = pvar(P)$, S no está en χ-forma resuelta, y $solve^{\mathcal{D}}(S, \chi) = \blacksquare$.

Un convenio importante es que todas las reglas de transformación de objetivos son aplicadas considerando P y C como conjuntos, no como secuencias. Por ejemplo, la regla $\mathbf{DF_1}$ permite seleccionar una producción arbitraria $f\overline{e}_n \rightarrow t$ que aparezca en el objetivo actual.

Finalmente, concluimos esta sección con tres ejemplos de resolución de objetivos que resaltan las principales propiedades del cálculo $CLNC(\mathcal{D})$. En cada paso de transformación de objetivos subrayamos el subobjetivo que ha sido seleccionado.

Ejemplo 11 *Computamos todas las posibles respuestas a partir del objetivo inicial* $G_0 \equiv \exists Ys.\ from\ Y \to Ys\ \square X \neq s(null\ \ Ys)\ \square\ \square \varepsilon$ *usando el CFLP*$(\mathcal{H})$*-programa del Ejemplo 7. Se tiene la siguiente derivación, la cual comienza a partir de* G_0*:*

$\exists Ys.\ from\ Y \to Ys\ \square\ \underline{X \neq s(null\ \ Ys)}\ \square\ \square \varepsilon \vdash_{\mathbf{AC}}$
$\exists R, Ys.\ \underline{s(null\ \ Ys) \to R}, from\ Y \to Ys\ \square\ \square\ \ X \neq R\ \square \varepsilon \vdash_{\mathbf{IM}\{R \mapsto s\ K\}}$
$\exists K, Ys.\ \underline{null\ \ Ys \to K}, from\ Y \to Ys\ \square\ \square\ \underline{X \neq s\ K}\ \square \varepsilon \vdash_{\mathbf{CS}\{K, Ys\}}$

En este momento de la derivación, el resolutor de restricciones proporciona dos posibles alternativas:

$$solve^{\mathcal{H}}(\{X \neq s\ K\}, \{K, Ys\}) = (\square\,\{X \mapsto 0\}) \vee (\{M \neq K\}\,\square\,\{X \mapsto s\ M\})$$

A partir de ellas, hay tres posibles continuaciones para la computación:

(1) $\exists K, Ys.\ \underline{null\ \ Ys \to K}, from\ Y \to Ys\ \square\ \square\ \ \square\,\{X \mapsto 0\} \vdash_{\mathbf{EL}}$
$\exists Ys.\ \underline{from\ Y \to Ys}\ \square\ \square\ \square\,\{X \mapsto 0\} \vdash_{\mathbf{EL}}$
$\square\ \square\ \square\,\{X \mapsto 0\}$

La primera respuesta computada es $S_1\,\square\,\sigma_1 \equiv \square\,\{X \mapsto 0\}$.

(2) $\exists M, K, Ys.\ \underline{null\ \ Ys \to K}, from\ Y \to Ys\,\square\ \square\ \ M \neq K\,\square\,\{X \mapsto s\ M\} \vdash_{\mathbf{DF}}$
$\exists M, K, Ys.\ \underline{Ys \to [\,]}, \underline{s\ 0 \to K}, from\ Y \to Ys\,\square\ \square$
$M \neq K\,\square\,\{X \mapsto s\ M\} \vdash^{2}_{\mathbf{SP}\{Ys \mapsto [\,], K \mapsto s\ 0\}}$
$\exists M.\ \underline{from\ Y \to [\,]}\ \square\ \square\ M \neq s\ 0\,\square\,\{X \mapsto s\ M\} \vdash_{\mathbf{DF}}$
$\exists N, M.\ Y \to N, \underline{[N|from\ (s\ N)] \to [\,]}\ \square\ \square\ \ M \neq s\ 0\,\square\,\{X \mapsto s\ M\} \vdash_{\mathbf{CF}} \blacksquare$

En esta segunda alternativa no se obtiene ninguna respuesta.

(3) $\exists M, K, Ys.\ \underline{null\ \ Ys \to K}, from\ Y \to Ys\,\square\ \square\ \ M \neq K\,\square\,\{X \mapsto s\ M\} \vdash_{\mathbf{DF}}$
$\exists U, Us, M, K, Ys.\ \underline{Ys \to [U|Us]}, \underline{0 \to K}, from\ Y \to Ys\,\square\ \square$
$M \neq K\,\square\,\{X \mapsto s\ M\} \vdash^{2}_{\mathbf{SP}\{Ys \mapsto [U|Us], K \mapsto 0\}}$
$\exists U, Us, M.\ \underline{from\ Y \to [U|Us]}\ \square\ \square\ \ M \neq 0\,\square\,\{X \mapsto s\ M\} \vdash_{\mathbf{DF}}$
$\exists N, U, Us, M.\ \underline{Y \to N}, [N|from\ (s\ N)] \to [U|Us]\,\square\ \square$
$M \neq 0\,\square\,\{X \mapsto s\ M\} \vdash_{\mathbf{SP}\{N\ \mapsto\ Y\}}$
$\exists U, Us, M.\ \underline{[Y|from\ (s\ Y)] \to [U|Us]}\ \square\ \square\ \ M \neq 0\,\square\,\{X \mapsto s\ M\} \vdash_{\mathbf{DC}}$
$\exists U, Us, M.\ \underline{Y \to U}, from\ (s\ Y) \to Us\ \square\ \square\ \ M \neq 0\,\square\,\{X \mapsto s\ M\} \vdash^{2}_{\mathbf{EL}}$
$\exists M.\ \square\ \square\ M \neq 0\,\square\,\{X \mapsto s\ M\}$

La segunda respuesta computada es $S_2\,\square\,\sigma_2 \equiv M \neq 0\,\square\,\{X \mapsto s\ M\}$.

Para este ejemplo, es también posible comprobar que $\Pi \,\square\, \theta \equiv X \neq s\ 0 \ \square\ \{Y \mapsto s\ Z\}$ *es una respuesta correcta de* G_0 *tal que* $Sol_{\mathcal{H}}(\Pi \,\square\, \theta) \subseteq \bigcup_{i=1}^{2} Sol_{\mathcal{H}}(S_i \,\square\, \sigma_i)$*; sin embargo, no existe una única respuesta computada* $S \,\square\, \sigma$ *que verifique que* $Sol_{\mathcal{H}}(\Pi \,\square\, \theta) \subseteq Sol_{\mathcal{H}}(S \,\square\, \sigma)$*. En la Subsección 4.2.4 veremos que esta propiedad es cierta en general a través del Teorema de Completitud del Cálculo* $CLNC(\mathcal{D})$*.*

Ejemplo 12 (Partición de una Lista) *El siguiente ejemplo particiona una lista con sólo un elemento usando el* $CFLP(\mathcal{H})$*-programa split dado en el Ejemplo 7.*

$$
\begin{array}{l}
\square\ \underline{split\ [X] == (Xs,\ Ys)}\ \square\ \square\ \varepsilon \Vdash_{\mathbf{AC}} \\
\exists R_1.\ \underline{split\ [X] \to R_1}\ \square\ \square\ R_1 == (Xs,\ Ys)\ \square\ \varepsilon \Vdash^{*}_{\mathbf{DF}} \\
\exists Ys_1, Zs_1, R, R_1.\ case\ R\ X\ Ys_1\ Zs_1 \to R_1, \\
\qquad\qquad \underline{split\ [\,] \to (Ys_1, Zs_1)}\ \square\ \square\ X == s\ 0 \to\!!\ R, \\
\qquad\qquad \underline{R_1 == (Xs,\ Ys)}\ \square\ \varepsilon \Vdash_{\mathbf{DF}} \\
\exists Ys_1, Zs_1, R, R_1.\ case\ R\ X\ Ys_1\ Zs_1 \to R_1, \\
\qquad\qquad \underline{([\,], [\,]) \to (Ys_1, Zs_1)}\ \square\ \square\ X == s\ 0 \to\!!\ R, \\
\qquad\qquad \underline{R_1 == (Xs,\ Ys)}\ \square\ \varepsilon \Vdash^{*}_{\mathbf{DC},\mathbf{SP}\{Ys_1 \mapsto [\,],\ Zs_1 \mapsto [\,]\}} \\
\exists R, R_1.\ case\ R\ X\ [\,]\ [\,] \to R_1\ \square\ \square\ \underline{X == s\ 0 \to\!!\ R, R_1 == (Xs,\ Ys)}\ \square\ \varepsilon \Vdash_{\mathbf{CS}\{R_1\}}
\end{array}
$$

Ahora, el resolutor de restricciones sobre $\mathcal{H}$ *da tres posibles alternativas*

$$
\begin{array}{l}
solve^{\mathcal{H}}(\{X == s\ 0 \to\!!\ R, R_1 == (Xs,\ Ys)\}, \{R_1\}) = \\
\qquad (\{R_1 == (Xs,\ Ys)\} \,\square\, \{R \mapsto true, X \mapsto s\ 0\}) \ \vee \\
\qquad (\{R_1 == (Xs,\ Ys)\} \,\square\, \{R \mapsto false, X \mapsto 0\}) \ \vee \\
\qquad (\{R_1 == (Xs,\ Ys), M \neq 0\} \,\square\, \{R \mapsto false, X \mapsto s\ M\})
\end{array}
$$

y hay tres posibles continuaciones del cómputo, cada una de ellas con una respuesta computada asociada

(1)
$$
\begin{array}{l}
\exists R_1.\ \underline{case\ true\ (s\ 0)\ [\,]\ [\,] \to R_1}\ \square\ \square\ R_1 == (Xs,\ Ys)\ \square\ \{X \mapsto s\ 0\} \Vdash_{\mathbf{DF}} \\
\exists R_1.\ \underline{([s\ 0], [\,]) \to R_1}\ \square\ \square\ R_1 == (Xs,\ Ys)\ \square\ \{X \mapsto s\ 0\} \Vdash_{\mathbf{SP}} \\
\square\ \square\ \underline{([s\ 0], [\,]) == (Xs,\ Ys)}\ \square\ \{X \mapsto s\ 0\} \Vdash_{\mathbf{CS}\{\}} \\
\square\ \square\ \square\ \{X \mapsto s\ 0, Xs \mapsto [s\ 0],\ Ys \mapsto [\,]\}
\end{array}
$$

Primera respuesta computada: $S_1 \,\square\, \sigma_1 \equiv \square\, \{X \mapsto s\ 0, Xs \mapsto [s\ 0],\ Ys \mapsto [\,]\}$*.*

(2)
$$
\begin{array}{l}
\exists R_1.\ \underline{case\ false\ 0\ [\,]\ [\,] \to R_1}\ \square\ \square\ R_1 == (Xs,\ Ys)\ \square\ \{X \mapsto 0\} \Vdash_{\mathbf{DF}} \\
\exists R_1.\ \underline{([\,], [0]) \to R_1}\ \square\ \square\ R_1 == (Xs,\ Ys)\ \square\ \{X \mapsto 0\} \Vdash_{\mathbf{SP}} \\
\square\ \square\ \underline{([\,], [0]) == (Xs,\ Ys)}\ \square\ \{X \mapsto 0\} \Vdash_{\mathbf{CS}\{\}} \\
\square\ \square\ \square\ \{X \mapsto 0, Xs \mapsto [\,],\ Ys \mapsto [0]\}
\end{array}
$$

Segunda respuesta computada: $S_2 \,\square\, \sigma_2 \equiv \square\, \{X \mapsto 0, Xs \mapsto [\,],\ Ys \mapsto [0]\}$*.*

(3) $\exists M, R_1.\ \underline{case\ false\ (s\ M)\ [\,]\ [\,] \to R_1}\ \square\ \square$
$R_1 == (Xs, Ys), M \neq 0\ \square\ \{X \mapsto s\ M\} \vdash_{\mathbf{DF}}$
$\exists M, R_1.\ \underline{([\,], [s\ M]) \to R_1}\ \square\ \square$
$R_1 == (Xs, Ys), M \neq 0\ \square\ \{X \mapsto s\ M\} \vdash_{\mathbf{SP}}$
$\exists M.\ \square\ \square\ \underline{([\,], [s\ M]) == (Xs, Ys), M \neq 0}\ \square\ \{X \mapsto s\ M\} \vdash_{\mathbf{CS}\{\}}$
$\exists M. \square\ \square\ \underline{M \neq 0}\ \square\ \{X \mapsto s\ M, Xs \mapsto [\,], Ys \mapsto [s\ M]\}$

Tercera respuesta computada: $S_3 \square \sigma_3 \equiv M \neq 0\ \square\ \{X \mapsto s\ M,\ Xs \mapsto [\,],\ Ys \mapsto [s\ M]\}$.

Ejemplo 13 *Cerramos esta sección con un tercer ejemplo que permite ilustrar el proceso de resolución de objetivos mediante la instancia particular $CLNC(\mathcal{FD})$ del cálculo de estrechamiento en su combinación con el resolutor de dominios finitos $solve^{\mathcal{FD}}$ presentado en el Capítulo 2. Computamos todas las respuestas a partir del objetivo inicial* $\square$ *check_list* $(from\ M) < 3\ \square\ \square$ *usando el $CFLP(\mathcal{FD})$-programa dado en el Ejemplo 9. Su resolución corresponde a la siguiente secuencia de reglas de transformación de objetivos:*

$\square\ \underline{check\_list\ (from\ M)\ <\ 3}\ \square\ \square\ \varepsilon \vdash_{\mathbf{AC}}$
$\exists X.\ \underline{check\_list\ (from\ M) \to X}\ \square\ \square\ X < 3\ \square\ \varepsilon \vdash_{\mathbf{DF}}$

En este punto de la derivación, observamos que X es una variable demandada por el almacén de restricciones del objetivo, por lo que es posible aplicar las reglas de programa que definen el símbolo de función check_list. En este caso particular, tenemos varias posibles alternativas debido a la elección indeterminista don't know *de una regla de programa para check_list:*

$\exists X.\ \underline{check\_list\ (from\ M) \to X}\ \square\ \square\ X < 3\ \square\ \varepsilon \vdash_{\mathbf{DF}}$
$\exists X.\ from\ M \to [\,], \underline{0 \to X}\ \square\ \square\ X < 3\ \square\ \varepsilon \vdash_{\mathbf{SP}\{X \mapsto 0\}}$
$from\ M \to [\,]\ \square\ \square\ \underline{0 < 3}\ \square\ \varepsilon \vdash_{\mathbf{CS}\{\}}$

En este momento, el resolutor confirma que $solve^{\mathcal{FD}}(\{0 < 3\}, \{\}) = \blacklozenge\ \square\ \varepsilon$, *luego*

$\underline{from\ M \to [\,]}\ \square\ \square\ \square\ \varepsilon \vdash_{\mathbf{DF}}$
$\underline{[M \mid from\ (M + 1)] \to [\,]}\ \square\ \square\ \square\ \varepsilon \vdash_{\mathbf{CF}} \blacksquare$

La aplicación de la primera regla de programa para check_list conduce a una derivación de fallo sin respuesta. Aplicamos la segunda regla de programa de check_list:

$\exists X.\ \underline{check\_list\ (from\ M) \to X}\ \square\ \square\ X < 3\ \square\ \varepsilon \vdash_{\mathbf{DF}}$
$\exists X', Xs', X.\ from\ M \to [X'|Xs'], \underline{1 \to X}\ \square$
$domain\ [X']\ 1\ 2\ \square\ X < 3\ \square\ \varepsilon \vdash_{\mathbf{SP}\{X \mapsto \mathbf{1}\}}$

$\exists X', Xs'.\ from\ M \to [X'|Xs']\ \square\ \underline{domain\ [X']\ 1\ 2}\ \square\ 1 < 3\ \square\ \varepsilon \vdash_{\mathbf{AC}}$
$\exists X', Xs'.\ from\ M \to [X'|Xs']\ \square\ \square\ \underline{1 < 3,\ domain\ [X']\ 1\ 2}\ \square\ \varepsilon \vdash_{\mathbf{DF}}$
$\exists X', Xs'.\ \underline{[M \mid from\ (M+1)] \to [X'|Xs']}\ \square\ \square\ 1 < 3,\ domain\ [X']\ 1\ 2\ \square\ \varepsilon \vdash_{\mathbf{DC}}$
$\exists X', Xs'.\ \underline{M \to X',\ from\ (M+1) \to Xs'}\ \square\ \square$
$1 < 3,\ domain\ [X']\ 1\ 2\ \square\ \varepsilon \vdash_{\mathbf{SP}\{X' \mapsto M\}}$
$\exists Xs'.\ \underline{from\ (M+1) \to Xs'}\ \square\ \square\ 1 < 3,\ domain\ [M]\ 1\ 2\ \square\ \varepsilon \vdash_{\mathbf{EL}}$
$\square\ \square\ \underline{1 < 3, domain\ [M]\ 1\ 2}\ \square\ \varepsilon \vdash_{\mathbf{CS}(\emptyset)}\ \square\ \square\ M \in [1..2]\ \square\ \varepsilon$

Debido a que $solve^{\mathcal{FD}}(\{1 < 3, domain\ [M]\ 1\ 2\}, \emptyset) = \{M \in [1..2]\}\ \square\ \varepsilon$. *Por tanto, obtenemos la primera respuesta computada:* $\Pi_1\ \square\ \theta_1 \equiv \{M \in [1..2]\}\ \square\ \varepsilon$. *Análogamente, podemos aplicar la tercera regla de programa de check_list:*

$\exists X.\ \underline{check\_list\ (from\ M) \to X}\ \square\ \square\ X < 3\ \square\ \varepsilon \vdash^{*}_{\mathbf{DF}}\ \square\ \square\ M \in [3..4]\ \square\ \varepsilon$

para así obtener la segunda respuesta computada: $\Pi_2\ \square\ \theta_2 \equiv \{M \in [3..4]\}\ \square\ \varepsilon$. *Ninguna respuesta más puede ser computada, porque si aplicamos la cuarta regla de programa de check_list volvemos a obtener una derivación de fallo:*

$\exists X.\ \underline{check\_list\ (from\ M) \to X}\ \square\ \square\ X < 3\ \square\ \varepsilon \vdash_{\mathbf{DF}}$
$\exists X', \overline{Xs', X.\ from\ M \to [X'|Xs']},\ \underline{4 \to X}\ \square\ domain\ [X']\ 5\ 7\ \square\ X < 3\ \square\ \varepsilon \vdash_{\mathbf{SP}\{X \mapsto \mathbf{4}\}}$
$\exists X', Xs'.\ from\ M \to [X'|Xs']\ \square\ \underline{domain\ [X']\ 5\ 7}\ \square\ 4 < 3\ \square\ \varepsilon \vdash_{\mathbf{AC}}$
$\exists X', Xs'.\ from\ M \to [X'|Xs']\ \square\ \square\ \underline{4 < 3,\ domain\ [X']\ 5\ 7}\ \square\ \varepsilon \vdash_{\mathbf{SF}\{X', Xs'\}} \blacksquare$

puesto que $solve^{\mathcal{FD}}(\{4 < 3, domain\ [X']\ 5\ 7\}, \{X', Xs'\}) = \blacksquare$. *Como veremos en la Subsección 4.3.2, la introducción de* árboles definicionales *permite guiar eficientemente la computación, evitando elecciones inadecuadas de reglas de programa. Más aún, veremos en el Apéndice B que estas son exactamente las mismas respuestas computadas que se obtienen en la implementación* $CFLP(\mathcal{FD})$ *de* $\mathcal{TOY}(\mathcal{FD})$.

4.2.4. Resultados de corrección y de completitud fuerte

En esta sección vamos a presentar las principales propiedades del cálculo de resolución de objetivos $CLNC(\mathcal{D})$: la corrección y completitud del cálculo con respecto a la semántica declarativa de $CFLP(\mathcal{D})$-programas formalizada mediante la lógica para la reescritura con restricciones $CRWL(\mathcal{D})$.

Los resultados que vamos a presentar, en especial el denominado *Teorema de Completitud* y su correspondiente *Lema de Progreso* asociado, ponen de manifiesto su dificultad técnica, lo que hace que su demostración resulte ser bastante más complicada que la de otros resultados teóricos similares relacionados con los lenguajes de programación lógico funcional sin restricciones [GHLR99, GHR01, Vad03a, Vad03b]. Se trata sin embargo de resultados teóricos mucho más fuertes y más generales que los que se han obtenido en trabajos previos [Lop92, AGL94, ALR99] para lenguajes

CFLP. Como principal diferencia con respecto al cálculo de estrechamiento perezoso con restricciones en el esquema $CFLP(\mathcal{D}, \mathcal{S}, \mathcal{L})$ [Mar00], en nuestro esquema $CFLP(\mathcal{D})$ se proporciona una semántica lógica en la que las nociones formales de respuesta correcta y de resolutor de restricciones se han fijado con total precisión.

Nuestro primer resultado permite probar la corrección de un solo paso de transformación de objetivos. Este resultado afirma que los pasos de transformación del cálculo $CLNC(\mathcal{D})$ preservan la admisibilidad de los objetivos, fallan sólo en caso de objetivos insatisfactibles y no introducen respuestas nuevas.

Lema 10 (Lema de Corrección) *Sea $\mathcal{P}$ un $CFLP(\mathcal{D})$-programa. El cálculo de resolución de objetivos $CLNC(\mathcal{D})$ verifica las siguientes propiedades:*

(1) Los pasos de transformación del cálculo preservan la admisibilidad de los objetivos: Si $G \Vdash_{CLNC(\mathcal{D})} G'$ y G es un objetivo admisible, entonces G' también es un objetivo admisible. Más aún, $fvar(G') \subseteq fvar(G)$.

(2) Los pasos de transformación fallan sólo en el caso de objetivos insatisfactibles: Si $G \Vdash_{CLNC(\mathcal{D})} \blacksquare$ entonces $Sol_{\mathcal{P}}(G) = \emptyset$ (o equivalentemente, el conjunto de respuestas $Ans_{\mathcal{P}}(G)$ incluye sólo respuestas triviales).

(3) Los pasos de transformación no introducen nuevas respuestas: Si $G \Vdash_{CLNC(\mathcal{D})} G'$ y $(\Pi \square \theta) \in Ans_{\mathcal{P}}(G')$ entonces $(\Pi \square \theta) \in Ans_{\mathcal{P}}(G)$.

La demostración del *Lema de Corrección* se incluye en el Apéndice A. El siguiente *Teorema de Corrección* se deduce fácilmente a partir del *Lema de Corrección*. Este resultado asegura que las respuestas computadas para un objetivo admisible G son respuestas correctas de G en el sentido dado en la Definición 18.

Teorema 6 (Corrección del Cálculo $CLNC(\mathcal{D})$) *Si G_0 es un objetivo inicial y $G_0 \Vdash^*_{CLNC(\mathcal{D})} G_n$, donde $G_n \equiv \exists \overline{U}.\ \square\ \square\ S\ \square\ \sigma$ es un objetivo en forma resuelta, entonces $(S\ \square\ \sigma) \in Ans_{\mathcal{P}}(G_0)$.*

Demostración 11 *Utilizando la Proposición 8 se tiene que $(S\ \square\ \sigma) \in Ans_{\mathcal{P}}(G_n)$. Si ahora aplicamos repetidamente de atrás hacia adelante el apartado (3) del* Lema de Corrección*, obtenemos directamente que $(S\ \square\ \sigma) \in Ans_{\mathcal{P}}(G_0)$.*

□

La completitud del cálculo $CLNC(\mathcal{D})$ se basa en la siguiente idea: siempre que se tenga una respuesta $(\Pi\ \square\ \theta) \in Ans_{\mathcal{P}}(G)$ y G sea un objetivo admisible que no esté ya en forma resuelta, existe una cantidad finita de posibles elecciones locales que permiten efectuar un primer paso de cómputo $G \Vdash G_j$ $(1 \leq j \leq l)$ de forma que los nuevos objetivos G_j que se obtienen están más cerca de ser resueltos y cubren todas las soluciones de la respuesta $\Pi\ \square\ \theta$. Este idea se precisa en el siguiente lema, el cual

hace uso de un sofisticado *orden de progreso* bien fundamentado. Técnicas similares ya han sido utilizadas en [GHLR99, GHR01, Vad03a, Vad03b, Vad07] para probar la completitud de cálculos de estrechamiento perezoso en lenguajes *FLP*. En el contexto actual del esquema $CFLP(\mathcal{D})$, el resolutor $solve^{\mathcal{D}}$ ha de ser también tenido en cuenta. Como consecuencia, el número l de posibles elecciones locales puede ser más grande que 1 en general, y el orden de progreso será en consecuencia técnicamente más complicado que aquellos que se utilizan en los trabajos anteriormente citados.

Definición 21 (Orden de Progreso para $CLNC(\mathcal{D})$**)** *Sea $\mathcal{P}$ un $CFLP(\mathcal{D})$-programa, $G \equiv \exists\overline{U}.\ P \ \square\ C \ \square\ S \ \square\ \sigma$ un objetivo admisible para $\mathcal{P}$, y $\mathcal{M} : (\Pi \square \theta) \in Ans_{\mathcal{P}}(G)$ una respuesta para G con testigo $\mathcal{M}$. Definimos los siguientes tamaños asociados a G y a $\mathcal{M}$:*

- *El* tamaño restringido del testigo $\mathcal{M} = \{\!\{\mathcal{T}_1, \ldots, \mathcal{T}_n\}\!\}$ *(representado por $\mid \mathcal{M} \mid$) es el multiconjunto de números naturales $\{\!\{\mid \mathcal{T}_1 \mid, \ldots, \mid \mathcal{T}_n \mid\}\!\}$, donde $\mid \mathcal{T}_i \mid$ representa el tamaño restringido de cada árbol de prueba $\mathcal{T}_i$ $(1 \leq i \leq n)$, como se definió en la Subsección 3.1.3.*
- *El tamaño $\mid G \mid_1$ es el número de apariciones en G de expresiones de la forma $F\overline{e}_k$ con F una variable y $k > 0$.*
- *El tamaño $\mid G \mid_2$ es el número de apariciones en G de expresiones rígidas y pasivas $h\overline{e}_m$ que no son patrones.*
- *El tamaño $\mid G \mid_3$ es el tamaño sintáctico total de los lados derechos de producciones en G.*
- *El* tamaño restringido del almacén de restricciones *(representado por $\mid S \mid$) es* 1 *si S está en forma resuelta y* 0 *en otro caso.*

Definimos un orden de progreso bien fundamentado *sobre parejas $(G, \mathcal{M})$:*

$$(G, \mathcal{M}) \rhd (G', \mathcal{M}') \Leftrightarrow_{def} (\mid \mathcal{M} \mid, \mid G \mid_1, \mid G \mid_2, \mid G \mid_3, \mid S \mid) \succ_{lex} (\mid \mathcal{M}' \mid, \mid G' \mid_1, \mid G' \mid_2, \mid G' \mid_3, \mid S' \mid)$$

donde $\succ_{lex}$ es el producto lexicográfico de $\succ_{mul} \times >_{\mathbb{N}} \times >_{\mathbb{N}} \times >_{\mathbb{N}} \times >_{\mathbb{N}}$, $\succ_{mul}$ es el orden de multiconjuntos para multiconjuntos sobre los números naturales $\mathbb{N}$, y $>_{\mathbb{N}}$ es el orden usual sobre $\mathbb{N}$. En [BN98] se proporcionan definiciones concretas para las definiciones de todas estas nociones.

La siguiente tabla muestra el comportamiento de las diferentes transformaciones del cálculo $CLNC(\mathcal{D})$ con respecto a las cinco componentes del orden del producto lexicográfico que define $\rhd$.

A continuación ofrecemos el resultado principal que nos va a permitir demostrar la completitud del cálculo $CLNC(\mathcal{D})$:

REGLA	$\mid \mathcal{M} \mid$	$\mid G \mid_1$	$\mid G \mid_2$	$\mid G \mid_3$	$\mid S \mid$
DC	$\succeq_{mul}$	$\geqslant_{\mathbb{N}}$	$\geqslant_{\mathbb{N}}$	$>_{\mathbb{N}}$	
SP	$\succeq_{mul}$	$\geqslant_{\mathbb{N}}$	$\geqslant_{\mathbb{N}}$	$>_{\mathbb{N}}$	
IM	$\succeq_{mul}$	$\geqslant_{\mathbb{N}}$	$>_{\mathbb{N}}$		
EL	$\succeq_{mul}$	$\geqslant_{\mathbb{N}}$	$\geqslant_{\mathbb{N}}$	$>_{\mathbb{N}}$	
PF	$\succ_{mul}$				
DF	$\succ_{mul}$				
FV	$\succeq_{mul}$	$>_{\mathbb{N}}$			
CS	$\succeq_{mul}$	$\geqslant_{\mathbb{N}}$	$\geqslant_{\mathbb{N}}$	$\geqslant_{\mathbb{N}}$	$>_{\mathbb{N}}$
AC	$\succ_{mul}$				

Figura 4.1: El orden de progreso $\rhd$

Lema 11 (Lema de Progreso) *Asumamos un objetivo admisible G que no esté en forma resuelta y una respuesta no-trivial con testigo $\mathcal{M} : (\Pi \square \theta) \in Ans_{\mathcal{P}}(G)$.*

(1) Hay una regla de transformación en $CLNC(\mathcal{D})$ aplicable al objetivo G.

(2) Para cualquier regla **RL** *del cálculo $CLNC(\mathcal{D})$ que sea aplicable al objetivo G, existen l objetivos G_j con respuestas no-triviales y testigos $\mathcal{M}_j : (\Pi_j \square \theta_j) \in Ans_{\mathcal{P}}(G_j)$ $(1 \leq j \leq l)$ tales que:*

- $G \Vdash_{\mathbf{RL}} G_j$ *para cada* $1 \leq j \leq l$,
- $Sol_{\mathcal{D}}(\Pi \square \theta) \subseteq \bigcup_{j=1}^{l} Sol_{\mathcal{D}}(\exists_{\setminus G}.\ \Pi_j \square \theta_j)$ *(bajo el supuesto de que el resolutor sea idealmente completo si* **RL** *es alguna de las dos transformaciones* **CS** *o* **SF***),*
- $(G, \mathcal{M}) \rhd (G_j, \mathcal{M}_j)$ *para cada* $1 \leq j \leq l$*, donde $\rhd$ es el orden de progreso bien fundamentado dado en la Figura 4.1.*

Demostración 12

(1) Si $G \equiv \exists \overline{U}.\ P \square C \square S \square \sigma$ es un objetivo admisible que no está en forma resuelta, entonces P, o bien C, no es vacío. Vamos a proceder asumiendo gradualmente que ninguna regla, excepto una (que será la que hemos denominado **EL***), es aplicable a G, para así concluir entonces que es la regla* **EL** *la que puede ser aplicada en último caso. Observamos que las reglas de fallo no pueden ser aplicadas porque, en caso contrario, G podría no tener una respuesta no-trivial, debido al apartado (2) del* Lema de Corrección.

Asumamos que la regla **AC** *no es aplicable. Entonces, C debería ser vacío y el objetivo sería de la forma $G \equiv \exists \overline{U}.\ P \square \square S \square \sigma$ con P no vacío. Asumamos ahora que las reglas* **DC**, **SP**, **IM**, **PF**, **DF** *y* **FV** *no son aplicables. Entonces, estamos en el caso en el que todas las producciones de P son de la forma*

$$h\overline{e}_m \rightarrow X \ o \ f\overline{e}_n\overline{a}_k \rightarrow X \ (k \geq 0) \ o \ p\overline{e}_n \rightarrow X \ o \ F\overline{a}_k \rightarrow X \ (k > 0)$$

donde $h\overline{e}_m$ es una expresión rígida y pasiva pero no un patrón y en todos los casos X es una variable producida pero no demandada por el objetivo (en particular, $X \notin odvar_{\mathcal{D}}(S)$). Consideramos ahora el conjunto χ de tales $X's$, esto es, $\chi = pvar(G)$.

Si la regla **CS** *no es aplicable, entonces es que S debe estar ya en χ-forma resuelta. Pero entonces, debido al hecho de que $\chi \cap odvar_{\mathcal{D}}(S) = \emptyset$ y a la condición de* **Discriminación** *de los resolutores de restricciones dada en la Definición 5 (Subsección 2.4.2), concluimos que $\chi \cap var(S) = \emptyset$. Elegimos ahora una variable $X \in \chi$ que sea minimal con respecto al orden de producción $\gg_P^+$ (tal elemento minimal ha de existir siempre, debido al número finito de variables que aparecen en G y a la propiedad* **NC** *de los objetivos admisibles). Esta variable X no puede aparecer ni en ninguna otra producción de P ni en la sustitución σ del objetivo por la condición de admisibilidad* **SL**, *por lo cual se verifica que $X \notin var(P \ \square \ C \ \square \ S \ \square \ \sigma)$. Por tanto, la regla* **EL** *puede aplicarse a la producción donde la variable X aparece.*

(2) Esta parte del Lema de Progreso *se demuestra mediante un análisis de casos, utilizando la tabla de la Figura 4.1 que muestra el comportamiento de las diferentes transformaciones del cálculo $CLNC(\mathcal{D})$ con respecto a las cinco componentes del orden lexicográfico que definen el orden de progreso. Los detalles de esta demostración se dan en el Apéndice A.*

□

Finalmente, mediante reiteradas aplicaciones del Lema de Progreso se consigue demostrar el resultado de completitud que estábamos buscando:

Teorema 7 (Completitud del Cálculo *$CLNC(\mathcal{D})$***)** *Supongamos un resolutor idealmente completo. Sea G_0 un objetivo inicial y $(\Pi_0 \ \square \ \theta_0) \in Ans_{\mathcal{P}}(G_0)$ una respuesta no-trivial. Existe un número finito de derivaciones que finalizan en objetivos en forma resuelta $G_0 \vdash^* G_i$ $(1 \leq i \leq k)$ tales que $Sol_{\mathcal{D}}(\Pi_0 \ \square \ \theta_0) \subseteq \bigcup_{i=1}^{k} Sol_{\mathcal{P}}(G_i)$.*

Demostración 13 *Mediante aplicaciones reiteradas del* Lema de Progreso*, podemos construir un árbol finitamente ramificado $\mathcal{T}$ cuya raíz sea $\mathcal{M}_0 : (\Pi_0 \ \square \ \theta_0) \in Ans_{\mathcal{P}}(G_0)$, y tal que cada nodo $\mathcal{M} : (\Pi \ \square \ \theta) \in Ans_{\mathcal{P}}(G)$ asociado a un objetivo G que no está en forma resuelta, tenga hijos $\mathcal{M}_j : (\Pi_j \ \square \ \theta_j) \in Ans_{\mathcal{P}}(G_j)$ $(1 \leq j \leq l)$. Puesto que $\rhd$ es un orden bien fundamentado, no puede haber caminos infinitos en $\mathcal{T}$. Debido al* Lema de König*, $\mathcal{T}$ es un árbol finito con k hojas asociadas a objetivos en forma resuelta G_i $(1 \leq i \leq k)$ tales que $G_0 \vdash^* G_i$ para cada $1 \leq i \leq k$. Probamos que $Sol_{\mathcal{D}}(\Pi_0 \ \square \ \theta_0) \subseteq \bigcup_{i=1}^{k} Sol_{\mathcal{P}}(G_i)$ por inducción sobre la profundidad p de $\mathcal{T}$.*

- Caso base *($p = 0$). $\mathcal{T}$ tiene sólo el nodo raíz $\mathcal{M}_0 : (\Pi_0 \ \square \ \theta_0) \in Ans_{\mathcal{P}}(G_0)$, donde G_0 es un objetivo en forma resuelta. En este caso, $k = 1$, $G_1 \equiv G_0$, y directamente $Sol_{\mathcal{D}}(\Pi_0 \ \square \ \theta_0) \subseteq Sol_{\mathcal{P}}(G_0)$ usando la Proposición 8.*

- Caso inductivo *($p > 0$). $\mathcal{T}$ tiene un nodo raíz $\mathcal{M}_0 : (\Pi_0 \ \square \ \theta_0) \in Ans_{\mathcal{P}}(G_0)$ y subárboles $\mathcal{T}_j$ $(1 \leq j \leq l)$, cada uno de ellos con raíz $\mathcal{M}_j : (\Pi_j \ \square \ \theta_j) \in Ans_{\mathcal{P}}(G_j)$ y hojas asociadas a objetivos en forma resuelta $G_{j,i}$ $(1 \leq i \leq k_j)$. Probamos que $Sol_{\mathcal{D}}(\Pi_0 \ \square \ \theta_0) \subseteq \bigcup_{j=1}^{l} \bigcup_{i=1}^{k_j} Sol_{\mathcal{P}}(G_{j,i})$. Mediante la aplicación del* Lema de Progreso*, tenemos $Sol_{\mathcal{D}}(\Pi_0 \square \theta_0) \subseteq \bigcup_{j=1}^{l} Sol_{\mathcal{D}}(\exists_{\setminus G_0}. \ \Pi_j \square \theta_j)$. Asumamos que $\overline{W}_j$ son las variables cuantificadas existencialmente en $\exists_{\setminus G_0}. \ \Pi_j \ \square \ \theta_j$. Por* hipótesis de inducción *para cada $1 \leq j \leq l$ (la profundidad de cada subárbol $\mathcal{T}_j$ es $p_j < p$), $Sol_{\mathcal{D}}(\Pi_j \square \theta_j) \subseteq \bigcup_{i=1}^{k_j} Sol_{\mathcal{P}}(G_{j,i})$. Más aún, puesto que $\overline{W}_j$ no son variables libres en $G_{j,i}$ para todo $1 \leq i \leq k_j$, se tiene también que $Sol_{\mathcal{D}}(\exists \overline{W}_j. \ \Pi_j \square \ \theta_j) \subseteq \bigcup_{i=1}^{k_j} Sol_{\mathcal{P}}(G_{j,i})$, y el resultado se sigue fácilmente de ambas inclusiones.*

□

A partir de la demostración del *Teorema de Completitud*, observamos que la completitud del cálculo $CLNC(\mathcal{D})$ es *fuerte* en el sentido de que la elección local de la regla de transformación de objetivos aplicada en cada paso puede ser una elección *don't care*. Más aún, el Ejemplo 11 muestra que el número k de respuestas computadas necesarias para cubrir las soluciones de la respuesta dada $\Pi \ \square \ \theta$ ha de permitirse que sea más grande que 1 en general. Una situación similar ocurre en el *Teorema de Completitud de Maher* para $CLP(\mathcal{D})$ [JMMS98], aunque la semántica subyacente y las técnicas de demostración sean bastantes diferentes en ese contexto. En nuestro contexto, la condición (6) de la Definición 5 (relativa al comportamiento de los resolutores de restricciones) es responsable del número *finito* k de respuestas computadas en el *Teorema de Completitud*.

Obsérvese asimismo que, de acuerdo con la condición (6) de completitud pedida en la Definición 5, tanto el apartado (2) del *Lema de Progreso* como el *Teorema de Completitud* del cálculo $CLNC(\mathcal{D})$ solo se podrían afirmar para el caso de soluciones bien tipadas, y bajo el supuesto de que los pasos de cómputo permitan preservar soluciones bien tipadas. Sin embargo, con el fin de separar los problemas de completitud relativos al resolutor del comportamiento de las restantes reglas de transformación de objetivos, se ha optado por demostrar el Lema 11 y el Teorema 7 bajo el supuesto ideal de que se está haciendo uso de un resolutor completo. De esta forma, hemos podido garantizar la propiedad de progreso (y por tanto, la completitud del cálculo) con respecto a todas las soluciones, sean estas bien tipadas o no. Proceder de esta manera permite además resaltar una característica importante del diseño del cálculo de estrechamiento con restricciones: el cálculo permite extender de manera

conservadora las propiedades que de partida satisface el resolutor elegido para resolver restricciones primitivas sobre el dominio al caso más general de la resolución de restricciones definidas por el usuario. Así, si se parte de un resolutor completo, automáticamente el cálculo también se vuelve completo. En este sentido, el propio cálculo de estrechamiento con restricciones se convierte en una versión extendida del resolutor elegido para el caso de restricciones no primitivas, manteniendo y conservando sus propiedades iniciales.

4.3. El cálculo de estrechamiento demandado $CDNC(\mathcal{D})$

En esta sección vamos a presentar un segundo cálculo de estrechamiento con restricciones al que denominaremos *cálculo de estrechamiento con restricciones dirigido por demanda* $CDNC(\mathcal{D})$ (del inglés, *Constrained Demanded Narrowing Calculus*). La organización de la presentación será similar a la utilizada para el cálculo $CLNC(\mathcal{D})$ en la sección anterior. La idea general será ahora la de asegurar la computación de respuestas a partir de objetivos, de forma correcta y completa con respecto a la semántica de $CRWL(\mathcal{D})$, al tiempo que utilizaremos los denominados *árboles definicionales* de un modo similar a como se describe en [Ant97, AEH00, Vad03a, Vad03b, Vad05, Vad07] con el fin de poder asegurar que todos los pasos de estrechamiento perezoso que se ejecutan durante el cómputo sean realmente necesarios.

4.3.1. Posiciones y subexpresiones

Para poder definir la clase de *árboles definicionales* y de $CFLP(\mathcal{D})$-programas que vamos a utilizar para diseñar el cálculo $CLNC(\mathcal{D})$ necesitamos poder manipular expresiones y patrones de una manera más explícita a como hemos hecho hasta ahora. Para ello, vamos a introducir las siguientes definiciones formales, que complementan a las ya dadas en el Capítulo 2.

Una *posición* es una secuencia p de números enteros positivos que permite identificar una subexpresión dentro de una expresión. Para cualquier expresión e, el conjunto $Pos(e)$ de *posiciones* en e se define inductivamente como sigue:

- La secuencia vacía denotada por ϵ, identifica la expresión e en sí misma.
- Para cualquier expresión de la forma $h\overline{e}_m$, la secuencia $i \cdot q$, donde i es un número entero positivo no más grande que m y q es una posición, identifica la subexpresión de e_i en la posición q.

La subexpresión de e en la posición p se denota por $e|_p$, y el resultado de *reemplazar* $e|_p$ por otra expresión e' en e se denota por $e[e']_p$. Si p y q son posiciones, escribimos $p \preccurlyeq q$ si p está por encima o es un *prefijo* de q, y escribimos $p \parallel q$ si las posiciones son

disjuntas. La expresión $p \cdot q$ denota la posición resultante de la concatenación de las posiciones p y q. El conjunto $Pos(e)$ se particiona entonces en $FPos(e)$ y $VPos(e)$ como sigue:

$$FPos(e) = \{p \in Pos(e) \mid e|_p \notin \mathcal{V}ar\} \text{ y } VPos(e) = \{p \in Pos(e) \mid e|_p \in \mathcal{V}ar\}.$$

Si e es una expresión lineal, la notación $pos(X, e)$ será usada para designar la posición donde aparece la variable X en e. La notación $svar(e)$ determina el conjunto de todas aquellas variables que aparecen en posiciones *seguras* de e, es decir, de todas aquellas variables que aparecen e en alguna posición cuyas posiciones anteriores están todas ocupadas por símbolos pasivos.

4.3.2. Árboles definicionales con solapamiento y restricciones

El cálculo $CDNC(\mathcal{D})$ se utiliza sobre una clase especial de $CFLP(\mathcal{D})$-programas denominada $COISS(\mathcal{D})$-*programas*, cuyas reglas de programa pueden ser organizadas en una estructura jerárquica denominada *árbol definicional* [Ant92]. De una manera más precisa, vamos a reformular las nociones presentadas en [Ant97, Ant01, Vad03a, Vad03b, Vad07] sobre *árboles definicionales con solapamiento* y *sistemas inductivamente secuenciales con solapamiento* en la línea de [Vad05], incluyendo ahora nuevas reglas de programa con restricciones sobre un dominio $\mathcal{D}$.

Definición 22 (Árboles Definicionales con Restricciones) *Sea $\mathcal{P}$ un programa sobre un dominio de restricciones dado $\mathcal{D}$. Un* patrón de llamada *es cualquier patrón lineal de la forma $f\bar{t}_n$, donde $f \in DF^n$ y $\bar{t}_n \in Pat_{\mathcal{D}}$. $\mathcal{T}$ es un* árbol definicional con restricciones sobre $\mathcal{D}$ *(abreviadamente $cDT(\mathcal{D})$)* con patrón de llamada τ *si y sólo si su profundidad es finita y uno de los siguientes casos se cumple:*

- $\mathcal{T} \equiv \underline{regla}(\tau \to r_1 \Leftarrow P_1 \square C_1 | \ldots | r_m \Leftarrow P_m \square C_m)$*, donde $\tau \to r_i \Leftarrow P_i \square C_i$ para todo $1 \leq i \leq m$ es una variante de una regla de programa en $\mathcal{P}$.*
- $\mathcal{T} \equiv \underline{caso}(\tau, X, [\mathcal{T}_1, \ldots, \mathcal{T}_k])$*, donde X es una variable en τ, $h_1, \ldots, h_k$ $(k > 0)$ son pares diferentes de símbolos pasivos de $\mathcal{P}$, y para todo $1 \leq i \leq k$, $\mathcal{T}_i$ es un $cDT(\mathcal{D})$ con patrón de llamada $\tau\sigma_i$, donde $\sigma_i = \{X \mapsto h_i\overline{Y}_{m_i}\}$, m_i es la aridad de h_i y $\overline{Y}_{m_i}$ son variables nuevas y distintas tales que $h_i\overline{Y}_{m_i} \in Pat_{\mathcal{D}}$.*

Representaremos un $cDT(\mathcal{D})$ $\mathcal{T}$ con patrón de llamada τ usando la notación $\mathcal{T}_\tau$. Un $cDT(\mathcal{D})$ de un símbolo de función $f \in DF^n$ *definido por $\mathcal{P}$ es un $cDT(\mathcal{D})$ $\mathcal{T}$ con patrón de llamada $f\overline{X}_n$, donde $\overline{X}_n$ son variables nuevas y distintas. Representaremos esta situación usando la notación $\mathcal{T}_f$.*

Ejemplo 14 *En el siguiente $CFLP(\mathcal{FD})$-programa vamos a usar las constructoras de listas ([ ] para denotar una lista vacía y $[X|Xs]$ para denotar una lista no vacía*

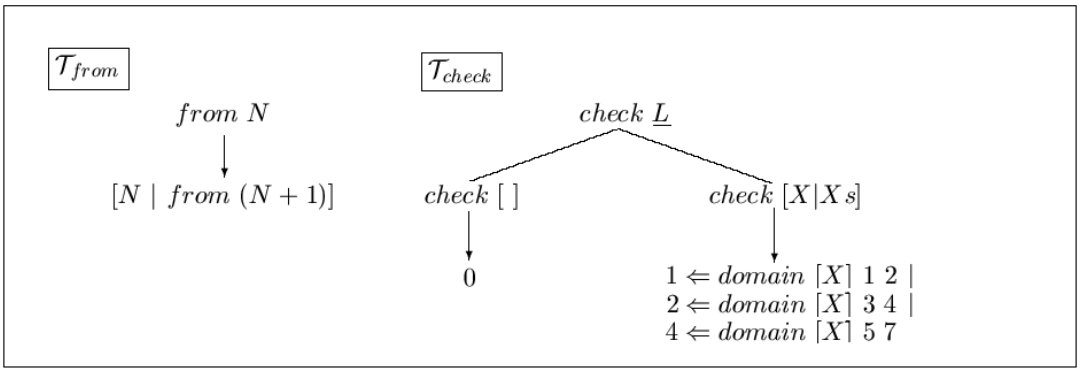


Figura 4.2: Árboles definicionales con restricciones para $from$ y *check*

*constituida por un primer elemento X y una lista Xs), los números enteros (*0, 1, 2, 3, 4, ...*) como elementos básicos del dominio $\mathcal{FD}$, la función primitiva de suma (+) sobre $\mathbb{Z}$, la restricción primitiva domain introducida en el Capítulo 2, una función from para definir una lista infinita que parte de un valor particular, y una función check que restringe el primer elemento de una lista y devuelve diferentes valores dependiendo del intervalo de los enteros.*

$$
\begin{array}{llllll}
from & N & \rightarrow & [N \mid from\ (N+1)] & & \\
check & [\,] & \rightarrow & 0 & & \\
check & [X|Xs] & \rightarrow & 1 & \Leftarrow & domain\ [X]\ \ 1\ \ 2 \\
check & [X|Xs] & \rightarrow & 2 & \Leftarrow & domain\ [X]\ \ 3\ \ 4 \\
check & [X|Xs] & \rightarrow & 4 & \Leftarrow & domain\ [X]\ \ 5\ \ 7
\end{array}
$$

En la Figura 4.2 se representan gráficamente los árboles definicionales con restricciones $\mathcal{T}_{from}$ y $\mathcal{T}_{check}$ correspondientes a los símbolos de función definida from y check, respectivamente:

$$\mathcal{T}_{from} \equiv \underline{regla}\ (from\ N\ \rightarrow\ [N \mid from\ (N+1)])$$

$$
\begin{array}{l}
\mathcal{T}_{check} \equiv \underline{caso}\ (check\ L, L, [\\
\qquad\qquad \underline{regla}\ (check\ [\,] \quad\ \rightarrow\ 0), \\
\qquad\qquad \underline{regla}\ (check\ [X|Xs]\ \rightarrow 1 \Leftarrow domain\ [X]\ 1\ 2\ | \\
\qquad\qquad\qquad\qquad\qquad\quad 2 \Leftarrow domain\ [X]\ 3\ 4\ | \\
\qquad\qquad\qquad\qquad\qquad\quad 4 \Leftarrow domain\ [X]\ 5\ 7\ )\ ]\ )
\end{array}
$$

4.3.3. $COISS(\mathcal{D})$-programas y objetivos con árboles definicionales

En esta sección vamos a definir la clase de los $COISS(\mathcal{D})$-programas con restricciones y árboles definicionales que usaremos en el cálculo $CDNC(\mathcal{D})$ como una subclase propia de los $CFLP(\mathcal{D})$-programas generales presentados en el Capítulo 2. Con respecto a la aplicabilidad del cálculo de estrechamiento con restricciones dirigido por demanda sobre esta clase de programas, mostraremos que cualquier $CFLP(\mathcal{D})$-programa puede ser transformado de una forma efectiva en un $COISS(\mathcal{D})$-programa semánticamente equivalente.

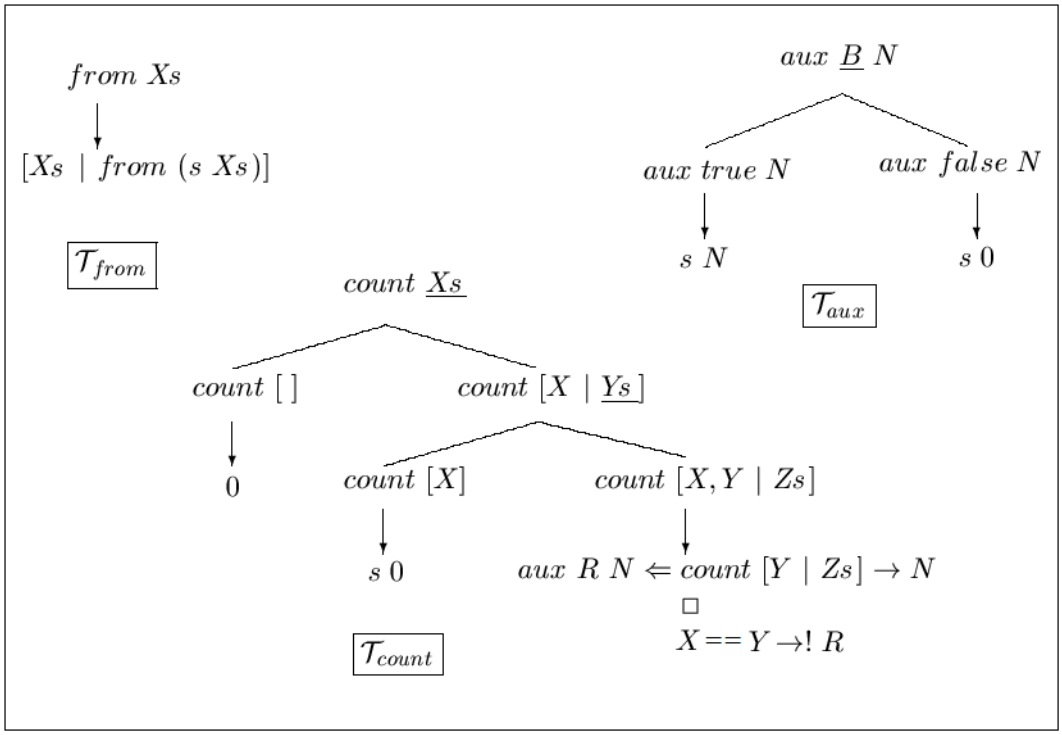


Figura 4.3: Árboles definicionales con restricciones para $from$, $count$ y aux

Definición 23 ($COISS(\mathcal{D})$-programas)

(1) Un símbolo de función $f \in DF^n$ se denomina inductivamente secuencial con restricciones y solapamiento *con respecto a un $CFLP(\mathcal{D})$-programa $\mathcal{P}$ si y sólo si existe un $cDT(\mathcal{D})$ $\mathcal{T}_f$ de f tal que la colección de todas las reglas de programa $\tau \to r_i \Leftarrow P_i \,\square\, C_i$ $(1 \leq i \leq m)$ obtenidas a partir de nodos diferentes $\underline{regla}(\tau \to r_1 \Leftarrow P_1 \,\square\, C_1 | \ldots | r_m \Leftarrow P_m \,\square\, C_m)$ que aparecen en $\mathcal{T}_f$ es igual, salvo variantes, a la colección de todas las reglas de programa en $\mathcal{P}$ cuyo lado izquierdo tiene el símbolo raíz f.*

(2) Un CFLP$(\mathcal{D})$*-programa* $\mathcal{P}$ *se denomina* sistema inductivamente secuencial con restricciones y solapamiento *(abreviadamente, COISS*$(\mathcal{D})$*, del inglés* Constraint Overlapping Inductively Sequential System*) si y sólo si cada función definida por* $\mathcal{P}$ *es inductivamente secuencial con restricciones y solapamiento.*

Ejemplo 15 *El siguiente CFLP*$(\mathcal{D})$*-programa puede ser usado sobre el dominio de restricciones* $\mathcal{H}$*. Usamos las constructoras* $0 \in DC^0$*,* $s \in DC^1$ *y una sintaxis Prolog para las constructoras de listas.*

$$
\begin{array}{llll}
from & Xs & \rightarrow & [Xs \mid from\ (s\ Xs)] \\
count & [\,] & \rightarrow & 0 \\
count & [X] & \rightarrow & s\ 0 \\
count & [X,Y|Zs] & \rightarrow & aux\ R\ N \Leftarrow count\ [Y|Zs] \rightarrow N \ \square\ X == Y \rightarrow\!!\ R \\
aux & true & N & \rightarrow\ s\ N \\
aux & false & N & \rightarrow\ s\ 0
\end{array}
$$

A partir de los árboles definicionales representados gráficamente en la Figura 4.3, es fácil comprobar que este CFLP$(\mathcal{H})$*-programa es un COISS*$(\mathcal{H})$*-programa. Por ejemplo, el símbolo de función count tiene el siguiente árbol definicional* $\mathcal{T}_{count}$*:*

$$
\begin{array}{l}
\underline{caso}\ (count\ Xs, Xs, [\\
\qquad \underline{regla}\ (count\ [\,] \rightarrow 0), \\
\qquad \underline{caso}\ (count\ [X|Ys], Ys, [\\
\qquad\qquad \underline{regla}\ (count\ [X] \rightarrow s\ 0), \\
\qquad\qquad \underline{regla}\ (count\ [X,Y|Zs] \rightarrow aux\ R\ N \Leftarrow count\ [Y|Zs] \rightarrow N \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \square \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad X == Y \rightarrow\!!\ R)\ ]\ )\ ]\ )
\end{array}
$$

El siguiente resultado será de utilidad para demostrar las principales propiedades del cálculo de estrechamiento *CDNC*$(\mathcal{D})$.

Lema 12 (Propiedad de Particionado) *Sea* $\mathcal{P}$ *un CFLP*$(\mathcal{D})$*-programa. Para cualquier* $e \in Exp_{\mathcal{D}}$*,* $t \in Pat_{\mathcal{D}}$ *y* $p \in Pos(e)$*, las siguientes afirmaciones son equivalentes:*

(1) $T : \mathcal{P} \vdash_{\mathcal{D}} e \rightarrow t \Leftarrow \Pi$.

(2) Existe $s \in Pat_{\mathcal{D}}$ *tal que* $T_1 : \mathcal{P} \vdash_{\mathcal{D}} e|_p \rightarrow s \Leftarrow \Pi$ *y* $T_2 : \mathcal{P} \vdash_{\mathcal{D}} e[s]_p \rightarrow t \Leftarrow \Pi$.

Más aún, cuando se demuestra "(1) $\Rightarrow$ *(2)" es posible elegir* $|\ T_1\ |, |\ T_2\ | \leq |\ T\ |$.

La demostración de este resultado puede encontrarse en el Apéndice A.

Definición 24 (Objetivos Admisibles para $COISS(\mathcal{D})$-programas) *Un* objetivo *para un $COISS(\mathcal{D})$-programa dado es de la forma $G \equiv \exists \overline{U}.\ P \sqcap C \sqcap S \sqcap \sigma$, donde el símbolo $\sqcap$ se interpreta como conjunción, y:*

- $\overline{U} =_{def} evar(G)$ *es el denominado conjunto de* variables existenciales *del objetivo G. Se trata de variables intermedias que pueden ser vinculadas a patrones parciales en una solución.*
- $P \equiv e_1 \rightarrow R_1, \ldots, e_n \rightarrow R_n$ *es una conjunción finita de producciones donde cada R_i es una variable distinta y cada e_i es una expresión o un par de la forma $< \tau, \mathcal{T} >$, donde τ es una instancia del patrón en la raíz de un $cDT(\mathcal{D})$ $\mathcal{T}$. Aquellas producciones $e \rightarrow R$ cuyo lado izquierdo e es simplemente una expresión se denominan* suspensiones*, mientras que aquellas cuyos lados izquierdos son de la forma $< \tau, \mathcal{T} >$ se denominan* producciones demandadas*. El conjunto de variables producidas de G se define entonces como $pvar(P) =_{def} \{R_1, \ldots, R_n\}$.*
- $C \equiv \delta_1, \ldots, \delta_k$ *es una conjunción finita de restricciones atómicas (posiblemente incluyendo apariciones de símbolos de función definida).*
- $S \equiv \pi_1, \ldots, \pi_l$ *es una conjunción finita de restricciones atómicas primitivas, denominada* restricción respuesta*.*
- σ *es una sustitución idempotente denominada* sustitución respuesta *tal que $vdom(\sigma) \cap var(P \sqcap C \sqcap S) = \emptyset$.*

Adicionalmente, cualquier objetivo admisible *debe satisfacer las siguientes condiciones de admisibilidad, denominadas* invariantes del objetivo*:*

LN *Cada variable producida es producida sólo una vez, es decir, las variables $R_1, \ldots, R_n$ deben ser todas ellas diferentes.*

EX *Todas las variables producidas deben ser existenciales, es decir, $pvar(P) \subseteq evar(G)$.*

NC *El cierre transitivo de la relación de producción $\gg_P$ debe ser irreflexivo, o equivalentemente, un orden parcial estricto.*

SL *Ninguna variable producida entra en la sustitución respuesta, es decir, $var(\sigma) \cap pvar(P) = \emptyset$.*

DT *Para cada producción demandada $< \tau, \mathcal{T} > \rightarrow R$ en P, la variable R es demandada por el objetivo, es decir, $R \in dvar_{\mathcal{D}}(G)$ (en el sentido ampliado de la Definición 25 dada más abajo), y las variables en $\mathcal{T}$ no aparecen en ningún otro lugar del objetivo.*

4.3.4. Variables demandadas en objetivos con árboles definicionales y restricciones

De manera similar a como se ha hecho en la presentación formal del cálculo de estrechamiento perezoso $CLNC(\mathcal{D})$, el cálculo $CDNC(\mathcal{D})$ va a usar también una noción adecuada de *variable demandada* para poder tratar con evaluación perezosa. En esta ocasión, es necesario considerar como caso especial la demanda con respecto a los árboles definicionales.

Definición 25 (Variables Demandadas en un $COISS(\mathcal{D})$-objetivo) *Consideramos un objetivo admisible $G \equiv \exists \overline{U}.\ P \,\square\, C \,\square\, S \,\square\, \sigma$ para un $COISS(\mathcal{D})$-programa dado y una variable $X \in var(G)$. Decimos que X es una* variable demandada *por el objetivo G si y sólo si uno de los siguientes casos se cumple:*

- *X es una variable obviamente demandada por el almacén de restricciones S del objetivo, es decir, $X \in odvar_{\mathcal{D}}(S)$.*
- *Existe alguna suspensión $(X\overline{a}_k \to R) \in P$ tal que $k > 0$ y R es una variable demandada por el objetivo G.*
- *Existe alguna producción demandada $(< e, \underline{caso}(\tau, Y, [\mathcal{T}_1, \ldots, \mathcal{T}_k]) >\!\!\to R) \in P$ tal que $X = e|_{pos(Y,\tau)}$ y R es una variable demandada por el objetivo G.*

Escribimos $dvar_{\mathcal{D}}(G)$ (o más precisamente $dvar_{\mathcal{D}}(P \,\square\, S)$) para designar el conjunto de variables demandadas por el objetivo G.

El siguiente resultado será también de utilidad para demostrar las principales propiedades del cálculo $CDNC(\mathcal{D})$ y muestra que la semántica proporcionada por la lógica para la reescritura $CRWL(\mathcal{D})$ no acepta un valor indefinido para las variables demandadas.

Lema 13 (Lema de Demanda) *Si $(\Pi \;\square\; \theta)$ es una respuesta no trivial de un objetivo admisible G para un $COISS(\mathcal{D})$-programa $\mathcal{P}$ y $X \in dvar_{\mathcal{D}}(G)$ entonces existe otra respuesta no-trivial $(\Pi \;\square\; \theta')$ de G (con θ' dada por la Definición 18) tal que $\theta =_{\setminus evar(G)} \theta'$ y $\theta'(X) \neq \bot$. Más aún:*

(1) Si $X \notin evar(G)$ entonces $\theta(X) \neq \bot$ se cumple para cualquier $(\Pi \;\square\; \theta) \in Ans_{\mathcal{P}}(G)$.

(2) Si $X \in evar(G)$ entonces existe $(\Pi \;\square\; \theta) \in Ans_{\mathcal{P}}(G)$ tal que $\theta(X) \neq \bot$.

Demostración 14 *Sea $(\Pi \;\square\; \theta) \in Ans_{\mathcal{P}}(G)$ no-trivial (es decir, $Sol_{\mathcal{D}}(\Pi) \neq \emptyset$). Por la Definición 18, existe $(\Pi \;\square\; \theta') \in Ans_{\mathcal{P}}(G)$ tal que $\theta =_{\setminus evar(G)} \theta'$. Probamos que $\theta'(X) \neq \bot$ razonando por inducción sobre el orden $\gg_P^+$ (el cierre transitivo de la relación producida es un orden bien fundamentado, debido a la propiedad* **NC** *de los objetivos admisibles). Consideramos casos para $X \in dvar_{\mathcal{D}}(G)$ de acuerdo con la Definición 25:*

- $X \in odvar_{\mathcal{D}}(S)$:
  Supongamos $\mu \in Sol_{\mathcal{D}}(\Pi)$. *Por la Definición 18,* $\Pi \models_{\mathcal{D}} S\theta'$, *y entonces* $\theta'\mu \in Sol_{\mathcal{D}}(S)$. *Como* $X \in odvar_{\mathcal{D}}(S)$, *se sigue que* $\mu(\theta'(X)) \neq \bot$. *Por lo tanto,* $\theta'(X) \neq \bot$.

- $(X\overline{a}_k \to R) \in P$ *con* $k > 0$ *y* $R \in dvar_{\mathcal{D}}(G)$:
  Puesto que $X \gg^+_P R$ *y* $R \in dvar_{\mathcal{D}}(G)$, *aplicando la* hipótesis de inducción, $\theta'(R) \neq \bot$. *Por la Definición 18,* $\mathcal{P} \vdash_{\mathcal{D}} \theta'(X)\overline{a_k\theta'} \to \theta'(R) \Leftarrow \Pi$. *Sin embargo, puesto que* $k > 0$, *esto sólo es posible en* $CRWL(\mathcal{D})$ *si* $\theta'(X) \neq \bot$.

- $X = e|_{pos(Y,\tau)}$, *donde* $(< e, \underline{caso}(\tau, Y, [\mathcal{T}_1, \ldots, \mathcal{T}_k]) > \to R) \in P$ *y* $R \in dvar_{\mathcal{D}}(G)$:
  En este caso, $e = f\overline{e}_n$ *con* $f \in DF^n$. *Puesto que* $X = e|_{pos(Y,\tau)}$, *se sigue que* $X \gg^+_P R$. *Más aún, puesto que* $R \in dvar_{\mathcal{D}}(G)$, *por* hipótesis de inducción $\theta'(R) \neq \bot$. *Por la Definición 18 y la regla* $DF_{\mathcal{P}}$ *del cálculo* $CRWL(\mathcal{D})$, $\mathcal{P} \vdash_{\mathcal{D}} f\ \overline{e_n\theta'} \to \theta'(R) \Leftarrow \Pi$ *usando* $(f\overline{t'}_n \to r \Leftarrow P \,\square\, C) \in [\mathcal{P}]_\bot$ *y una deducción* $\mathcal{P} \vdash_{\mathcal{D}} e_i\theta' \to t'_i \Leftarrow \Pi$ *para todo* $1 \leq i \leq n$. *Por otra parte,* $pos(Y, \tau) = i \cdot p$ *con* $1 \leq i \leq n$ *y* $p \in Pos(e_i)$ *porque* $\tau \preceq e$. *Debido a la forma del árbol definicional* $\underline{caso}$ *(ver la Definición 22),* t'_i *tiene un símbolo pasivo* h_j $(1 \leq j \leq k)$ *en la posición* p. *Más aún, sólo puede haber símbolos pasivos sobre* h_j *in* t'_i. *De aquí,* $t'_i \neq \bot$. *Más aún, puesto que* $\tau \preceq e$ *con* $X = e_i|_p$, *estos deben ser los mismos símbolos pasivos y en el mismo orden sobre* $\theta'(X)$ *en la posición* p *de* $e_i\theta'$. *De aquí se sigue que* $\mathcal{P} \vdash_{\mathcal{D}} e_i\theta' \to t'_i \Leftarrow \Pi$ *aplica la* $CRWL(\mathcal{D})$*-regla* **DC** *para dar lugar a* $\mathcal{P} \vdash_{\mathcal{D}} \theta'(X) \to h_j \ldots \Leftarrow \Pi$. *Concluimos que* $\theta'(X) \neq \bot$.

□

4.3.5. Estrechamiento dirigido por demanda con restricciones

Al igual que el cálculo $CLNC(\mathcal{D})$, el cálculo $CDNC(\mathcal{D})$ está formado por un conjunto de reglas de transformación para objetivos admisibles. Cada paso de transformación toma la forma $G \Vdash G'$, especificando uno de los posibles modos de ejecutar un paso de resolución de objetivos. Escribiremos $G \Vdash_{\mathbf{RL}} G'$ para indicar que $G \Vdash G'$ por medio de la $CDNC(\mathcal{D})$-regla de transformación **RL**. Las derivaciones son también secuencias de $\Vdash$-pasos. Como en el caso de las derivaciones con restricciones *SLD* para $CLP(\mathcal{D})$-programas [JMMS98], las derivaciones con éxito terminarán eventualmente con un objetivo en forma resuelta. Las derivaciones fallidas (finalizando con un objetivo obviamente inconsistente ■) y las derivaciones infinitas también son posibles. De manera similar a [Vad03a, Vad03b, Vad05, Vad07], todas las reglas de transformación son aplicadas considerando P y C como conjuntos, no como secuencias.

Las reglas de transformación de objetivos que tienen que ver con suspensiones de la forma $e \to R$ están diseñadas para conseguir el objetivo de modelar el comportamiento del estrechamiento perezoso con restricciones y compartición (*sharing*), co-

mo en el cálculo $CLNC(\mathcal{D})$, pero ahora involucrando funciones primitivas, funciones definidas posiblemente indeterministas y de orden superior, variables funcionales, y árboles definicionales con restricciones.

SS Suspensión Simple

$$\exists X, \overline{U}.\ t \to X, P \sqcap C \sqcap S \sqcap \sigma \Vdash_{\mathbf{SS}} \exists \overline{U}.\ (P \sqcap C \sqcap S)\sigma_0 \sqcap \sigma$$

si $t \in Pat_{\mathcal{D}}$ y $\sigma_0 = \{X \mapsto t\}$.

IM Imitación

$$\exists X, \overline{U}.\ h\overline{e}_m \to X, P \sqcap C \sqcap S \sqcap \sigma \Vdash_{\mathbf{IM}} \exists \overline{X}_m, \overline{U}.\ (\overline{e_m \to X_m}, P \sqcap C \sqcap S)\sigma_0 \sqcap \sigma$$

si $h\overline{e}_m \notin Pat_{\mathcal{D}}$ es pasiva, $X \in dvar_{\mathcal{D}}(P \sqcap S)$ y $\sigma_0 = \{X \mapsto h\overline{X}_m\}$ con $\overline{X}_m$ variables nuevas tales que $h\overline{X}_m \in Pat_{\mathcal{D}}$.

EL Eliminación

$$\exists X, \overline{U}.\ e \to X, P \sqcap C \sqcap S \sqcap \sigma \Vdash_{\mathbf{EL}} \exists \overline{U}.\ P \sqcap C \sqcap S \sqcap \sigma$$

si $X \notin var(P \sqcap C \sqcap S \sqcap \sigma)$.

PF Función Primitiva

$$\exists X, \overline{U}.\ p\overline{e}_n \to X, P \sqcap C \sqcap S \sqcap \sigma \Vdash_{\mathbf{PF}} \exists \overline{X}_q, X, \overline{U}.\ \overline{e_q \to X_q}, P \sqcap C \sqcap p\overline{t}_n \to\!! X, S \sqcap \sigma$$

si $p \in PF^n$, $X \in dvar_{\mathcal{D}}(P \sqcap S)$, y $\overline{X}_q$ son variables nuevas ($0 \leq q \leq n$ es el número de $e_i \notin Pat_{\mathcal{D}}$) tal que $t_i \equiv X_j$ ($0 \leq j \leq q$) si $e_i \notin Pat_{\mathcal{D}}$ y $t_i \equiv e_i$ en otro caso para cada $1 \leq i \leq n$.

DT Árbol Definicional

$$\exists X, \overline{U}.\ f\overline{e}_n \to X, P \sqcap C \sqcap S \sqcap \sigma \Vdash_{\mathbf{DT_1}} \exists X, \overline{U}.\ < f\overline{e}_n, \mathcal{T}_{f\overline{X}_n} > \to X, P \sqcap C \sqcap S \sqcap \sigma$$

$$\exists X, \overline{U}.\ f\overline{e}_n\overline{a}_k \to X, P \sqcap C \sqcap S \sqcap \sigma \Vdash_{\mathbf{DT_2}} \exists X, X', \overline{U}.\ < f\overline{e}_n, \mathcal{T}_{f\overline{X}_n} > \to X', X'\overline{a}_k \to X, P \sqcap C \sqcap S \sqcap \sigma$$

si $f \in DF^n (k > 0)$, $X \in dvar_{\mathcal{D}}(P \sqcap S)$, y tanto X' como todas las variables en $\mathcal{T}_{f\overline{X}_n}$ son variables nuevas.

FV Variable Funcional

$$\exists X, \overline{U}.\ F\overline{e}_q \to X, P \,\square\, C \,\square\, S \,\square\, \sigma \vdash_{\mathbf{FV}} \exists \overline{X}_p, X, \overline{U}.\ (h\overline{X}_p\overline{e}_q \to X, P \,\square\, C \,\square\, S)\sigma_0 \,\square\, \sigma\sigma_0$$

si $F \notin pvar(P), q > 0$, $X \in dvar_{\mathcal{D}}(P \,\square\, S)$, $\sigma_0 = \{F \mapsto h\overline{X}_p\}$ y $\overline{X}_p$ son variables nuevas tales que $h\overline{X}_p \in Pat_{\mathcal{D}}$.

La notación $\overline{e_m \to X_m}$ permite abreviar la secuencia $e_1 \to X_1, \ldots, e_m \to X_m$. Ahora, la principal novedad con respecto al cálculo $CLNC(\mathcal{D})$ es la regla **DT**: si e tiene un símbolo de función definida en la raíz y R es una variable demandada, sólo la transformación **DT** es aplicable, despertando así la suspensión correspondiente a e con el $cDT(\mathcal{D})$ apropiado e introduciendo la nueva producción demandada en el nuevo objetivo.

Las nuevas reglas de transformación de objetivos para producciones demandadas $< e, \mathcal{T} > \to R$ permiten codificar la estrategia de *estrechamiento necesario* guiada por un $cDT(\mathcal{D})$ $\mathcal{T}$, de un modo similar a como se hace en [HP99, Vad03b, Vad07].

- Si $\mathcal{T}$ es un árbol de la forma $\underline{regla}$, entonces la regla de transformación **RRA** elige una de las reglas disponibles para reescribir e, introduciendo suspensiones apropiadas y restricciones en el nuevo objetivo de modo que la evaluación perezosa quede asegurada.

- Si $\mathcal{T}$ es un árbol de la forma $\underline{caso}$, una de las siguientes transformaciones, **CSS**, **DI** o **DN**, puede ser aplicada, de acuerdo con el tipo de símbolos que aparecen en e en la posición de distinción de casos del árbol:

  - Si h es un símbolo pasivo h_i, entonces **CS** selecciona el subárbol apropiado (si es posible, en caso contrario, la regla **CC** haría que el cómputo fallase).
  - Si h es una variable no producida Y, entonces la regla **DI** selecciona de manera indeterminista un subárbol, generando un vínculo apropiado para Y.
  - Si h es un símbolo demandado de función primitiva o de función definida g, la regla **DN** introduce una nueva suspensión demandada en el nuevo objetivo, con el fin de evaluar $e|_p$.
  - En otro caso, la selección del subobjetivo $< e, \mathcal{T} > \to R$ debe retrasarse obligatoriamente hasta que se produzca un nuevo paso en la computación que consiga despertarla.

CSS Selección de Caso

$$\exists R, \overline{U}.\ < e, \underline{caso}(\tau, X, [\mathcal{T}_1, \ldots, \mathcal{T}_k]) >\!\!\rightarrow R, P \,\square\, C \,\square\, S \,\square\, \sigma \vdash_{\mathbf{CSS}} \exists R, \overline{U}.\ < e, \mathcal{T}_i >\!\!\rightarrow R, P \,\square\, C \,\square\, S \,\square\, \sigma$$

si $e|_{pos(X,\tau)} = h_i \ldots$, con $1 \leq i \leq k$ determinado por e, donde h_i es el símbolo pasivo asociado a $\mathcal{T}_i$.

DI Instanciación Demandada

$$\exists R, \overline{U}.\ < e, \underline{caso}(\tau, X, [\mathcal{T}_1, \ldots, \mathcal{T}_k]) >\!\!\rightarrow R, P \,\square\, C \,\square\, S \,\square\, \sigma \vdash_{\mathbf{DI}} \exists \overline{Y}_{m_i}, R, \overline{U}.\ (< e, \mathcal{T}_i >\!\!\rightarrow R, P \,\square\, C \,\square\, S)\sigma_0 \,\square\, \sigma\sigma_0$$

si $e|_{pos(X,\tau)} = Y$, $Y \notin pvar(P)$, $\sigma_0 = \{Y \mapsto h_i \overline{Y}_{m_i}\}$ con h_i $(1 \leq i \leq k)$ el símbolo pasivo asociado a $\mathcal{T}_i$, y $\overline{Y}_{m_i}$ variables nuevas.

DN Estrechamiento Demandado

$$\exists R, \overline{U}.\ < e, \underline{caso}(\tau, X, [\mathcal{T}_1, \ldots, \mathcal{T}_k]) >\!\!\rightarrow R, P \,\square\, C \,\square\, S \,\square\, \sigma \vdash_{\mathbf{DN}} \exists R', R, \overline{U}.\ e|_{pos(X,\tau)} \rightarrow R', < e[R']_{pos(X,\tau)}, \underline{caso}(\tau, X, [\mathcal{T}_1, \ldots, \mathcal{T}_k]) >\!\!\rightarrow R, P \,\square\, C \,\square\, S \,\square\, \sigma$$

si $e|_{pos(X,\tau)} = g \ldots$ con $g \in FS$ activo (símbolo de función definida o primitiva) y R' es una variable nueva.

RRA Aplicación de Regla de Reescritura

$$\exists R, \overline{U}.\ < e, \underline{regla}(\tau \rightarrow r_1 \Leftarrow P_1 \,\square\, C_1 | \ldots | r_k \Leftarrow P_k \,\square\, C_k) >\!\!\rightarrow R, P \,\square\, C \,\square\, \overline{S \,\square\, \sigma} \vdash_{\mathbf{RRA}} \exists \overline{X}, R, \overline{U}.\ \sigma_f(R_1) \rightarrow R_1, \ldots, \sigma_f(R_m) \rightarrow R_m, r_i\sigma_c \rightarrow R, P_i\sigma_c, P \,\square\, C_i\sigma_c, C \,\square\, S \,\square\, \sigma$$

- $\sigma_0 = \sigma_c \uplus \sigma_f$ con $vdom(\sigma_0) = var(\tau)$ y $\tau\sigma_0 = e$.
- $\sigma_c =_{def} \sigma \restriction_{dom_c(\sigma_0)}$, donde $dom_c(\sigma_0) = \{X \in vdom(\sigma_0) \mid \sigma_0(X) \in Pat_{\mathcal{D}}\}$.
- $\sigma_f =_{def} \sigma \restriction_{dom_f(\sigma_0)}$, donde $dom_f(\sigma_0) = \{X \in vdom(\sigma_0) \mid \sigma_0(X) \notin Pat_{\mathcal{D}}\} = \{R_1, \ldots, R_m\}$.
- $\overline{X} \equiv var(\tau \rightarrow r_i \Leftarrow P_i \,\square\, C_i) \backslash dom_c(\sigma_0)$.

CC Caso no Cubierto

$$\exists R, \overline{U}.\ < e, \underline{caso}(\tau, X, [\mathcal{T}_1, \ldots, \mathcal{T}_k]) >\!\!\rightarrow R, P \,\square\, C \,\square\, S \,\square\, \sigma \Vdash_{\mathbf{CC}} \blacksquare$$

si $e|_{pos(X,\tau)} = h \ldots$ es un símbolo pasivo y $h \notin \{h_1, \ldots, h_k\}$, donde h_i es el símbolo pasivo asociado a $\mathcal{T}_i$ $(1 \leq i \leq k)$.

Las reglas de transformación de objetivos que manejan restricciones están diseñadas para combinar restricciones atómicas (primitivas o definidas por el usuario) con la acción de un resolutor de restricciones que satisface los requisitos dados en el Capítulo 2. La regla de fallo **SF** se usa para la detección de fallo en el proceso de resolución de restricciones.

CS Resolución de Restricciones

$$\exists \overline{U}.\ P \,\square\, C \,\square\, S \,\square\, \sigma \Vdash_{\mathbf{CS}\{\chi\}} \exists \overline{Y}_i, \overline{U}.\ (P \,\square\, C)\sigma_i \,\square\, S_i \,\square\, \sigma\sigma_i$$

si $\chi = pvar(P)$, S no está en χ-forma resuelta, $solve^{\mathcal{D}}(S, \chi) = \bigvee_{i=1}^{k}(S_i \,\square\, \sigma_i)$, y $\overline{Y}_i$ son las nuevas variables introducidas por el resolutor en $S_i \,\square\, \sigma_i$, para cada $1 \leq i \leq k$.

AC Restricción Atómica

$$\exists \overline{U}.\ P \,\square\ p\overline{e}_n \rightarrow\!!\ t, C \,\square\, S \,\square\, \sigma \Vdash_{\mathbf{AC}} \exists \overline{X}_q, \overline{U}.\ \overline{e_q \rightarrow X_q}, P \,\square\, C \,\square\ p\bar{t}_n \rightarrow\!!\ t, S \,\square\, \sigma$$

si $p \in PF^n$, $p\overline{e}_n \rightarrow\!!\ t$ es una restricción, $\overline{X}_q$ son variables nuevas $(0 \leq q \leq n$ es el número de $e_i \notin Pat_{\mathcal{D}})$ tales que $t_i \equiv X_j$ $(0 \leq j \leq q)$ si $e_i \notin Pat_{\mathcal{D}}$ y $t_i \equiv e_i$ en caso contrario para cada $1 \leq i \leq n$.

SF Fallo en la Resolución

$$\exists \overline{U}.\ P \,\square\, C \,\square\, S \,\square\, \sigma \Vdash_{\mathbf{SF}\{\chi\}} \blacksquare$$

si $\chi = pvar(P)$, S no está en χ-forma resuelta, y $solve^{\mathcal{D}}(S, \chi) = \blacksquare$.

Finalizamos esta sección con un ejemplo de resolución de objetivos que destaca las principales propiedades del cálculo $CDNC(\mathcal{D})$. En cada paso de transformación de objetivo, subrayamos la parte del objetivo que ha sido seleccionada.

Ejemplo 16 *Vamos a computar todas las respuestas posibles a partir de la restricción definida por el usuario* $N \neq s\ (count\ (from\ M))$ *usando el* $COISS(\mathcal{H})$*-programa dado en el Ejemplo 15 y los árboles definicionales que se muestran en la*

Figura 4.3. Este ejemplo ilustra el uso de producciones demandadas y de suspensiones para conseguir el efecto de una evaluación dirigida por demanda.

$$\square\ \underline{N \neq s\ (count\ (from\ M))}\ \square\ \square\ \varepsilon \Vdash_{\mathbf{AC}}$$
$$\exists L.\ \underline{s\ (count\ (from\ M)) \rightarrow L}\ \square\ \square\ N \neq L\ \square \Vdash_{\mathbf{IM}\{L\ \mapsto\ s\,K\}}$$
$$\exists K.\ count\ (from\ M) \rightarrow K\ \square\ \square\ \underline{N \neq s\ K}\ \square \Vdash_{\mathbf{CS}\{K\}}$$

Ahora, el resolutor de restricciones sobre $\mathcal{H}$ proporciona dos posibles alternativas

$$solve^{\mathcal{H}}(\{N \neq s\ K\}, \{K\}) = (\square\,\{N \mapsto 0\}) \vee (\{N' \neq K\}\,\square\,\{N \mapsto s\ N'\})$$

y en consecuencia, hay dos posibles continuaciones del cómputo

(1) $\exists K.\ \underline{count\ (from\ M) \rightarrow K}\ \square\ \square\ \square\ \{N\ \mapsto\ 0\} \Vdash_{\mathbf{EL}}$
$\square\ \square\ \square\ \{N\ \mapsto\ 0\}$

La primera respuesta computada es: $S_1\,\square\,\sigma_1 \equiv\ \square\ \{N\ \mapsto\ 0\}$.

(2) $\exists K, N'.\ \underline{count\ (from\ M) \rightarrow K}\ \square\ \square\ N' \neq K\ \square\ \{N\ \mapsto\ s\ N'\} \Vdash_{\mathbf{DT}}$
$\exists K, N'.\ \underline{< count\ (from\ M),\ \mathcal{T}_{count} >\!\rightarrow K}\ \square\ \square\ N' \neq K\ \square$
$\{N\ \mapsto\ s\ N'\} \Vdash_{\mathbf{DN}}$
$\exists K', K, N'.\ \underline{from\ M \rightarrow K', < count\ K',\ \mathcal{T}_{count} >\!\rightarrow K}\ \square\ \square$
$N' \neq K\ \square\ \{N\ \mapsto\ s\ N'\} \Vdash_{\mathbf{DT}}$
$\exists K', K, N'.\ \underline{< from\ M,\ \mathcal{T}_{from} >\!\rightarrow K', < count\ K', \mathcal{T}_{count} >\!\rightarrow K}\ \square\ \square$
$N' \neq K\ \square\ \{N\ \mapsto\ s\ N'\} \Vdash_{\mathbf{RRA}}$
$\exists K', K, N'.\ \underline{[M|from\ (s\ M)] \rightarrow K', < count\ K',\ \mathcal{T}_{count} >\!\rightarrow K}\ \square\ \square$
$N' \neq K\ \square\ \{N\ \mapsto\ s\ N'\} \Vdash_{\mathbf{IM}\{K'\ \mapsto\ [A|As]\}}$
$\exists A, As, K, N'.\ \underline{M \rightarrow A}, from\ (s\ M) \rightarrow As,$
$< count\ [A|As],\ \mathcal{T}_{count} >\!\rightarrow K\ \square\ \square$
$N' \neq K\ \square\ \{N\ \mapsto\ s\ N'\} \Vdash_{\mathbf{SS}\{A\ \mapsto\ M\}}$
$\exists As, K, N'.\ from\ (s\ M) \rightarrow As, \underline{< count\ [M|As],\ \mathcal{T}_{count} >\!\rightarrow K}\ \square\ \square$
$N' \neq K\ \square\ \{N\ \mapsto\ s\ N'\} \Vdash_{\mathbf{CSS}}$
$\exists As, K, N'.\ \underline{from\ (s\ M) \rightarrow As}, < count\ [M|As],\ \mathcal{T}_{count\ [\cdot|\cdot]} >\!\rightarrow K\ \square\ \square$
$N' \neq K\ \square\ \{N\ \mapsto\ s\ N'\} \Vdash_{\mathbf{DT}}$
$\exists As, K, N'.\ \underline{< from\ (s\ M),\ \mathcal{T}_{from} >\!\rightarrow As},$
$< count\ [M|As],\ \mathcal{T}_{count\ [\cdot|\cdot]} >\!\rightarrow K\ \square\ \square$
$N' \neq K\ \square\ \{N\ \mapsto\ s\ N'\} \Vdash_{\mathbf{RRA}}$
$\exists As, K, N'.\ \underline{[s\ M|from\ (s\ (s\ M))] \rightarrow As},$
$< count\ [M|As],\ \mathcal{T}_{count\ [\cdot|\cdot]} >\!\rightarrow K\ \square\ \square$
$N' \neq K\ \square\ \{N\ \mapsto\ s\ N'\} \Vdash_{\mathbf{IM}\{As\ \mapsto\ [B|Bs]\}}$
$\exists B, Bs, K, N'.\ \underline{s\ M \rightarrow B}, from\ (s\ (s\ M)) \rightarrow Bs,$
$< count\ [M, B|Bs],\ \mathcal{T}_{count\ [\cdot|\cdot]} >\!\rightarrow K\ \square\ \square$
$N' \neq K\ \square\ \{N\ \mapsto\ s\ N'\} \Vdash_{\mathbf{SS}\{B\ \mapsto\ s\,M\}}$

$$\exists Bs, K, N'.\ from\ (s\ (s\ M)) \to Bs,$$
$$\frac{< count\ [M, s\ M|Bs],\ \mathcal{T}_{count\ [\cdot|\cdot]} >\!\!\to K}{\{N\ \mapsto\ s\ N'\} \Vdash_{\mathbf{CSS}}}\ \square\ \square\ N' \neq K\ \square$$

$$\exists Bs, K, N'.\ from\ (s\ (s\ M)) \to Bs,$$
$$\frac{< count\ [M, s\ M|Bs],\ \mathcal{T}_{count\ [\cdot,\cdot|\cdot]} >\!\!\to K}{\{N\ \mapsto\ s\ N'\} \Vdash_{\mathbf{RRA}}}\ \square\ \square\ N' \neq K\ \square$$

$$\exists R, N'', Bs, K, N'.\ from\ (s\ (s\ M)) \to Bs, \underline{aux\ R\ N'' \to K},$$
$$count\ [s\ M|Bs] \to N''\ \square\ M == (s\ M) \to!\ R\ \square\ N' \neq K\ \square$$
$$\{N\ \mapsto\ s\ N'\} \Vdash_{\mathbf{DT}}$$

$$\exists R, N'', Bs, K, N'.\ from\ (s\ (s\ M)) \to Bs,$$
$$\frac{< aux\ R\ N'',\ \mathcal{T}_{aux} >\!\!\to K,\ count\ [s\ M|Bs] \to N''\ \square}{M == (s\ M) \to!\ R\ \square\ N' \neq K\ \square}$$
$$\{N\ \mapsto\ s\ N'\} \Vdash_{\mathbf{DI}\{R\ \mapsto\ false\}}$$

$$\exists N'', Bs, K, N'.\ from\ (s\ (s\ M)) \to Bs,$$
$$\frac{< aux\ false\ N'',\ \mathcal{T}_{aux,false} >\!\!\to K,}{count\ [s\ M|Bs] \to N''}$$
$$\square\ M \neq s\ M\ \square\ N' \neq K\ \square\ \{N\ \mapsto\ s\ N'\} \Vdash_{\mathbf{RRA}}$$

$$\exists N'', Bs, K, N'.\ from\ (s\ (s\ M)) \to Bs, \underline{s\ 0 \to K},$$
$$count\ [s\ M|Bs] \to N''\ \square\ M \neq s\ M\ \square\ N' \neq K\ \square$$
$$\{N\ \mapsto\ s\ N'\} \Vdash_{\mathbf{SS}\{K\ \mapsto\ s\ 0\}}$$

$$\exists N'', Bs, N'.\ from\ (s\ (s\ M)) \to Bs, \underline{count\ [s\ M|Bs] \to N''}\ \square$$
$$M \neq s\ M\ \square\ N' \neq s\ 0\ \square\ \{N\ \mapsto\ s\ N'\} \Vdash_{\mathbf{EL}}$$

$$\exists Bs, N'.\ \underline{from\ (s\ (s\ M)) \to Bs}\ \square\ M \neq s\ M\ \square\ N' \neq s\ 0\ \square$$
$$\{N\ \mapsto\ s\ N'\} \Vdash_{\mathbf{EL}}$$

$$\exists N'.\ \square\ \underline{M \neq s\ M}\ \square\ N' \neq s\ 0\ \square\ \{N\ \mapsto\ s\ N'\} \Vdash_{\mathbf{AC}}$$

$$\exists N'.\ \square\ \square\ \underline{M \neq s\ M, N' \neq s\ 0}\ \square\ \{N\ \mapsto\ s\ N'\} \Vdash_{\mathbf{CS}\{\emptyset\}}$$

$$solve^{\mathcal{H}}(\{M \neq s\ M, N' \neq s\ 0\}, \emptyset) =$$
$$(\{N' \neq s\ 0\}\ \square\ \{M\ \mapsto\ 0\}) \vee$$
$$(\{M' \neq M, N' \neq s\ 0\}\ \square\ \{M\ \mapsto\ s\ M'\})$$

$$\exists N'.\ \square\ \square\ N' \neq s\ 0\ \square\ \{M\ \mapsto\ 0, N\ \mapsto\ s\ N'\}$$
Segunda respuesta: $S_2 \square \sigma_2 \equiv N' \neq s\ 0\ \square\ \{M\ \mapsto\ 0, N\ \mapsto\ s\ N'\}$

$$\exists M', N'.\ \square\ \square\ M' \neq M, N' \neq s\ 0\ \square\ \{M\ \mapsto\ s\ M', N\ \mapsto\ s\ N'\}$$
Tercera respuesta: $S_3 \square \sigma_3 \equiv M' \neq M, N' \neq s\ 0\ \square\ \{M\ \mapsto\ s\ M',$
$$N\ \mapsto\ s\ N'\}$$

De manera similar a como ocurría en el cálculo $CLNC(\mathcal{D})$, para este ejemplo también es posible demostrar que $\Pi\ \square\ \theta \equiv N \neq s\ (s\ 0)\ \square\ \{M\ \mapsto\ 0\}$ *es una respuesta*

correcta tal que $Sol_{\mathcal{H}}(\Pi \square \theta) \subseteq \bigcup_{i=1}^{2} Sol_{\mathcal{H}}(S_i \square \sigma_i)$. *Sin embargo, no existe una* única *respuesta computada* $S \square \sigma$ *para la que se verifique que* $Sol_{\mathcal{H}}(\Pi \square \theta) \subseteq Sol_{\mathcal{H}}(S \square \sigma)$. *Veremos en la siguiente subsección que de nuevo esto es cierto en general.*

4.3.6. Resultados de corrección y completitud fuerte

En esta sección presentamos las principales características del cálculo de resolución de objetivos $CDNC(\mathcal{D})$ con respecto a la semántica declarativa proporcionada por la lógica para la reescritura $CRWL(\mathcal{D})$, es decir, su *corrección* y su *completitud*. Para probar ambas propiedades vamos a usar las mismas técnicas que las que hemos usado para el cálculo $CLNC(\mathcal{D})$. El primer resultado nos va a permitir demostrar la corrección de un solo paso de transformación. Este resultado muestra que los pasos de transformación preservan la admisibilidad de los objetivos, fallan sólo en el caso de objetivos insatisfactibles y no introducen nuevas respuestas.

Lema 14 (Lema de Corrección) *Sea* $\mathcal{P}$ *un* $COISS(\mathcal{D})$*-programa. El cálculo de resolución de objetivos* $CDNC(\mathcal{D})$ *verifica las siguientes propiedades:*

(1) Los pasos de transformación del cálculo preservan la admisibilidad de objetivos: Si $G \vdash_{CDNC(\mathcal{D})} G'$ *y* G *es un objetivo admisible, entonces* G' *también es un objetivo admisible. Más aún,* $fvar(G') \subseteq fvar(G)$.

(2) Los pasos de transformación fallan sólo en el caso de objetivos insatisfactibles: Si $G \vdash_{CDNC(\mathcal{D})} \blacksquare$ *entonces* $Sol_{\mathcal{P}}(G) = \emptyset$ *(o equivalentemente, el conjunto de respuestas* $Ans_{\mathcal{P}}(G)$ *sólo incluye respuestas triviales).*

(3) Los pasos de transformación no introducen nuevas respuestas: Si $G \vdash_{CDNC(\mathcal{D})} G'$ *y* $(\Pi \square \theta) \in Ans_{\mathcal{P}}(G')$ *entonces* $(\Pi \square \theta) \in Ans_{\mathcal{P}}(G)$.

La corrección del cálculo se sigue directamente del *Lema de Corrección* y asegura que las respuestas computadas para un objetivo admisible G son respuestas correctas de G con respecto a la semántica declarativa del esquema $CFLP(\mathcal{D})$.

Teorema 8 (Corrección de $CDNC(\mathcal{D})$**)** *Si* G_0 *es un objetivo admisible inicial tal que* $G_0 \vdash^*_{CDNC(\mathcal{D})} G_n$, *donde* $G_n \equiv \exists \overline{U}.\ \square \square S \square \sigma$ *es un objetivo en forma resuelta, entonces* $(S \square \sigma) \in Ans_{\mathcal{P}}(G_0)$.

La completitud del cálculo $CDNC(\mathcal{D})$ se basa en la misma idea que hemos utilizado para demostrar la completitud del cálculo $CLNC(\mathcal{D})$: siempre que tengamos una respuesta $(\Pi \ \square \ \theta) \in Ans_{\mathcal{P}}(G)$ para un objetivo admisible G que no esté en forma resuelta, existe una cantidad finita de posibilidades de elegir un primer paso de cómputo $G \vdash G_j$ $(1 \leq j \leq l)$ de forma que los nuevos objetivos G_j están más próximos a ser resueltos y permiten cubrir todas las soluciones de $\Pi \ \square \ \theta$. Esta idea la precisamos formalmente a través del correspondiente *Lema de Progreso*,

el cual recae también sobre un *orden de progreso* bien fundamentado $\rhd$ para objetivos admisibles con árboles definicionales, técnicamente más complicado que el utilizado para el cálculo $CLNC(\mathcal{D})$, ya que ha de combinar las técnicas ya presentadas para *CFLP* en las secciones anteriores con las técnicas utilizadas para árboles definicionales en [Vad03a, Vad03b, Vad07] para demostrar la completitud de cálculos de estrechamiento perezoso en lenguajes *FLP*. Con este propósito, recogemos en la siguiente definición algunos ordenes de utilidad.

Definición 26 (Orden de Progreso para $CDNC(\mathcal{D})$) *Sea $\mathcal{P}$ un $COISS(\mathcal{D})$-programa, $G \equiv \exists \overline{U}.\ P \ \square\ C \ \square\ S \ \square\ \sigma$ un objetivo admisible para $\mathcal{P}$ y $\mathcal{M} : (\Pi \,\square\, \theta) \in Ans_{\mathcal{P}}(G)$ una respuesta para G con testigo $\mathcal{M}$. Definimos los siguientes tamaños y ordenes asociados a G y a $\mathcal{M}$:*

(a) Un multiconjunto de parejas de números naturales $\mathcal{W}(G, \mathcal{M}) =_{def} \{\| T \|_{G,\mathcal{M}} \mid T \in \mathcal{M}\}$, denominado peso del testigo*, donde para cada $CRWL(\mathcal{D})$-prueba $T \in \mathcal{M}$, la pareja de números $\| T \|_{G,\mathcal{M}} \in \mathbb{N} \times \mathbb{N}$ se define como sigue:*

- *Si $T : \mathcal{P} \vdash_{\mathcal{D}} t\theta \to \theta(R) \Leftarrow \Pi$ para una producción demandada $(< \tau, \mathcal{T} > \to R) \in P$, entonces $\| T \|_{G,\mathcal{M}} =_{def} (| T |_{OR}, | T |)$.*
- *Si $T : \mathcal{P} \vdash_{\mathcal{D}} t\theta \to \theta(R) \Leftarrow \Pi$ para una suspensión $(t \to R) \in P$, entonces $\| T \|_{G,\mathcal{M}} =_{def} (| T |_{OR}, 1 + | T |)$.*
- *Si $T : \mathcal{P} \vdash_{\mathcal{D}} \delta\theta \Leftarrow \Pi$ para una restricción $\delta \in C$, entonces $\| T \|_{G,\mathcal{M}} =_{def} (| T |_{OR}, | T |)$.*

En todos los casos, $| T |_{OR}$ es el número de pasos de deducción en la derivación T que usan la $CRWL(\mathcal{D})$-regla de deducción $\boldsymbol{DF}_{\mathcal{P}}$ y $| T |$ es el número total de pasos de deducción en T.

(b) Un multiconjunto de números naturales $\mathcal{D}(G) =_{def} \{| \mathcal{T} | \mid (< \tau, \mathcal{T} > \to R) \in P\}$, denominado peso de los árboles definicionales *del objetivo G, donde cada $| \mathcal{T} | \in \mathbb{N}$ es el número total de nodos en el árbol definicional $\mathcal{T}$.*

(c) El tamaño $| G |_0 =_{def} (\mathcal{W}(G, \mathcal{M}), \mathcal{D}(G))$.

Definimos un orden de progreso bien fundamentado $\rhd$ *sobre parejas $(G, \mathcal{M})$ a partir del tamaño $| G |_0$ y de los tamaños dados en la Definición 26:*

$$(G, \mathcal{M}) \rhd (G', \mathcal{M}') \Leftrightarrow_{def} (| \mathcal{M} |, | G |_0, | G |_1, | G |_2, | G |_3, | S |) \succ_{lex} (| \mathcal{M}' |, | G' |_0, | G' |_1, | G' |_2, | G' |_3, | S' |)$$

donde $\succ_{lex}$ es el producto lexicográfico de $\succ_{mul} \times \succ_{lex} \times >_{\mathbb{N}} \times >_{\mathbb{N}} \times >_{\mathbb{N}} \times >_{\mathbb{N}}$, $\succ_{mul}$ es el orden de multiconjuntos para multiconjuntos sobre $\mathbb{N}$, y $>_{\mathbb{N}}$ es el orden usual sobre $\mathbb{N}$. En [BN98] se proporcionan definiciones concretas para las definiciones de estas nociones.

REGLA	$\mid \mathcal{M} \mid$	$\mathcal{W}(G, \mathcal{M})$	$\mathcal{D}\ (G)$	$\mid G \mid_0$	$\mid G \mid_1$	$\mid G \mid_2$	$\mid G \mid_3$	$\mid S \mid$
SS	$\succeq_{mul}$	$\succeq_{mul}$	$=_{mul}$	$\succeq_{lex}$	$\geqslant_{\mathbb{N}}$	$\geqslant_{\mathbb{N}}$	$>_{\mathbb{N}}$	
IM	$\succeq_{mul}$	$\succeq_{mul}$	$=_{mul}$	$\succeq_{lex}$	$\geqslant_{\mathbb{N}}$	$>_{\mathbb{N}}$		
EL	$\succeq_{mul}$	$\succeq_{mul}$	$=_{mul}$	$\succeq_{lex}$	$\geqslant_{\mathbb{N}}$	$\geqslant_{\mathbb{N}}$	$>_{\mathbb{N}}$	
PF	$\succ_{mul}$	$\succ_{mul}$						
DT	$\succeq_{mul}$	$\succ_{mul}$						
FV	$\succeq_{mul}$	$\succeq_{mul}$	$=_{mul}$	$\succeq_{lex}$	$>_{\mathbb{N}}$			
CSS	$\succeq_{mul}$	$\succeq_{mul}$	$\succ_{mul}$	$>_{lex}$				
DI	$\succeq_{mul}$	$\succeq_{mul}$	$\succ_{mul}$	$>_{lex}$				
DN	$\succ_{mul}$	$\succ_{mul}$						
RRA	$\succ_{mul}$	$\succ_{mul}$						
CS	$\succeq_{mul}$	$\succeq_{mul}$	$=_{mul}$	$\succeq_{lex}$	$\geqslant_{\mathbb{N}}$	$\geqslant_{\mathbb{N}}$	$\geqslant_{\mathbb{N}}$	$>_{\mathbb{N}}$
AC	$\succ_{mul}$	$\succ_{mul}$						

Figura 4.4: Orden de progreso $(G, \mathcal{M}) \rhd (G_j, \mathcal{M}_j)$

Lema 15 (Lema de Progreso) *Asumamos un objetivo admisible G que no esté en forma resuelta y una respuesta no-trivial con testigo $\mathcal{M} : (\Pi \,\square\, \theta) \in Ans_{\mathcal{P}}(G)$.*

(1) Hay alguna regla de transformación en $CDNC(\mathcal{D})$ que es aplicable a G.

(2) Para cualquier regla **RL** *del cálculo $CDNC(\mathcal{D})$ que sea aplicable a G, existen l objetivos G_j con respuestas no-triviales y testigos $\mathcal{M}_j : (\Pi_j \,\square\, \theta_j) \in Ans_{\mathcal{P}}(G_j)$ $(1 \leq j \leq l)$ tales que:*

- $G \Vdash_{\mathbf{RL}} G_j$ *para cada* $1 \leq j \leq l$,
- $Sol_{\mathcal{D}}(\Pi \,\square\, \theta) \subseteq \bigcup_{j=1}^{l} Sol_{\mathcal{D}}(\exists_{\setminus G}.\ \Pi_j \,\square\, \theta_j)$ *(bajo el supuesto de que el resolutor sea idealmente completo si* **RL** *es alguna de las dos transformaciones* **CS** *o* **SF***),*
- $(G, \mathcal{M}) \rhd (G_j, \mathcal{M}_j)$ *para cada* $1 \leq j \leq l$, *donde* $\rhd$ *es el orden de progreso bien fundamentado presentado en la Definición 26 e ilustrado en la Figura 4.4.*

Demostración 15

(1) Si $G \equiv \exists \overline{U}.\ P \,\square\, C \,\square\, S \,\square\, \sigma$ es un objetivo admisible que no está en forma resuelta, entonces P, o bien C, no puede ser vacío. Vamos a proceder asumiendo gradualmente que ninguna regla, excepto la regla denominada **EL**, *es aplicable al objetivo G, para concluir entonces que es esta regla* **EL** *la que puede ser aplicada en último caso. En primer lugar, observamos que las reglas de fallo no pueden ser aplicables porque, de otro modo, G podría no tener respuestas no-triviales, debido al apartado (2) del* Lema de Corrección. *Asumamos ahora*

que la regla **AC** *no es aplicable. Entonces, C debe ser vacío, y el objetivo ha de ser de la forma $G \equiv \exists\overline{U}.\ P \square \square S \square \sigma$ con P no vacío. Asumamos ahora que las reglas* **SS**, **IM**, **PF**, **DT**, **FV** *no son aplicables sobre suspensiones y que las reglas* **CSS**, **DI**, **DN**, **RRA** *no son aplicables sobre producciones demandadas. Entonces, debe darse el caso de que cada una de las producciones en P sea de una de las siguientes formas:*

(a) $h\overline{e}_m \to R$ o $f\overline{e}_n\overline{a}_k \to R$ $(k \geq 0)$ o $p\overline{e}_n \to R$ o $F\overline{a}_k \to R$ $(k > 0)$, donde $h\overline{e}_m$ es una expresión rígida y pasiva pero no un patrón y en todos los casos R es una variable producida pero no demandada por el objetivo, en particular, $R \notin odvar_{\mathcal{D}}(S)$.

(b) $R'\overline{a}_k \to R$ $(k > 0)$ o $< e, \underline{caso}(\tau, X, [\mathcal{T}_1, \ldots, \mathcal{T}_k]) > \to R$ con $e|_{pos(\tau,X)} = R'$, y ambas, R y R', son variables producidas y demandadas (debido a la condición de admisibilidad **DT** *y a la Definición 25).*

Sin embargo, observamos que realmente no pueden aparecer en P producciones de la forma (b) (pues, de otro modo, podríamos encontrar una producción de la forma (b) y otra suspensión $e \to R'$ de la forma (a) con R' una variable producida y demandada, lo que da lugar a una contradicción). Así pues, todas las producciones en P han de ser de la forma (a) mencionada más arriba.

Consideramos ahora el conjunto χ de estas R's, esto es, $\chi =_{def} pvar(G)$. Si la regla **CS** *no es aplicable, entonces S debe estar en χ-forma resuelta. Pero entonces, debido al hecho de que $\chi \cap odvar_{\mathcal{D}}(S) = \emptyset$, y a los requisitos pedidos a los resolutores de restricciones en el Capítulo 2, concluimos que $\chi \cap var(S) = \emptyset$. Elegimos ahora una variable $R \in \chi$ que sea minimal en la relación $\gg_P^+$ (tal elemento minimal ha de existir, debido al número finito de variables que aparecen en G y a la propiedad* **NC** *de objetivos admisibles). Tal variable R no puede aparecer ni en ninguna otra producción de P ni en la sustitución σ del objetivo, por la condición de admisibilidad* **SL**, *por lo que entonces ha de verificar que $R \notin var(P \ \square \ C \ \square \ S \ \square \ \sigma)$. Por tanto, la regla* **EL** *puede ser aplicada a la producción donde R aparece en el objetivo.*

(2) Esta parte del Lema de Progreso *se demuestra mediante un análisis de casos, utilizando la tabla de la Figura 4.4 que muestra el comportamiento de las diferentes transformaciones del cálculo $CLNC(\mathcal{D})$ con respecto a las distintas componentes del orden lexicográfico que definen el orden de progreso. Los detalles de esta demostración se dan en el Apéndice A.*

□

Finalmente, mediante la aplicación reiterada del *Lema de Progreso*, el siguiente resultado de completitud para el cálculo $CDNC(\mathcal{D})$ es fácil de demostrar. La prueba

es completamente análoga a la realizada para el cálculo $CLNC(\mathcal{D})$ en el Teorema 7 y también revela que el cálculo $CDNC(\mathcal{D})$ es *fuertemente completo*, es decir, la elección local de la regla de transformación de objetivo aplicada en cada paso es una elección *don't care*. De nuevo, obsérvese que tanto el *Lema de Progreso* como el *Teorema de Completitud* han sido formulados bajo el supuesto ideal de que se está haciendo uso de un resolutor de restricciones completo, con el fin de garantizar que lo que afirman ambos resultados sea cierto con respecto a cualquier tipo de solución, sea ésta bien tipada o no. De esta forma, las propiedades exhibidas por el cálculo $CDNC(\mathcal{D})$ para resolver restricciones definidas por el usuario extienden conservativamente las propiedades que inicialmente poseía el resolutor sobre restricciones primitivas del dominio.

Teorema 9 (Completitud del Cálculo $CDNC(\mathcal{D})$) *Supongamos un resolutor idealmente completo. Sea G_0 un objetivo inicial y $(\Pi_0 \ \square \ \theta_0) \in Ans_{\mathcal{P}}(G_0)$ una respuesta no-trivial. Existe un número finito de derivaciones que finalizan en objetivos en forma resuelta $G_0 \vdash^* G_i$ $(1 \leq i \leq k)$ tales que $Sol_{\mathcal{D}}(\Pi_0 \ \square \ \theta_0) \subseteq \bigcup_{i=1}^{k} Sol_{\mathcal{P}}(G_i)$.*

De nuevo, el *Teorema de Completitud* que hemos obtenido es un resultado más fuerte y más general que los resultados previos que se habían obtenido para lenguajes *FLP* y *CFLP* en trabajos anteriores [Lop92, AGL94, ALR99]. La suposición de que se dispone de un resolutor idealmente completo se ha utilizado en el Lema 15 y en el Teorema 9 por las razones ya comentadas en el último párrafo de la sección anterior.

4.3.7. Resultados de optimalidad

Como ya se ha visto, el cálculo de resolución de objetivos $CDNC(\mathcal{D})$ utiliza una técnica de cálculo de pasos de estrechamiento necesario mediante árboles definicionales, siguiendo la idea de la estrategia de *estrechamiento necesario* (del inglés, *needed narrowing*, abreviado como *NN* en lo que sigue) originalmente propuesta en [AEH94] y posteriormente expuesta con más detalle en [AEH00]. En esta subsección vamos a exponer una comparación entre la estrategia *NN* y el cálculo $CDNC(\mathcal{D})$, estudiando en particular si los resultados de optimalidad establecidos para *NN* valen en algún sentido para $CDNC(\mathcal{D})$.

En primer lugar, hay que observar que *NN* y $CDNC(\mathcal{D})$ se refieren a dos clases diferentes de sistemas de reescritura. En el caso de *NN*, se trata de la familia de los sistemas de reescritura *inductivamente secuenciales* (del inglés, *inductively sequential systems*, abreviado como *ISS*), que son sistemas de reescritura ortogonales con disciplina de constructoras, y por tanto, incondicionales. En [Ant97] se ha presentado una generalización de *NN* para la familia de los sistemas de reescritura *inductivamente secuenciales con solapamiento* (del inglés, *overlapping inductively sequential systems*, abreviado como *OISS*), una extensión de la familia *ISS* que admite funciones indeterministas, pero sigue estando limitada a sistemas incondicionales con

disciplina de constructoras. En cambio, $CDNC(\mathcal{D})$ se ha planteado como cálculo de resolución de objetivos para programas $COISS(\mathcal{D})$, los cuales son aún más generales que *OISS* por admitir reglas de reescritura condicionales para funciones de orden superior, empleando restricciones sobre el dominio $\mathcal{D}$ como condiciones.

Deben notarse además algunas diferencias relevantes entre la semántica para *ISS* y *OISS* que se adopta como base para los resultados sobre *NN* presentados en [AEH94, AEH00, Ant97] y la semántica para $COISS(\mathcal{D})$ adoptada en esta tesis. Esta última se caracteriza mediante la lógica de reescritura con restricciones $CRWL(\mathcal{D})$ desarrollada en el Capítulo 3, la cual incluye el tratamiento del indeterminismo con *call-time choice* y el tratamiento de la igualdad estricta como restricción. En cambio, la semántica adoptada en [AEH94, AEH00, Ant97] como referencia para los resultados de corrección y completitud del estrechamiento es la reescritura de términos, la cual no trata al indeterminismo como *call-time choice* ni a la igualdad estricta como restricción. En lugar de esto, [AEH94, AEH00, Ant97] proponen un mecanismo menos general, a saber: reglas de reescritura que capturan el comportamiento de la igualdad estricta entre términos construidos cerrados.

Como consecuencia de las diferencias explicadas en los dos párrafos anteriores, los resultados de corrección y completitud para $CDNC(\mathcal{D})$ presentados en esta tesis son más generales que los presentados para *NN* en [AEH94, AEH00, Ant97] y consideran el indeterminismo de diferente modo. Pasemos ahora a considerar los resultados de optimalidad para *NN* que se presentan en [AEH94, AEH00, Ant97]. Estos pueden resumirse como sigue:

1. *Optimalidad relativa a la independencia de soluciones calculadas.*

   a) *Resultado para ISS*: si σ y σ' son dos sustituciones calculadas por *NN* para un mismo objetivo en cómputos diferentes, entonces hay alguna variable X del objetivo tal que $\sigma(X)$ y $\sigma(X')$ son no unificables debido a un choque entre constructoras distintas (véase el Teorema 4 en [AEH94] y el Teorema 5 en [AEH00]).

   b) *Resultado para OISS*: si σ y σ' son dos sustituciones calculadas por *NN* para un mismo objetivo en cómputos diferentes, entonces hay alguna variable X del objetivo tal que $\sigma(X)$ y $\sigma(X')$ son no unificables debido a un choque entre constructoras distintas, excepto en el caso de que los dos cómputos solo difieran en la elección de diferentes reglas de reescritura con lados izquierdos idénticos, asociadas a una misma hoja del árbol definicional correspondiente (véase el Teorema 3 en [Ant97]).

2. *Optimalidad relativa a la ausencia de pasos de cómputo innecesarios.*

   a) *Resultado para ISS*: el cálculo de una sustitución σ como solución de un objetivo mediante la estrategia *NN* se puede presentar como sucesión de

multipasos de estrechamiento (donde cada multipaso representa la ejecución simultánea de varios pasos de estrechamiento en posiciones independientes) con coste c, de tal manera que cualquier sucesión de multipasos de estrechamiento que calcule σ con coste c' cumple que $c \leq c'$. La medida de coste utilizada aquí tiene una definición técnica basada en la siguiente idea: cada multipaso contribuye con un coste n igual al número de clases de equivalencia del conjunto de posiciones afectadas por ese multipaso, entendiendo que dos posiciones son equivalentes si los subtérminos que las ocupan son descendientes de un antepasado común. Esto se basa a su vez en una generalización al estrechamiento de la noción de descendiente habitualmente utilizada para sistemas de reescritura ortogonales.

b) *Resultado para OISS*: el cálculo de una sustitución σ como solución de un objetivo mediante la estrategia *NN* "*performs only unavoidable steps, modulo non-deterministic choices*" (el texto en inglés señalado en cursiva es cita literal del enunciado del Corolario 34 en [Ant97]).

Es un hecho conocido que los resultados de optimalidad relativos a la independencia de soluciones calculadas en general se pierden en el caso de sistemas de reescritura condicionales con variables extra que pueden aparecer en las condiciones pero no en el lado izquierdo. Piénsese por ejemplo en un sistema de reescritura que incluya las siguientes reglas: $p(X) \rightarrow true \Leftarrow q(X,Y) == true$, $q(a,b) \rightarrow true$, $q(a,c) \rightarrow true$. Puesto que la estrategia *NN* solo se basa en cálculos que afectan a la organización de los lados izquierdos de las reglas de reescritura según indiquen los árboles definicionales, es obviamente posible plantear una generalización de *NN* que opera con estas reglas de reescritura y calcula dos veces, en dos cómputos distintos, la solución $\sigma = \{X \mapsto a\}$ para el objetivo $p(X) == true$. De hecho, esta solución es calculada por $CDNC(\mathcal{H})$ en dos cómputos diferentes. Pese a contraejemplos como éste, se puede afirmar que el cálculo $CDNC(\mathcal{D})$ sigue reteniendo la esencia del resultado de independencia de soluciones calculadas en el siguiente sentido: los resultados 1.*a*) y 1.*b*) discutidos más arriba se demuestran a partir del hecho de que dos ramas diferentes de un mismo árbol definicional dan lugar al cálculo de dos sustituciones independientes, hecho que se mantiene obviamente en el cálculo $CDNC(\mathcal{D})$.

En cuanto a los resultados de optimalidad relativos a la ausencia de pasos de cómputo innecesarios, su formulación técnica en los trabajos [AEH94, AEH00, Ant97] (con conceptos tales como multipaso, y una medida de coste que identifica a los subtérminos descendientes de un antepasado común) está condicionada por el hecho de que en dichos trabajos los cómputos se formalizan como sucesiones de términos resultantes de la reiteración de pasos de estrechamiento, sin utilizar ningún mecanismo formal que garantice la compartición (en inglés, *sharing*) de los descendientes de un mismo subtérmino en posteriores pasos. En cambio, en $CDNC(\mathcal{D})$ los cómputos se formalizan mediante un cálculo de transformación de objetivos, y hay

un recurso formal para garantizar la compartición mediante las producciones que aparecen en los objetivos. Es más, la compartición es necesaria para la corrección de $CDNC(\mathcal{D})$ con respecto a la semántica *call-time choice* del indeterminismo exigida por la lógica de reescritura con restricciones $CRWL(\mathcal{D})$. Por estas razones, en $CDNC(\mathcal{D})$ no tiene sentido plantear la optimalidad relativa a la ausencia de pasos de cómputo innecesarios en términos de una medida de coste de los cómputos semejante a las empleadas en [AEH94, AEH00, Ant97]. No obstante, también en este caso se puede afirmar que $CDNC(\mathcal{D})$ retiene la condición esencial que subyace a la ausencia de pasos de cómputo innecesarios, a saber: cuando se aplica una cierta regla de programa cuyo lado izquierdo ha sido seleccionado e instanciado con ayuda de los árboles definicionales, se puede asegurar que la posición en la que se va a aplicar dicha regla debe ser reducida necesariamente a forma estable para que el objetivo llegue a ser resuelto. Lo que no se puede asegurar, debido al indeterminismo (y como ya sucedía en [Ant97]) es que la regla seleccionada (y no otra con el mismo lado izquierdo) sea la que finalmente conduzca a la solución. Se puede además utilizar el cálculo de reescritura con restricciones $CRWL(\mathcal{D})$ para dar una justificación semántica de la necesidad de reducir a forma estable las posiciones detectadas como necesarias con ayuda de los árboles definicionales. Esta idea fue explorada incipientemente en [Vad02], y su desarrollo formal en el marco del esquema $CFLP(\mathcal{D})$ de esta tesis puede ser una línea de trabajo futuro.

4.3.8. Transformación de $CFLP(\mathcal{D})$-programas

A pesar de que el cálculo $CDNC(\mathcal{D})$ sólo es aplicable sobre la clase especial de $CFLP(\mathcal{D})$-programas denominada $COISS(\mathcal{D})$-programas, cuyas reglas de programa admiten una organización en forma de árbol definicional, esta restricción no supone en la práctica un inconveniente serio de aplicabilidad. Con el fin de demostrar que los $COISS(\mathcal{D})$-programas tienen la misma expresividad que los $CFLP(\mathcal{D})$-programas, vamos a proponer en esta sección un *algoritmo de transformación de $CFLP(\mathcal{D})$-programas*, basado en las técnicas de transformación descritas en [LLR93], con el fin de probar que cualquier $CFLP(\mathcal{D})$-programa puede ser transformado, de una manera efectiva, en un $COISS(\mathcal{D})$-programa que preserva la semántica declarativa de $CRWL(\mathcal{D})$. Vamos a comenzar definiendo algunas notaciones auxiliares que resultarán de utilidad en la descripción del algoritmo.

Definición 27 (Posiciones Demandadas) *Sea $\mathcal{P}$ un $CFLP(\mathcal{D})$-programa. Un* patrón genérico de llamada *es cualquier patrón de llamada de la forma $f\overline{X}_n$, donde $\overline{X}_n$ son variables diferentes. Sea τ un patrón de llamada. Decimos que:*

(1) Una regla de programa $(l \to r \Leftarrow P \,\square\, C) \in \mathcal{P}$ se ajusta *al patrón de llamada τ si y sólo si $\tau \preceq l$.*

(2) Una posición $p \in VPos(\tau)$ es demandada por el lado izquierdo *l de una regla de programa en $\mathcal{P}$ que se ajusta al patrón de llamada τ si y sólo si l tiene una aparición de un símbolo pasivo en la posición p.*

(3) Una posición $p \in VPos(\tau)$ es demandada *si y sólo si p es una posición demandada por el lado izquierdo de una regla de programa en $\mathcal{P}$ que se ajusta al patrón de llamada τ.*

(4) Una posición $p \in VPos(\tau)$ es uniformemente demandada *si y sólo si p es una posición demandada por cualquier lado izquierdo de una regla de programa en $\mathcal{P}$ que se ajusta al patrón de llamada τ.*

Sea $\mathcal{P}$ un *CFLP*($\mathcal{D}$)-programa. El *algoritmo de transformación* de *CFLP*($\mathcal{D}$)-programas en *COISS*($\mathcal{D}$)-programas se ejecuta mediante la invocación de una función denominada *COIS* (del inglés, *Constrained Overlapping Inductively Sequential*) definida sobre el dominio de restricciones $\mathcal{D}$, la cual toma como argumento el *CFLP*($\mathcal{D}$)-programa $\mathcal{P}$ y devuelve como resultado un *COISS*($\mathcal{D}$)-programa equivalente. Para transformar programas completos basta simplemente con aplicar este algoritmo a cada una de las funciones definidas $f \in DF^n$ del programa $\mathcal{P}$ mediante la *llamada inicial*

$$COIS(\mathcal{P}) = \bigcup_{f \in DF} COIS(f\overline{X}_n, \mathcal{P}_f)$$

donde $f\overline{X}_n$ es un patrón genérico de llamada para cada $f \in DF^n$, y $\mathcal{P}_f$ es el subconjunto de $\mathcal{P}$ formado por todas aquellas reglas de programa que definen f (obviamente, $\mathcal{P} = \bigcup_{f \in DF} \mathcal{P}_f$). Las llamadas recursivas tendrán la forma $COIS(\tau, \mathcal{P})$, donde τ es un patrón de llamada y $\mathcal{P}$ es un conjunto de *CFLP*($\mathcal{D}$)-reglas de programa ajustadas por τ. Distinguimos los siguientes casos:

1) *Alguna posición en $VPos(\tau)$ es uniformemente demandada.*
Sea $p \in VPos(\tau)$ la posición más a la izquierda que cumpla esta condición y sea X la variable en la posición p en τ. Sean $h_1, \ldots, h_m$ los símbolos pasivos que aparecen en la posición p en los lados izquierdos de reglas de programas de $\mathcal{P}$ ajustadas por τ. Definimos $COIS(\tau, \mathcal{P}) = \bigcup_{1 \leq i \leq m} COIS(\tau_i, \mathcal{P}_i)$, donde:

   - $\tau_i = \tau\{X \mapsto h_i\overline{X}_{r_i}\}$, siendo $\overline{X}_{r_i}$ variables nuevas y r_i la aridad de h_i.
   - $\mathcal{P}_i = \{(l \to r \Leftarrow P \,\square\, C) \in \mathcal{P} \mid l \preceq \tau_i\}$.

   En este caso, $\mathcal{T}_\tau$ tiene la forma $\underline{caso}(\tau, X, [\mathcal{T}_{\tau_1}, \ldots, \mathcal{T}_{\tau_m}])$, donde los árboles definicionales $\mathcal{T}_{\tau_i}$ son creados recursivamente.

2) *Ninguna posición en $VPos(\tau)$ es demandada.*
En este caso, todos los lados izquierdos de $\mathcal{P}$ que son ajustados por τ son en realidad variantes de τ, por lo que podemos definir $COIS(\tau, \mathcal{P}) = \mathcal{P}$. Más aún, $\mathcal{T}_\tau$ tiene la estructura $\underline{regla}(\tau \to r_1 \Leftarrow P_1 \square C_1 | \ldots | r_m \Leftarrow P_m \square C_m)$.

3) *Alguna posición en $VPos(\tau)$ es demandada, pero ninguna es uniformemente demandada.*
Sea $p \in VPos(\tau)$ la posición más a la izquierda que satisfaga tal condición. Tomamos $\mathcal{P}_p^+ = \{R \in \mathcal{P} \mid$ el lado izquierdo de R demanda $p\ \}$ y $\mathcal{P}_p^- = \mathcal{P} \backslash \mathcal{P}_p^+$. Vamos a considerar dos nuevos símbolos de función, f_1 y f_2, con las mismas aridades que f (el símbolo de función definida en la raíz de τ), y definimos entonces $COIS(\tau, \mathcal{P}) = COIS(\tau_1, \mathcal{P}_1) \cup COIS(\tau_2, \mathcal{P}_2) \cup \{\tau \to \tau_1, \tau \to \tau_2\}$, donde:

   - τ_1 (respectivamente, τ_2) es el nuevo patrón de llamada que es obtenido al cambiar la aparición de f en la raíz de τ por el nuevo símbolo de función f_1 (respectivamente, f_2).
   - $\mathcal{P}_1$ (respectivamente, $\mathcal{P}_2$) es el conjunto de reglas de programa construido a partir de $\mathcal{P}_p^+$ (respectivamente, a partir de $\mathcal{P}_p^-$) mediante el cambio de la aparición de f en los lados izquierdos de sus reglas de programa por el nuevo símbolo de función f_1 (respectivamente, f_2).

   El árbol definicional $\mathcal{T}_\tau$ tiene la forma $\underline{regla}(\tau \to \tau_1 | \tau_2)$, donde $\mathcal{T}_{\tau_1}$ y $\mathcal{T}_{\tau_2}$ son árboles definicionales creados recursivamente.

Ejemplo 17 *Ilustramos mediante un ejemplo concreto la aplicación del algoritmo de transformación de programas. Vamos a considerar el siguiente $CFLP(\mathcal{FD})$-programa $\mathcal{P}$ cuyas reglas definen una función check_list que no es inductivamente secuencial, es decir, que no pueden ser organizadas en un árbol definicional.*

$$
\begin{array}{llcll}
check\_list & [\,] & \to & 0 & \\
check\_list & [X|Xs] & \to & 1 & \Leftarrow \quad X /= 1 \\
check\_list & [X|Xs] & \to & 2 & \Leftarrow \quad X /= 0 \\
check\_list & [X,Y|Zs] & \to & 3 & \Leftarrow \quad X /= Y
\end{array}
$$

Vamos a obtener un $COISS(\mathcal{D})$-programa $\mathcal{P}'$ equivalente a $\mathcal{P}$ mediante la aplicación del algoritmo de transformación de programas:

$\mathcal{P}' = COIS(\mathcal{P}) = COIS(check\_list\ L, \mathcal{P}) =$

Puesto que L está en una posición uniformemente demandada, consideramos:

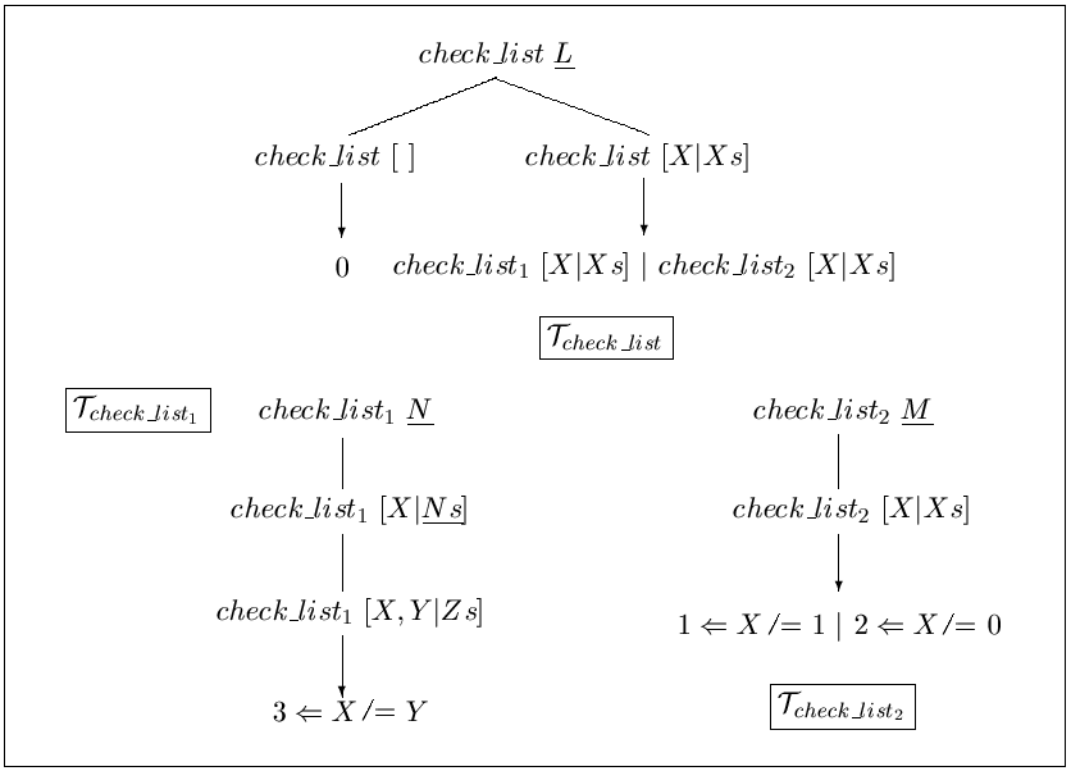


Figura 4.5: Árboles definicionales con restricciones para *check_list*

$\mathcal{P}_1 = \{check\_list\ [\,] \rightarrow 0\}$,
$\mathcal{P}_2 = \mathcal{P} \setminus \mathcal{P}_1$

$= COIS(check\_list\ [\,], \mathcal{P}_1) \cup COIS(check\_list\ [X|Xs], \mathcal{P}_2) =$

- $COIS(check\_list\ [\,], \mathcal{P}_1) = \mathcal{P}_1$, *debido a que no hay posiciones demandadas.*
- $COIS(check\_list\ [X|Xs], \mathcal{P}_2) =$

Xs está en una posición demandada, pero no uniformemente demandada. Por tanto, consideramos:

$\mathcal{P}_2^+ = \{check\_list\ [X,Y|Zs] \ \rightarrow\ 3 \Leftarrow X\ /= Y\}$,
$\mathcal{P}_2^- = \mathcal{P}_2 \setminus \mathcal{P}_2^+$

$\mathcal{P}_{21} = \{check\_list_1\ [X,Y|Zs] \ \rightarrow\ 3 \Leftarrow X\ /= Y\}$,
$\mathcal{P}_{22} = \{check\_list_2\ [X|Xs] \ \rightarrow\ 1 \Leftarrow X\ /= 1,$
$\quad check\_list_2\ [X|Xs] \ \rightarrow\ 2 \Leftarrow X\ /= 0\}$

$= COIS(check\_list_1\ [X|Xs], \mathcal{P}_{21}) \cup COIS(check\_list_2\ [X|Xs], \mathcal{P}_{22})$
$\cup \{check\_list\ [X|Xs] \rightarrow check\_list_1\ [X|Xs],$
$check\_list\ [X|Xs] \rightarrow check\_list_2\ [X|Xs]\} =$

- $COIS(check\_list_1\ [X|Xs], \mathcal{P}_{21}) = COIS(check\_list_1\ [X, Y|\ Zs],\ \mathcal{P}_{21}) = \mathcal{P}_{21}$, *porque* Xs *está en una posición uniformemente demandada, y* X, Y, Zs *no son demandadas.*

- $COIS(check\_list_2\ [X|Xs], \mathcal{P}_{22}) = \mathcal{P}_{22}$, *porque ninguna posición es demandada.*

Obtenemos $\mathcal{P}' = \mathcal{P}_1 \cup \mathcal{P}_{21} \cup \mathcal{P}_{22} \cup \{check\_list\ [X|Xs] \rightarrow check\_list_1\ [X|Xs]$, $check\_list\ [X|Xs] \rightarrow check\_list_2\ [X|Xs]\}$. *Una representación gráfica de los árboles definicionales obtenidos se da en la Figura 4.5.*

Finalmente, probamos la principal propiedad del algoritmo *COIS* de transformación de $CFLP(\mathcal{D})$-programas. Demostramos que el algoritmo de transformación preserva la semántica $CRWL(\mathcal{D})$ de los programas, esto es: dado un $CFLP(\mathcal{D})$-programa $\mathcal{P}$ y un $COISS(\mathcal{D})$-programa $\mathcal{P}'$ resultado de transformar $\mathcal{P}$, son derivables en $CRWL(\mathcal{D})$ las mismas c-sentencias a partir de $\mathcal{P}$ que de $\mathcal{P}'$.

Teorema 10 (Propiedades de *COIS*) *Para cualquier* $CFLP(\mathcal{D})$*-programa* $\mathcal{P}$ *dado,* $COIS(\mathcal{P})$ *siempre termina y lo hace devolviendo otro* $CFLP(\mathcal{D})$*-programa* $\mathcal{P}'$ *tal que:*

(1) $\mathcal{P}'$ *es un* $COISS(\mathcal{D})$*-programa. Si* $\mathcal{P}$ *involucra un conjunto de símbolos de función* FS *entonces* $\mathcal{P}'$ *involucra un conjunto* FS' *tal que* $FS \subseteq FS'$*. Más aún, si* P *es un* $COISS(\mathcal{D})$*-programa entonces* $\mathcal{P}' \equiv \mathcal{P}$*.*

(2) Para cualquier c-sentencia φ *expresable en la signatura de* $\mathcal{P}$*, se tiene que* $\mathcal{P} \vdash_{\mathcal{D}} \varphi$ *si y sólo si* $\mathcal{P}' \vdash_{\mathcal{D}} \varphi$*.*

La demostración de este resultado se da en el Apéndice A.

Parte II: Técnicas de Depuración Declarativa para Programas en el Esquema $CFLP(\mathcal{D})$

Capítulo 5

Depuración declarativa de respuestas incorrectas

Las herramientas de depuración son una necesidad práctica a la hora de diagnosticar las causas de cómputos erróneos y de ayudar a los programadores a entender por qué sus programas no funcionan como se esperaba. Los paradigmas de programación declarativos, entre los que se incluye la programación lógico funcional con restricciones, y dentro de ella el esquema $CFLP(\mathcal{D})$ presentado en la primera parte de esta tesis, involucran casi siempre detalles operacionales complejos, como por ejemplo, la resolución de restricciones y la evaluación perezosa. Estas características no encajan bien con las técnicas de depuración tradicionales, las cuales recaen en la inspección de trazas de cómputo de bajo nivel. Como una posible solución a este problema, la *diagnosis declarativa* permite usar *Árboles de Cómputo* (abreviadamente, CTs) en lugar de trazas. Los CTs son construidos *a posteriori* para representar la estructura de un cómputo que es calificado, en su nivel más externo, como un *síntoma de error* por el usuario. Cada nodo en un CT permite representar la computación de algún resultado observable, dependiendo de los resultados de sus nodos hijos. Así, la diagnosis declarativa explora un CT buscando lo que se denominan *nodos críticos*, los cuales computan un resultado incorrecto a partir de hijos cuyos resultados son todos correctos; tales nodos críticos han de apuntar a un fragmento de programa incorrecto como responsable del cómputo erróneo. La búsqueda de nodos críticos puede ser implementada mediante la ayuda de un *oráculo* externo (usualmente el usuario con algún tipo de soporte semiautomático), quien tiene un conocimiento declarativo subyacente de la semántica esperada del programa, la denominada *interpretación pretendida* del programa.

La descripción genérica de la diagnosis declarativa que hemos descrito en el párrafo anterior sigue a [Nai97]. La diagnosis declarativa fue propuesta originariamente dentro del campo de la programación lógica [Sha82, Fer87, Llo87b] y ha sido satisfactoriamente extendida a otros paradigmas de programación, incluyendo

la programación funcional perezosa [NF94, NS97, Nil01b, PN03a, Pop07], la programación lógica con restricciones [TF00, FLT03], la programación lógico funcional [NB95, CLR01, CR02, CR04, Cab04] y la programación imperativa [SF89]. En el caso de la programación funcional, a los resultados inesperados observados después de algún cómputo se les denomina *valores incorrectos*, mientras que en programación lógica con restricciones y en programación lógico funcional, un resultado inesperado puede ser, o bien una sola respuesta computada calificada como incorrecta o bien un conjunto de respuestas computadas (para el mismo objetivo con un espacio de búsqueda finito) calificado como *incompleto*. Estas dos posibilidades dan lugar a la diagnosis declarativa de respuestas computadas *incorrectas* e *incompletas*, respectivamente. El caso particular de *fallo finito* que en ocasiones se obtiene es también un síntoma de respuestas perdidas de especial relevancia.

En contraste con las aproximaciones más recientes a la diagnosis de errores que hacen uso de *interpretación abstracta* (como por ejemplo [CC77, CC92, CLMV99, HPBL02, ABCF03] y algunas otras aproximaciones descritas en [DHM00]), la diagnosis declarativa siempre involucra preguntas complejas para el usuario. Este problema ha sido tratado mediante diversas técnicas, como el uso de *especificaciones parciales* dadas por el usuario de la semántica del programa [BDM97, CR04], *inferencias seguras de información* a partir de respuestas previamente dadas por el usuario [CR02] o incluso CTs guiados por la necesidad particular de un problema de depuración sobre un dominio de cómputo también particular [FLT03]. La investigación en diagnosis declarativa todavía tiene que resolver muchos aspectos que tienen que ver tanto con sus fundamentos más teóricos como con el desarrollo de herramientas prácticas. Algunos problemas relacionados con la eficiencia de las herramientas de depuración declarativa se mencionarán más adelante.

El propósito de este capítulo es el de presentar un método declarativo de diagnosis de respuestas computadas incorrectas en el esquema $CFLP(\mathcal{D})$, dejando para el Capítulo 6 la presentación de un método similar pero para el diagnóstico de respuestas incompletas (o también llamadas *respuestas perdidas*). Para ello, y siguiendo las ideas de la semántica declarativa de $CFLP(\mathcal{D})$ expuestas en el Capítulo 3, vamos a obtener una noción adecuada de interpretación pretendida, así como de un tipo de árboles de prueba apropiados con un significado lógico que jugarán el papel de CTs. El propósito es el de conseguir una combinación natural de aproximaciones previas que han sido desarrolladas por separado para el esquema $CLP(\mathcal{D})$ en [TF00] y para los lenguajes de programación lógico funcionales perezosos en [CR02]. Daremos resultados teóricos que demuestran que el método de depuración propuesto es lógicamente correcto para cualquier sistema de resolución de objetivos en el esquema $CFLP(\mathcal{D})$ cuyas respuestas computadas sean consecuencia lógica del programa en el sentido de la semántica de $CFLP(\mathcal{D})$ formalizada mediante la lógica para la reescritura $CRWL(\mathcal{D})$. Además, presentaremos un prototipo de depurador práctico, denominado $\mathcal{DDT}$, desarrollado como una extensión de herramientas previamente existentes pero menos poderosas en [CR04, Cab05]. $\mathcal{DDT}$ implementa el método

de diagnosis propuesto para $CFLP(\mathcal{R})$-programación en el sistema $\mathcal{TOY}$ [ALR07] usando el dominio $\mathcal{R}$ de las restricciones aritméticas sobre los números reales.

Comenzamos este capítulo con una sección inicial que permite motivar nuestra propuesta mediante un ejemplo que será utilizado de manera recurrente a lo largo de todo el capítulo. A continuación, extenderemos el esquema $CFLP(\mathcal{D})$ para poder fundamentar los principales resultados teóricos que se describen en este capítulo. Presentaremos un método para la diagnosis declarativa de respuestas computadas incorrectas en cualquier sistema admisible de resolución de objetivos para $CFLP(\mathcal{D})$. Finalmente, describiremos la herramienta de depuración $\mathcal{DDT}$. A lo largo del capítulo incluiremos también las demostraciones de los principales resultados que se han obtenido.

5.1. Ejemplos motivadores

Como motivación para el resto del capítulo, vamos a considerar el siguiente fragmento de programa escrito en el sistema de programación $\mathcal{TOY}$, el cual soporta varias de las instancias del esquema $CFLP(\mathcal{D})$ presentadas en el Capítulo 2.

Ejemplo 18 (Construyendo escaleras en $\mathcal{TOY}$)

```
infixr 40 &&
(&&) :: bool → bool → bool
false && Y = false
true && Y = Y

head :: [A] → A
head [X|Xs] = X

type point = (real,real)
type figure = point → bool

rect :: point → real → real → figure
rect (X,Y) LX LY (X',Y') = (X'≥X) && (X' ≤ X+LX) && (Y'≥Y) && (Y' ≤ Y+LY)
% Incorrecto. Debería ser:                          ... (Y'≥Y) ...

intersect :: figure → figure → figure
intersect F1 F2 P = F1 P && F2 P

ladder :: point → real → real → [figure]
ladder (X,Y) LX LY = [rect (X,Y) LX LY | ladder (X+LX, Y+LY) LX LY]
```

En este caso, el sistema $\mathcal{TOY}$ se ha utilizado para implementar la instancia $CFLP(\mathcal{R})$ del esquema $CFLP(\mathcal{D})$, donde el parámetro $\mathcal{D}$ se reemplaza por el dominio de los

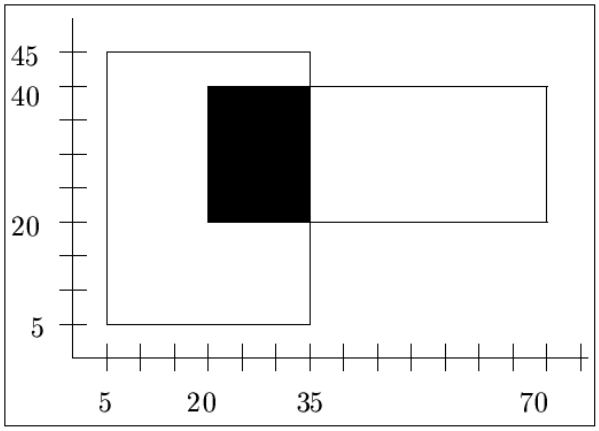


Figura 5.1: Intersección de dos rectángulos

números reales $\mathcal{R}$, descrito en la Subsección 2.2.2, el cual proporciona números reales, operaciones aritméticas y varias restricciones aritméticas, incluyendo igualdades, desigualdades e inecuaciones. El tipo figure es utilizado para representar figuras geométricas como funciones booleanas, la función rect es utilizada para representar rectángulos (más precisamente, un rectángulo (rect (X,Y) LX LY) significa que tiene dos vértices opuestos de coordenadas (X,Y) y (X+LX,Y+LY), respectivamente); y la función ladder es utilizada para construir una lista infinita de rectángulos en la forma de una escalera. Aunque el texto del programa no parezca que incluya restricciones, este usa operadores aritméticos y de comparación que dan lugar a la resolución de restricciones en tiempo de ejecución. De manera más precisa, vamos a considerar la siguiente sesión en $\mathcal{TOY}$:

```
Toy> /run(examples/debug/ladder)        % compila el programa ladder.toy
Toy> /cflpr                             % carga CFLP(R)

Toy(R)> intersect (head (ladder (20,20) 50 20))                  % objetivo
                  (head (ladder (5,5) 30 40)) (X,Y) == R

{R → true} {Y≤ 5, X≥2.0E+01, X≤35}      % respuesta computada
```

El objetivo pregunta por la pertenencia de un punto genérico (X,Y) a la intersección de los dos rectángulos (rect (20,20) 50 20) y (rect (5,5) 30 40), computados indirectamente como los primeros peldaños de dos escaleras particulares de rectángulos. El diagrama que aparece en la Figura 5.1 muestra estos dos rectángulos, así como el rectángulo correspondiente a su intersección (resaltado en negro). El sistema $\mathcal{TOY}$ ha resuelto el objetivo mediante una combinación de estrechamiento perezoso y reso-

lución de restricciones de una manera similar a como se ha descrito en el Capítulo 4 para el cálculo $CDNC(\mathcal{D})$; la respuesta computada está formada por una sustitución $\mathsf{R} \rightarrow \mathsf{true}$ y tres restricciones impuestas sobre las variables X e Y [1]. La única restricción impuesta sobre Y (la restricción $\mathsf{Y} \leq \mathsf{5}$) permite valores arbitrariamente pequeños de Y, los cuales no pueden corresponderse con puntos que pertenezcan al rectángulo esperado como intersección. Por tanto, el usuario verá la respuesta computada como incorrecta con respecto al significado pretendido del programa. Como veremos en las siguientes secciones, la técnica de depuración declarativa presentada en este capítulo conduce al diagnóstico de la regla de programa para la función rect como responsable de la respuesta incorrecta. De hecho, la regla de programa es incorrecta con respecto a la semántica pretendida del programa; como se muestra en el Ejemplo 18, la tercera inecuación en el lado derecho debería ser $\mathsf{Y'} \geq \mathsf{Y}$ en lugar de $\mathsf{Y'} \leq \mathsf{Y}$. Tras esta corrección, ninguna otra respuesta errónea computada se observa ya para el objetivo que hemos discutido.

La aproximación que utilizaremos en este capítulo para la depuración de respuestas incorrectas usará CTs cuyos nodos tengan asociados *c-hechos* de la forma $f\bar{t}_n \rightarrow t \Leftarrow \Pi$ (como los que se han introducido y explicado en el Capítulo 3) además de un c-hecho especial en la raíz de la forma $G_0\sigma \Leftarrow \Pi$, siendo G_0 el objetivo inicial y $\Pi \ \square \ \sigma$ un objetivo en forma resuelta que representa una respuesta calculada considerada por el usuario como errónea. Más aún, el CT total debe ser tal que la validez del c-hecho en cada nodo se siga a partir de la validez de los c-hechos de sus hijos, bajo la suposición de que la regla de programa que relaciona el nodo padre con los nodos hijos es correcta con respecto a la interpretación pretendida del programa. Conseguiremos que se satisfaga este requisito construyendo el CT como un árbol de prueba abreviado con respecto a un sistema de inferencia lógicamente correcto para derivar c-hechos, directamente adaptado de la lógica para la reescritura con restricciones $CRWL(\mathcal{D})$. Una vez que un CT haya sido construido, la búsqueda de un *nodo crítico* podrá ser implementada con la ayuda de un oráculo externo (usualmente el programador) que tenga un conocimiento declarativo de la validez de los c-hechos basado en una interpretación pretendida conocida del programa. Cualquier CT con un c-hecho inválido en su raíz va a tener con seguridad al menos un nodo crítico etiquetado con un c-hecho inválido y cuyos hijos están todos etiquetados con c-hechos válidos. Cada nodo crítico estará relacionado con alguna regla de programa responsable de la computación del c-hecho a partir de los c-hechos de los hijos. En consecuencia, esta regla de programa será diagnosticada como incorrecta.

Como cualquier técnica de depuración, la diagnosis declarativa tiene limitaciones. Un fragmento de programa corregido puede incluir todavía otros errores más sutiles que pueden ser observados en las respuestas computadas para otros objetivos. En nuestro caso, podemos considerar el siguiente objetivo:

[1] Hay otras cinco respuestas computadas formadas por la sustitución $\mathsf{R} \rightarrow \mathsf{false}$ y varias restricciones impuestas sobre X e Y.

```
Toy(R)> intersect (head (ladder (70,40) -50 -20))
                  (head (ladder (35,45) -30 -40)) (X,Y) == R
```

cuyo significado con respecto a la semántica pretendida es el mismo que para el objetivo previo, excepto que los rectángulos que juegan el papel de pasos iniciales de las dos escaleras son representados ahora de manera diferente. Puesto que la expresión booleana en el lado derecho de la regla de programa "corregida" para la función rect da lugar al resultado false siempre que LX o LY se liga a un número negativo, las respuestas incorrectas que incluyen la sustitución R → false son las que van a ser computadas. Más aún, otras respuestas que incluyan la sustitución R → true serán también esperadas por el usuario, pero no aparecerán entre las respuestas computadas obtenidas. La aproximación tradicional a la depuración declarativa en el esquema $CLP(\mathcal{D})$ incluye el diagnóstico tanto de respuestas computadas *incorrectas* como de respuestas *perdidas* [TF00]. Como ya se ha comentado, la diagnosis declarativa de respuestas perdidas en $CFLP(\mathcal{D})$ será abordada en el Capítulo 6.

5.2. Interpretación pretendida de un programa

En la Definición 9 del Capítulo 3 se define una *c-interpretación* sobre un dominio de restricciones $\mathcal{D}$ como cualquier conjunto $\mathcal{I}$ de c-hechos que incluya todos los c-hechos triviales y sea cerrado bajo $\mathcal{D}$-implicación. Equivalentemente, una c-interpretación es cualquier conjunto $\mathcal{I}$ de c-hechos tal que $cl_{\mathcal{D}}(\mathcal{I}) \subseteq \mathcal{I}$, donde $cl_{\mathcal{D}}(\mathcal{I})$ se definía como sigue: $cl_{\mathcal{D}}(\mathcal{I}) = \{\varphi' \mid \varphi'$ es un c-hecho trivial o bien $\exists \varphi \in \mathcal{I}\,(\varphi \succcurlyeq_{\mathcal{D}} \varphi')\}$. Comenzamos esta sección observando que dos c-interpretaciones distintas pueden incluir los mismos c-hechos cerrados, como se muestra en el ejemplo siguiente.

Ejemplo 19 *Para un $CFLP(\mathcal{R})$-programa compuesto únicamente por la regla:*

$$notZero\ N \rightarrow true \Leftarrow N * N \not= 0$$

vamos a considerar los siguiente c-hechos : $\varphi = notZero\,N \rightarrow true \Leftarrow N*N \not= 0$; $\varphi_1 = notZero\,N \rightarrow true \Leftarrow N > 0$; $\varphi_2 = notZero\,N \rightarrow true \Leftarrow N < 0$. Se tiene entonces que $\varphi \succcurlyeq_{\mathcal{D}} \varphi_1$ (debido a que $N > 0 \models_{\mathcal{R}} N*N \not= 0$) pero $\varphi_1 \not\succcurlyeq_{\mathcal{D}} \varphi$ (debido a que $N*N \not= 0 \not\models_{\mathcal{R}} N > 0$). Análogamente, $\varphi \succcurlyeq_{\mathcal{D}} \varphi_2$ pero $\varphi_2 \not\succcurlyeq_{\mathcal{D}} \varphi$. Por tanto, es posible construir dos c-interpretaciones diferentes $\mathcal{I} \subset \mathcal{J}$ sobre $\mathcal{R}$ incluyendo los mismos c-hechos cerrados, tales que $\varphi_1 \in \mathcal{I} \cap \mathcal{J}$, $\varphi_2 \in \mathcal{I} \cap \mathcal{J}$ y $\varphi \in \mathcal{J} \setminus \mathcal{I}$.

En lo sucesivo, escribimos $\mathcal{I} \Vdash_{\mathcal{D}} \varphi$ para indicar que la c-sentencia φ (no necesariamente un c-hecho) es semánticamente válida en la c-interpretación $\mathcal{I}$ en el sentido indicado en la Definición 10 del Capítulo 3. Estamos ya en disposición de definir algunas nociones semánticas de utilidad para la depuración declarativa de respuestas incorrectas que complementan a las que se introdujeron en el Capítulo 3 en la

Definición 11 y que también dependen de una c-interpretación dada $\mathcal{I}$ sobre $\mathcal{D}$. En concreto, nos interesa definir formalmente el concepto de conjunto de soluciones de un objetivo admisible.

Tal y como se ha explicado en la Sección 5.1, la técnica de depuración de respuestas incorrectas que proponemos en este capítulo se basa en un cálculo lógico que nos permite inferir enunciados de la forma $G_0\sigma \Leftarrow \Pi$, siendo G_0 un objetivo inicial y $\Pi \ \square \ \sigma$ un objetivo en forma resuelta representando una respuesta computada que el usuario considera como incorrecta. Por este motivo, podemos suponer que los enunciados derivables en el cálculo serán de la forma $G \Leftarrow \Pi$, donde Π es un conjunto de restricciones primitivas y G es un objetivo compuesto por producciones y restricciones. Esta observación nos permite simplificar en el contexto de este capítulo la sintaxis general utilizada en el Capítulo 4 para representar un objetivo admisible $G \equiv \exists\overline{U}.\ P \ \square \ C \ \square \ \Pi \ \square \ \sigma$ en la forma reducida $G \equiv \exists\overline{U}.\ (P \ \square \ \Delta)$, donde la parte P consta de producciones y la parte Δ de restricciones, las cuales pueden ser tanto primitivas como definidas por el usuario. Así pues, la parte del objetivo compuesta por la sustitución σ dejará de utilizarse en la sintaxis de los objetivos admisibles en todo este capítulo. Con este convenio de notación, para definir el conjunto de soluciones de un objetivo admisible dado en la forma $G \equiv \exists\overline{U}.\ (P \ \square \ \Delta)$, tendremos primero que definir el conjunto de soluciones de un conjunto de producciones P, ya que la Definición 11 del Capítulo 3 recoge con precisión cuál es el conjunto de soluciones correspondiente al conjunto de restricciones Δ.

Definición 28 (Nociones Semánticas Dependientes de Interpretaciones)

(1) El conjunto de las soluciones de una producción $e \to t$ es un subconjunto $Sol_{\mathcal{I}}(e \to t) \subseteq Val_{\mathcal{D}}$ definido como $Sol_{\mathcal{I}}(e \to t) = \{\eta \in Val_{\mathcal{D}} \mid \mathcal{I} \Vdash_{\mathcal{D}} e\eta \to t\eta\}$. El conjunto de soluciones de un conjunto de producciones P está definido como $Sol_{\mathcal{I}}(P) = \bigcap_{(e\to t)\in P} Sol_{\mathcal{I}}(e \to t)$.

(2) El conjunto de soluciones de un objetivo admisible $G \equiv \exists\overline{U}.\ (P \ \square \ \Delta)$ es un subconjunto $Sol_{\mathcal{I}}(G) \subseteq Val_{\mathcal{D}}$ definido como sigue: $Sol_{\mathcal{I}}(G) = \{\eta \in Val_{\mathcal{D}} \mid \eta' \in Sol_{\mathcal{I}}(P) \cap Sol_{\mathcal{I}}(\Delta)$ para algún $\eta' =_{\setminus\overline{U}} \eta\}$, donde $Sol_{\mathcal{I}}(\Delta)$ se define como en la Definición 11 del Capítulo 3.

El Lema 8, su Corolario 7 y el Lema 9 enunciados en la Subsección 4.2.1 son válidos también cuando se sustituye "$Sol_{\mathcal{P}}$" por "$Sol_{\mathcal{I}}$", siendo $\mathcal{I}$ cualquier c-interpretación.

Las técnicas de diagnosis declarativa recaen sobre una descripción declarativa de la semántica pretendida de los programas. Asumiremos que el usuario conoce (al menos hasta la extensión necesaria para responder cuestiones durante la sesión de depuración) una *interpretación pretendida* $\mathcal{I}$ del programa $\mathcal{P}$, la cual es una c-interpretación para la que se espera que se satisfaga que $\mathcal{I}$ sea modelo del programa $\mathcal{P}$, a menos que $\mathcal{P}$ sea incorrecto. Por ejemplo, $rect\ (X,Y)\ LX\ LY\ (A,B) \to false \Leftarrow A < X$ podría pertenecer al modelo pretendido $\mathcal{I}$ para el fragmento de programa que se muestra en el Ejemplo 18.

De entre las dos posibles semánticas de modelos presentadas para $CFLP(\mathcal{D})$ en el Capítulo 3, vamos a optar por utilizar en la depuración declarativa de respuestas incorrectas la noción de *semántica débil* introducida en la Definición 13, la cual depende exactamente de los c-hechos cerrados pertenecientes a $\mathcal{I}$. Esta elección se puede motivar a través del propio Ejemplo 19: como se explicó en el Capítulo 3, un c-hecho φ que pertenezca a una c-interpretación puede no ser cerrado, incluyendo variables. Usando semántica fuerte, el que un c-hecho como φ sea válido o no va a depender de si el usuario está pensando en la c-interpretación $\mathcal{J}$ o en la c-interpretación $\mathcal{I}$ como modelo pretendido de su programa. Sin embargo, con semántica débil la validez de φ se hace menos dependiente del conjunto de c-hechos abiertos que el usuario escoja como modelo pretendido de su programa $\mathcal{P}$. En lo sucesivo emplearemos la notación $\mathcal{I} \models_{\mathcal{D}} \mathcal{P}$ para expresar que la c-interpretación $\mathcal{I}$ sobre el dominio $\mathcal{D}$ es un modelo débil del programa $\mathcal{P}$ (escrito antes como $\mathcal{I} \models^{w}_{\mathcal{D}} \mathcal{P}$ en el Capítulo 3). Por tanto, presentaciones diferentes del modelo pretendido de un programa serán equivalentes para el propósito de este trabajo, siempre que incluyan los mismos c-hechos cerrados.

5.3. Síntomas y errores positivos

La diagnosis declarativa comienza con un *síntoma* observado del comportamiento erróneo de un programa y persigue detectar algún *error* en dicho programa que pueda ser la causa del síntoma observado. Las nociones apropiadas de síntoma y de error usadas en el escenario de depuración de respuestas incorrectas en el esquema $CFLP(\mathcal{D})$ son las siguientes:

Definición 29 (Escenario de Depuración de Respuestas Incorrectas) *Asumamos que $\mathcal{I}$ es la interpretación pretendida para un programa $\mathcal{P}$ y consideremos una forma resuelta $S = \Pi \square \sigma$ producida como respuesta computada para un objetivo inicial G_0 por algún sistema de resolución de objetivos en el esquema $CFLP(\mathcal{D})$.*

(1) Un **síntoma de incorrección** *ocurre si el programador juzga que $Sol_{\mathcal{D}}(S) \not\subseteq Sol_{\mathcal{I}}(G_0)$ (o lo que es lo mismo, $Sol_{\mathcal{D}}(\Pi) \not\subseteq Sol_{\mathcal{I}}(G_0\sigma)$), mostrando así que S es una* respuesta errónea *con respecto a $\mathcal{I}$.*

(2) $\mathcal{P}$ es incorrecto *con respecto a $\mathcal{I}$ si y sólo si existe alguna regla de programa $(f\bar{t}_n \rightarrow r \Leftarrow P \square \Delta) \in \mathcal{P}$ (manifestando un* error*) que no es válida en $\mathcal{I}$ (en el sentido precisado en el apartado (2) de la Definición 13 del Capítulo 3). Un* **diagnóstico de incorrección** *se proporciona apuntando a una regla de programa que no es válida en $\mathcal{I}$.*

En cualquier escenario de depuración de respuestas incorrectas como el que acabamos de fijar supondremos además que el cómputo que ha dado lugar a la respuesta incorrecta observada se ha producido sin errores de tipo y sin invocaciones incorrectas al resolutor.

Como ejemplo concreto, la respuesta computada mostrada en el Ejemplo 18 es incorrecta con respecto al modelo pretendido para el programa asumido en aquel ejemplo, por las razones ya discutidas en la Sección 5.1. Como ilustraba el ejemplo, las respuestas computadas incluyen típicamente restricciones sobre las variables que aparecen en el objetivo inicial. Sin embargo, los sistemas de resolución de objetivos para $CFLP(\mathcal{D})$-programas también mantienen información interna sobre restricciones relativas a variables usadas en pasos de cómputo intermedios, pero que no aparecen en el objetivo inicial. Esta información es relevante para los propósitos de la depuración declarativa. Por tanto, en el resto de este capítulo asumiremos que las respuestas computadas $S = \Pi \square \sigma$ son tales que Π también incluye restricciones relativas a variables intermedias. El Π correspondiente al síntoma de error inicialmente observado por el usuario será el que aparezca en los c-hechos de todos los nodos del CT usado en una sesión de depuración.

Ejemplo 20 ($CFLP(\mathcal{R})$-programa incorrecto y respuesta errónea)

$$
\begin{array}{llllll}
from & N & \rightarrow & [N \mid from\ (N-1)] & \Leftarrow & N > 0 \\
from & N & \rightarrow & [\,] & \Leftarrow & N == 0 \\
from & N & \rightarrow & [N \mid from\ (N+1)] & \Leftarrow & N < 0 \\
check & [\,] & \rightarrow & 0 & & \\
check & [X \mid Xs] & \rightarrow & 1 & \Leftarrow & X > 0 \\
check & [X \mid Xs] & \rightarrow & 2 & \Leftarrow & X < 0 \\
check & [X, Y \mid Zs] & \rightarrow & 3 & \Leftarrow & X < Y
\end{array}
$$

Asumamos que la interpretación pretendida de este programa es tal que todas las instancias válidas cerradas del c-hecho check $[X, Y|Zs] \rightarrow 3$ *deben satisfacer la restricción* $X < Y$*. Asumamos asimismo que la última regla de este programa, check* $[X, Y|Zs] \rightarrow 3 \Leftarrow X < Y$ *se ha visto afectada por un error, llegando a ser check* $[X, Y|Zs] \rightarrow 3 \Leftarrow X \not= Y$*. Consideremos que el programa incorrecto resultante es* $\mathcal{P}$ *y el objetivo inicial check* $(from\ A) == B$*. Entonces, los sistemas de resolución de objetivos como* $\mathcal{TOY}$ *(o también los cálculos formales de resolución de objetivos presentados en el Capítulo 4) pueden obtener la respuesta computada* $S \equiv \{A > 1\} \square \{B \mapsto 3\}$ *(o más exactamente, calculan la respuesta* $\exists C.\ (\{A > 0, A - 1 \rightarrow! C, C > 0, A \not= C\} \square \{B \mapsto 3\})$*, incluyendo una restricción que afecta a una variable intermedia* C*). Se observa así que* S *es incorrecta en la interpretación pretendida, debido a que incluye soluciones en las que check devuelve el valor* 3 *a partir de una lista cuyo segundo elemento no es mayor que el primero.*

5.4. El cálculo positivo $CPPC(\mathcal{D})$

Asumiendo que $S = \exists \overline{U}.\ (\Pi \square \sigma)$ es una respuesta computada (en la que se han cuantificado existencialmente todas las posibles variables intermedias) para un ob-

jetivo inicial G_0 usando el $CFLP(\mathcal{D})$-programa $\mathcal{P}$, la diagnosis declarativa de respuestas incorrectas necesita un CT adecuado para representar la computación realizada. En nuestro contexto, obtendremos este CT a partir de una prueba lógica $\mathcal{P} \vdash_{CPPC(\mathcal{D})} G_0 \Leftarrow S$, la cual deriva la sentencia $G_0 \Leftarrow S$ a partir del programa $\mathcal{P}$ en el denominado *Cálculo de Prueba Positivo con Restricciones* $CPPC(\mathcal{D})$ (del inglés, *Constraint Positive Proof Calculus*), definido a partir de las reglas de inferencia del cálculo para la reescritura con restricciones $CRWL(\mathcal{D})$ dadas en la Definición 16 del Capítulo 3. En concreto, además de las reglas **TI**, **RR**, **SP**, **DC**, **IR**, **PF** y **AC** de $CRWL(\mathcal{D})$, el cálculo $CPPC(\mathcal{D})$ incorpora una nueva regla **EX** para poder tratar los cuantificadores existenciales en respuestas computadas y una reformulación de la regla $\mathbf{DF}_{\mathcal{P}}$ como se muestra a continuación. Para este cálculo diremos que la $CPPC(\mathcal{D})$-prueba $\mathcal{P} \vdash_{CPPC(\mathcal{D})} G_0 \Leftarrow S$ sirve de *testigo* a la respuesta computada S para el objetivo inicial G_0.

EX Existencial

$$\frac{G_0\sigma \ \Leftarrow \ \Pi}{G_0 \Leftarrow \ \exists \overline{U}.\ (\Pi \ \square \ \sigma)}$$

si $fvar(G_0) \cap \overline{U} = \emptyset$.

$\mathbf{DF}_{\mathcal{P}}$ $\mathcal{P}$-Función Definida

$$(\mathbf{AR}_f) + (\mathbf{FA}_f) \quad \frac{e_1 \to t_1 \Leftarrow \Pi \ \ldots \ e_n \to t_n \Leftarrow \Pi \quad \boxed{f\ \bar{t}_n \to t \Leftarrow \Pi} \qquad \dfrac{P \ \square \ \Delta \Leftarrow \Pi \qquad r \to t \Leftarrow \Pi}{}}{f\ \bar{e}_n \to t \Leftarrow \Pi}$$

$$(\mathbf{AR}_f) + (\mathbf{FA}_f) \quad \frac{e_1 \to t_1 \Leftarrow \Pi \ \ldots \ e_n \to t_n \Leftarrow \Pi \quad \boxed{f\ \bar{t}_n \to s \Leftarrow \Pi} \quad s\ \bar{a}_k \to t \Leftarrow \Pi \qquad \dfrac{P \ \square \ \Delta \Leftarrow \Pi \qquad r \to s \Leftarrow \Pi}{}}{f\ \bar{e}_n \bar{a}_k \to t \Leftarrow \Pi}$$

si $f \in DF^n$ $(k > 0)$, $(f\,\bar{t}_n \to r \Leftarrow P \,\square\, \Delta) \in [\mathcal{P}]_\perp \equiv \{R\theta \mid R \in \mathcal{P}, \theta \in Sub_{\mathcal{D}}\}$ y $s \in Pat_{\mathcal{D}}$.

La regla $\mathbf{DF}_{\mathcal{P}}$ se presenta ahora como la aplicación consecutiva de dos pasos de inferencia, denominados, respectivamente, $\mathbf{AR}_f$ y $\mathbf{FA}_f$, los cuales no pueden ser aplicados de forma separada. Como ya dijimos en el Capítulo 3, tomamos el convenio de que las premisas $P \ \square \ \Delta \Leftarrow \Pi$ deben ser entendidas como una abreviatura para

varias premisas $\alpha \Leftarrow \Pi$, una para cada producción en P y para cada restricción en Δ. El propósito de esta inferencia compuesta es el de introducir c-hechos de la forma $f\,\bar{t}_n \rightarrow t \;\Leftarrow \Pi$ como conclusión de la regla de inferencia $\mathbf{FA}_f$, denominados *c-hechos con caja* en lo sucesivo. Como veremos, sólo los c-hechos con caja y el c-hecho raíz son los que aparecerán como nodos de CTs obtenidos a partir de $CPPC(\mathcal{D})$-pruebas. Por tanto, todas las preguntas que se realicen durante una sesión de depuración declarativa serán sobre la validez o no de c-hechos en el modelo pretendido del programa, el cual, como ya hemos visto, está en sí mismo representado como un conjunto de c-hechos.

5.4.1. Árboles de prueba positivos

Cualquier $CPPC(\mathcal{D})$-derivación $\mathcal{P} \vdash_{CPPC(\mathcal{D})} G_0 \Leftarrow S$ puede ser representada en la forma de un *árbol de prueba positivo PPT* sobre el dominio $\mathcal{D}$ (del inglés, *Positive Proof Tree*), con $G \Leftarrow S$ en la raíz y c-sentencias en los nodos internos, y tal que la c-sentencia en cualquier nodo se infiere a partir de las c-sentencias de sus hijos usando alguna regla de inferencia de $CPPC(\mathcal{D})$. En particular, la sentencia en la raíz debe ser inferida usando la regla **EX**, la cual no se vuelve a aplicar en ninguna otra parte del árbol de prueba.

Como ejemplo particular, la Figura 5.2 muestra un PPT que representa gráficamente la siguiente $CPPC(\mathcal{R})$-derivación basada en el $CFLP(\mathcal{R})$-programa del Ejemplo 18:

$$\begin{aligned}\mathcal{P} \vdash_{CPPC(\mathcal{R})} \mathit{intersect}\ (\mathit{head}\ (\mathit{ladder}\ (20,20)\ 50\ 20))& \\ (\mathit{head}\ (\mathit{ladder}\ (5,5)\ 30\ 40))\ (X,Y) == R \Leftarrow& \\ X \leq 35 \wedge X \geq 20 \wedge Y \leq 5\ \square\ \{R \rightarrow \mathit{true}\}&\end{aligned}$$

Esta derivación sirve de testigo a la respuesta computada en ese ejemplo. Como sabemos, esta respuesta es incorrecta con respecto al modelo pretendido del programa. Otro ejemplo alternativo en el que pueden observarse algunas otras particularidades del cálculo de prueba positivo es el que se muestra en la Figura 5.3. En esta figura se muestra un PPT para la $CPPC(\mathcal{R})$-derivación:

$$\begin{aligned}\mathcal{P} \vdash_{CPPC(\mathcal{R})} \mathit{check}\ (\mathit{from}\ A) == B \Leftarrow& \\ \exists C.\ (A > 0 \wedge A - 1 \rightarrow! \ C \wedge C > 0 \wedge A \not\models C\ \square\ \{B \mapsto 3\})&\end{aligned}$$

Esta derivación sirve de testigo para la respuesta computada S del Ejemplo 20, que también es una respuesta incorrecta (con sus correspondientes restricciones de variables intermedias tal y como ésta es calculada por un sistema de resolución de objetivos).

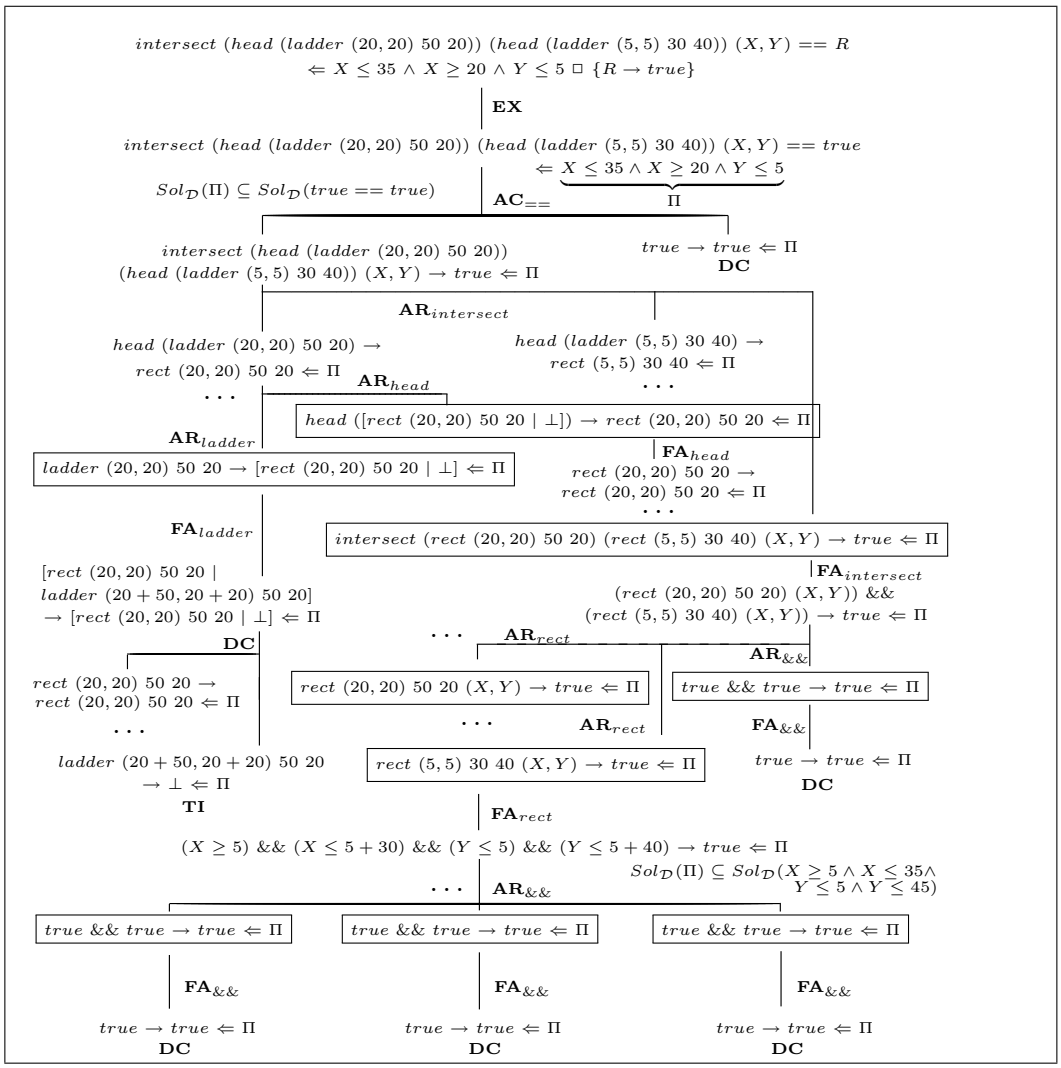


Figura 5.2: Un árbol de prueba positivo en $CPPC(\mathcal{R})$

Como preparación para los resultados de la Sección 5.5, observamos finalmente que el cálculo $CPPC(\mathcal{D})$ es semánticamente correcto en el siguiente sentido:

Teorema 11 (Corrección Semántica del Cálculo $CPPC(\mathcal{D})$) *Si G es un objetivo admisible inicial para un $CFLP(\mathcal{D})$-programa $\mathcal{P}$ y S es un objetivo en forma resuelta tal que $\mathcal{P} \vdash_{CPPC(\mathcal{D})} G \Leftarrow S$ entonces $\mathcal{P} \models_{\mathcal{D}} G \Leftarrow S$ (es decir, $Sol_{\mathcal{D}}(S) \subseteq Sol_{\mathcal{I}}(G)$ para todo $\mathcal{I} \models_{\mathcal{D}} \mathcal{P}$).*

Aunque este resultado puede verse en realidad como un corolario del apartado (1) del Teorema 5 enunciado en el Capítulo 3, detallamos a continuación su demostración.

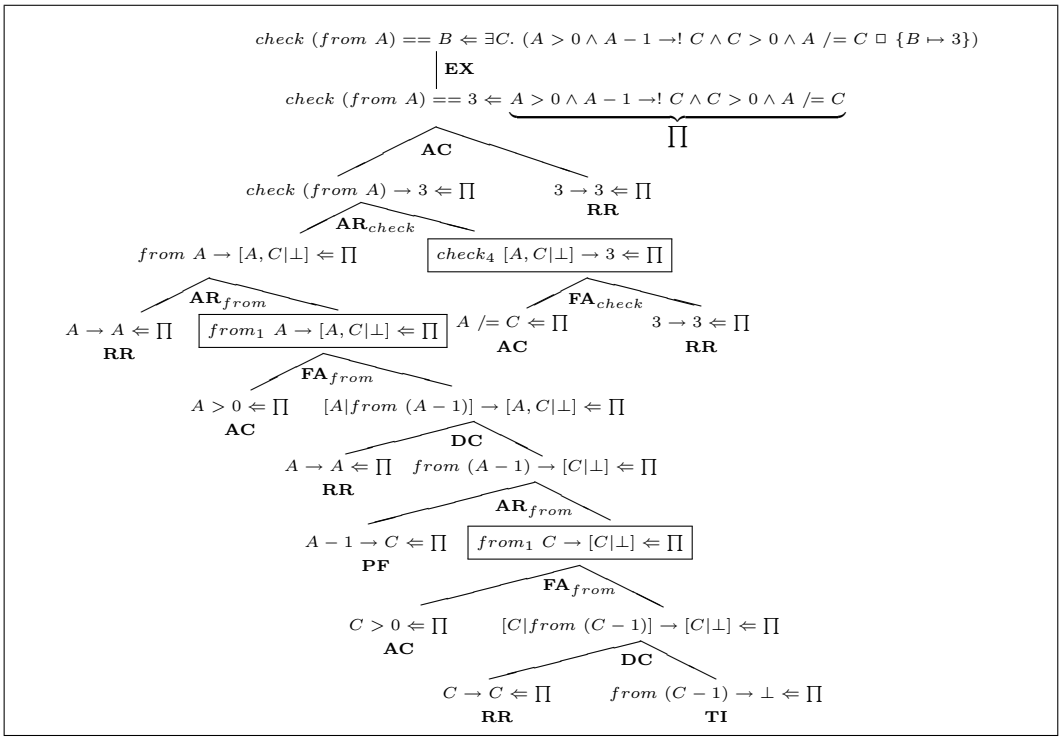


Figura 5.3: Otro árbol de prueba positivo en $CPPC(\mathcal{R})$

Demostración 16 *La demostración del teorema se basa en la corrección semántica del cálculo CRWL($\mathcal{D}$) que hemos presentado en el Capítulo 3 y que se demuestra detalladamente por inducción en el Apéndice A. Tan solo necesitamos probar que las nuevas reglas* **EX**, $\mathbf{AR}_f$ *y* $\mathbf{FA}_f$ *introducidas en el cálculo CPPC($\mathcal{D}$) son semánticamente correctas con respecto a la semántica declarativa introducida también en el Capítulo 3:*

- *La regla* **EX** *es semánticamente correcta. Supongamos por* hipótesis de inducción *que* $\mathcal{P} \models_{\mathcal{D}} G_0\sigma \Leftarrow \Pi$ *y probemos que también* $\mathcal{P} \models_{\mathcal{D}} G_0 \Leftarrow \exists\overline{U}.\ (\Pi \ \square \ \sigma)$. *Sea* $\mathcal{I}$ *un modelo arbitrario de* $\mathcal{P}$ *tal que* $\mathcal{I} \models_{\mathcal{D}} G_0\sigma \Leftarrow \Pi$, *es decir,* $Sol_{\mathcal{D}}(\Pi) \subseteq Sol_{\mathcal{I}}(G_0\sigma)$. *Probamos que* $\mathcal{I} \models_{\mathcal{D}} G_0 \Leftarrow \exists\overline{U}.\ (\Pi \ \square \ \sigma)$, *es decir,* $Sol_{\mathcal{D}}(\exists\overline{U}.\ (\Pi \ \square \ \sigma)) \subseteq Sol_{\mathcal{I}}(G_0)$. *Sea* $\eta \in Sol_{\mathcal{D}}(\exists\overline{U}.\ (\Pi \ \square \ \sigma))$. *Aplicando la Definición 11, ha de existir* $\eta' \in Val_{\mathcal{D}}$ *tal que* $\eta' =_{\setminus\overline{U}} \eta$, $\eta' \in Sol_{\mathcal{D}}(\Pi)$ *y* $\eta' \in Sol(\sigma)$. *Puesto que por* hipótesis de inducción $Sol_{\mathcal{D}}(\Pi) \subseteq Sol_{\mathcal{I}}(G_0\sigma)$, *se sigue que* $\eta' \in Sol_{\mathcal{I}}(G_0\sigma)$. *Más aún, como* $\eta' \in Sol(\sigma)$, *usando el Lema 8 (Lema de Sustitución) obtenemos que* $\eta' \in Sol_{\mathcal{I}}(G_0)$. *En consecuencia, ha de existir* $\eta' \in Val_{\mathcal{D}}$ *tal que* $\eta' =_{\setminus\overline{U}} \eta$ *y* $\eta' \in Sol_{\mathcal{I}}(G_0)$. *Finalmente, usando la condición de aplicabilidad* $fvar(G_0) \cap \overline{U} = \emptyset$ *asociada a la regla* **EX** *y el Lema 9 (Lema de Coincidencia), podemos concluir que* $\eta \in Sol_{\mathcal{I}}(G_0)$.

- *La regla* $\mathbf{AR}_f$ *es semánticamente correcta. Supongamos por* hipótesis de inducción *que* $\mathcal{P} \models_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$ *para cada* $1 \leq i \leq n$, $\mathcal{P} \models_{\mathcal{D}} f\bar{t}_n \to s \Leftarrow \Pi$, $\mathcal{P} \models_{\mathcal{D}} s\bar{a}_k \to t \Leftarrow \Pi$ *y probamos que también* $\mathcal{P} \models_{\mathcal{D}} f\bar{e}_n\bar{a}_k \to t \Leftarrow \Pi$. *Sea* $\mathcal{I}$ *un modelo arbitrario de* $\mathcal{P}$ *tal que* $\mathcal{I} \models_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$ *para cada* $1 \leq i \leq n$ *(es decir,* $Sol_{\mathcal{D}}(\Pi) \subseteq Sol_{\mathcal{I}}(e_i \to t_i)$ *para todo* $1 \leq i \leq n$*),* $\mathcal{I} \models_{\mathcal{D}} f\bar{t}_n \to s \Leftarrow \Pi$ *(es decir,* $Sol_{\mathcal{D}}(\Pi) \subseteq Sol_{\mathcal{I}}(f\bar{t}_n \to s)$*) y* $\mathcal{I} \models_{\mathcal{D}} s\bar{a}_k \to s \Leftarrow \Pi$ *(es decir,* $Sol_{\mathcal{D}}(\Pi) \subseteq Sol_{\mathcal{I}}(s\bar{a}_k \to t)$*). Probamos que* $\mathcal{I} \models_{\mathcal{D}} f\bar{e}_n\bar{a}_k \to t \Leftarrow \Pi$*, es decir,* $Sol_{\mathcal{D}}(\Pi) \subseteq Sol_{\mathcal{I}}(f\bar{e}_n\bar{a}_k \to t)$*. Sea* $\eta \in Sol_{\mathcal{D}}(\Pi)$*. Tenemos entonces que* $\eta \in Sol_{\mathcal{I}}(e_i \to t_i)$ *para cada* $1 \leq i \leq n$*, y por la Definición 11,* $\mathcal{I} \Vdash_{\mathcal{D}} e_i\eta \to t_i\eta$ *para cada* $1 \leq i \leq n$*. Análogamente,* $\eta \in Sol_{\mathcal{I}}(f\bar{t}_n \to s)$*, por la Definición 11,* $\mathcal{I} \Vdash_{\mathcal{D}} f\bar{t}_n\eta \to s\eta$*, y por la* Propiedad de Conservación *del Lema 5,* $(f\bar{t}_n\eta \to s\eta) \in \mathcal{I}$*. Del mismo modo,* $\eta \in Sol_{\mathcal{I}}(s\bar{a}_k \to t)$ *y por la Definición 11,* $\mathcal{I} \Vdash_{\mathcal{D}} (s\eta)(\bar{a}_k\eta) \to t\eta$*. Pero entonces, mediante la aplicación de la regla* $\mathbf{DF}_{\mathcal{I}}$*, tenemos que* $\mathcal{I} \Vdash_{\mathcal{D}} f(\bar{e}_n\eta)(\bar{a}_k\eta) \to t\eta$*. A partir de la Definición 11, obtenemos finalmente que* $\eta \in Sol_{\mathcal{I}}(f\bar{e}_n\bar{a}_k \to t)$.

- *La regla* $\mathbf{FA}_f$ *es semánticamente correcta. Supongamos por* hipótesis de inducción *que* $\mathcal{P} \models_{\mathcal{D}} P \;\square\; \Delta \Leftarrow \Pi$, $\mathcal{P} \models_{\mathcal{D}} r \to s \Leftarrow \Pi$ *y probemos que* $\mathcal{P} \models_{\mathcal{D}} f\bar{t}_n \to s \Leftarrow \Pi$*, donde* $(f\bar{t}_n \to r \Leftarrow P \;\square\; \Delta) \in [\mathcal{P}]_\perp$*. Por la definición de* $[\mathcal{P}]_\perp$*, existen* $(f\overline{t'}_n \to r' \Leftarrow P' \;\square\; \Delta') \in \mathcal{P}$ *y* $\theta \in Sub_{\mathcal{D}}$ *tales que* $(f\overline{t'}_n \to r' \Leftarrow P' \;\square\; \Delta')\theta \equiv (f\bar{t}_n \to r \Leftarrow P \;\square\; \Delta)$*. Sea* $\mathcal{I}$ *un modelo arbitrario de* $\mathcal{P}$ *para el que* $\mathcal{I} \models_{\mathcal{D}} P \;\square\; \Delta \Leftarrow \Pi$ *(es decir,* $Sol_{\mathcal{D}}(\Pi) \subseteq Sol_{\mathcal{I}}(P \;\square\; \Delta)$*) y* $\mathcal{P} \models_{\mathcal{D}} r \to s \Leftarrow \Pi$ *(es decir,* $Sol_{\mathcal{D}}(\Pi) \subseteq Sol_{\mathcal{I}}(r \to s)$*). Probamos que* $\mathcal{I} \models_{\mathcal{D}} f\bar{t}_n \to s \Leftarrow \Pi$*, es decir,* $Sol_{\mathcal{D}}(\Pi) \subseteq Sol_{\mathcal{I}}(f\bar{t}_n \to s)$*. Consideramos* $\eta \in Sol_{\mathcal{D}}(\Pi)$*. Entonces, tenemos que* $\eta \in Sol_{\mathcal{I}}(P \;\square\; \Delta)$*, y por la Definición 11,* $\mathcal{I} \Vdash_{\mathcal{D}} (P \;\square\; \Delta)\eta$*, y también que* $\mathcal{I} \Vdash_{\mathcal{D}} (P' \;\square\; \Delta')\theta\eta$*. Análogamente,* $\eta \in Sol_{\mathcal{I}}(r \to s)$*, y por la Definición 11,* $\mathcal{I} \Vdash_{\mathcal{D}} r\eta \to s\eta$*, y también,* $\mathcal{I} \Vdash_{\mathcal{D}} r'\theta\eta \to s\eta$*. Tenemos entonces que* $(f\overline{t'}_n \to r' \Leftarrow P' \;\square\; \Delta') \in \mathcal{P}$*,* $\theta\eta \in Sub_{\mathcal{D}}$ *son sustituciones cerradas y* $s\eta \in Pat_{\mathcal{D}}$ *es un patrón cerrado tales que* $(f\overline{t'}_n \to r' \Leftarrow P' \;\square\; \Delta')\theta\eta \equiv (f\bar{t}_n \to r \Leftarrow P \;\square\; \Delta)\eta$ *es también cerrado,* $\mathcal{I} \Vdash_{\mathcal{D}} (P' \;\square\; \Delta')\theta\eta$ *e* $\mathcal{I} \Vdash_{\mathcal{D}} r'\theta\eta \to s\eta$*. Puesto que* $\mathcal{I}$ *es un modelo de* $\mathcal{P}$*, mediante la aplicación del apartado* (2) *de la Definición 13, se obtiene que* $((f\overline{t'}_n)\theta\eta \to s\eta) \in \mathcal{I}$*, es decir,* $((f\bar{t}_n)\eta \to s\eta) \in \mathcal{I}$*, o también* $(f\bar{t}_n \to s)\eta \in \mathcal{I}$*. Finalmente, mediante la aplicación de la* Propiedad de Conservación *del Lema 5, esto es equivalente a* $\mathcal{I} \Vdash_{\mathcal{D}} (f\bar{t}_n \to s)\eta$*, y por la Definición 11 podemos concluir que* $\eta \in Sol_{\mathcal{I}}(f\bar{t}_n \to s)$.

□

5.4.2. Cálculos de resolución de objetivos $CPPC(\mathcal{D})$-admisibles

Diremos que un sistema de resolución de objetivos en $CFLP(\mathcal{D})$ es $CPPC(\mathcal{D})$-*admisible* si para cualquier respuesta computada S que haya sido obtenida para un objetivo inicial G_0 usando un programa $\mathcal{P}$ hay alguna $CPPC(\mathcal{D})$-prueba que sirva

de testigo de la derivación $\mathcal{P} \vdash_{CPPC(\mathcal{D})} G_0 \Leftarrow S$. Esta noción tiene interés debido a que pretendemos extraer los CTs necesarios para la diagnosis declarativa a partir de PPTs construidos con pasos de inferencia del cálculo $CPPC(\mathcal{D})$. El siguiente resultado muestra que existen sistemas de resolución de objetivos $CPPC(\mathcal{D})$-admisibles:

Teorema 12 (Sistemas de Resolución de Objetivos $CPPC(\mathcal{D})$-admisibles) *Los cálculos de resolución de objetivos $CLNC(\mathcal{D})$ y $CDNC(\mathcal{D})$ presentados en el Capítulo 4 son $CPPC(\mathcal{D})$-admisibles.*

Demostración 17 *La demostración se obtiene directamente por los teoremas de corrección de los cálculos $CLNC(\mathcal{D})$ y $CDNC(\mathcal{D})$ presentados en el Capítulo 4 con respecto a la lógica para la reescritura con restricciones $CRWL(\mathcal{D})$. Basta tener en cuenta que la deducción en $CPPC(\mathcal{D})$ comienza con la aplicación de la regla* **EX** *para eliminar las posibles variables existenciales de la respuesta computada. A continuación se aplican las reglas del cálculo $CRWL(\mathcal{D})$, que son las mismas que las del cálculo $CPPC(\mathcal{D})$, con la única diferencia de que ahora indicamos explícitamente el c-hecho utilizado en la aplicación de la regla $\mathbf{DF}_{\mathcal{P}}$, que antes quedaba oculto porque no era necesario.*

□

A la vista de este resultado, resulta razonable asumir también la $CPPC(\mathcal{D})$-admisibilidad para aquellos sistemas de resolución de objetivos que han sido implementados, como es el caso de *Curry* [Han06] y de $\mathcal{TOY}$ [ALR07], y cuyos modelos de cómputo están basados en estrechamiento perezoso con restricciones.

5.5. Depuración declarativa de respuestas incorrectas mediante árboles de prueba positivos abreviados

Estamos ya en condiciones de presentar el método de diagnosis declarativa de respuestas incorrectas en el esquema $CFLP(\mathcal{D})$ y de probar su corrección. Los resultados obtenidos pueden ser aplicados a cualquier sistema de resolución de objetivos que sea $CPPC(\mathcal{D})$-admisible. En primer lugar, probamos que la observación de un síntoma de error implica la existencia de algún error en el programa:

Teorema 13 (Síntomas de Error) *Asumamos que un sistema de resolución de objetivos $CPPC(\mathcal{D})$-admisible computa S como respuesta para el objetivo inicial G_0 usando el programa $\mathcal{P}$. Si S es incorrecta con respecto a la interpretación pretendida $\mathcal{I}$ del usuario entonces alguna regla de programa perteneciente a $\mathcal{P}$ es incorrecta con respecto a $\mathcal{I}$.*

Demostración 18 *Debido a la CPPC($\mathcal{D}$)-admisibilidad del sistema de resolución de objetivos sabemos que* $\mathcal{P} \vdash_{CPPC(\mathcal{D})} G_0 \Leftarrow S$. *Entonces, a partir del Teorema 11 se tiene que* $\mathcal{P} \models_{\mathcal{D}} G_0 \Leftarrow S$, *es decir, que* $Sol_{\mathcal{D}}(S) \subseteq Sol_{\mathcal{J}}(G_0)$ *para cada* $\mathcal{J} \models_{\mathcal{D}} \mathcal{P}$. *Puesto que* S *es incorrecto con respecto a la interpretación pretendida del usuario* $\mathcal{I}$, *ha de darse el caso de que* $Sol_{\mathcal{D}}(S) \not\subseteq Sol_{\mathcal{I}}(G_0)$, *debido al apartado* (1) *de la Definición 29. Por tanto, podemos concluir que la interpretación pretendida* $\mathcal{I}$ *no es un modelo de* $\mathcal{P}$. *Entonces, por el apartado* (2) *de la Definición 29, alguna regla de programa* ($f\bar{t}_n \rightarrow t \Leftarrow P \ \square \ \Delta$) *perteneciente a* $\mathcal{P}$ *no es válida en* $\mathcal{I}$.

□

El teorema anterior no proporciona un método práctico para poder encontrar una regla de programa errónea. Como se explicó al comienzo de este capítulo, un método de diagnosis declarativo debería tratar de encontrar la regla de programa errónea inspeccionando un CT. Con este fin, en esta sección vamos a proponer el uso de árboles de prueba $CPPC(\mathcal{D})$ abreviados como CTs. Puesto que la regla $\mathbf{DF}_{\mathcal{P}}$ es la única regla de inferencia en el cálculo $CPPC(\mathcal{D})$ que realmente depende del programa, los árboles de prueba abreviados omitirán los pasos de inferencia que estén relacionados con cualquier otra regla del cálculo $CPPC(\mathcal{D})$. De manera más precisa, dado un PPT $\mathcal{T}$, su *árbol de prueba positivo abreviado* $APPT$ sobre $\mathcal{D}$ (del inglés, *Abbreviated Positive Proof Tree*), queda definido como sigue:

- La raíz del $\mathcal{AT}$ es la raíz de $\mathcal{T}$.
- Los hijos de un nodo N en $\mathcal{AT}$ son los descendientes más próximos de N en $\mathcal{T}$, los cuales se corresponden con c-hechos con caja introducidos por pasos $\mathbf{DF}_{\mathcal{P}}$ de inferencia.

Un nodo en un $APPT$ se denomina *nodo crítico* si y sólo si la c-sentencia en el nodo no es válida en la interpretación pretendida $\mathcal{I}$ del programa, mientras que todas las c-sentencias en los nodos hijos son válidas en $\mathcal{I}$.

La Figura 5.4 muestra el $APPT$ asociado al PPT de la Figura 5.3. El único nodo incorrecto es el que se muestra rodeado por una caja doble, revelando así que la forma errónea de la última regla de programa para *check*, es decir, *check* $[X, Y|Zs] \rightarrow 3 \Leftarrow X \not= Y$ (véase el Ejemplo 20) es incorrecta con respecto a la interpretación pretendida del usuario.

Nuestro último teorema garantiza que la diagnosis declarativa con $APPT$s usada como CTs conduce a la detección correcta de errores de programa.

Teorema 14 (Diagnosis Declarativa de Respuestas Incorrectas) *Bajo las suposiciones del Teorema 13, cualquier APPT que sirva de testigo para* $\mathcal{P} \vdash_{CPPC(\mathcal{D})} G_0 \Leftarrow S$ *(el cual debe de existir debido a la CPPC($\mathcal{D}$)-admisibilidad del sistema de*

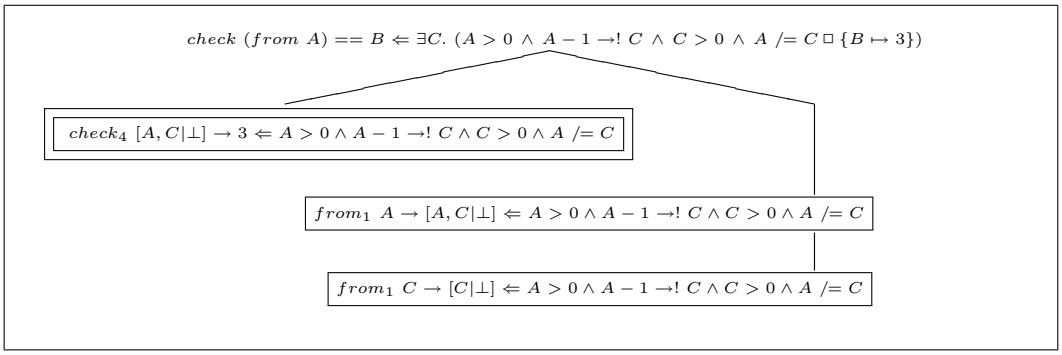


Figura 5.4: Un árbol de prueba positivo abreviado en $CPPC(\mathcal{R})$

resolución de objetivos) tiene algún nodo crítico. Más aún, cada nodo crítico apunta a una regla de programa perteneciente a $\mathcal{P}$ que es incorrecta en la interpretación pretendida del usuario.

Demostración 19 *Sea $\mathcal{T}$ un PPT que sirve de testigo a la deducción $\mathcal{P} \vdash_{CPPC(\mathcal{D})} G_0 \Leftarrow S$ y sea $\mathcal{AT}$ el correspondiente APPT. Gracias a la* **completitud débil de la depuración declarativa** *(véase [Nai97]), $\mathcal{AT}$ tiene algún nodo crítico. Demostramos en primer lugar que cualquier nodo crítico N de $\mathcal{AT}$ se corresponde con un* nodo con caja *introducido en $\mathcal{T}$ mediante una aplicación de la $CPPC(\mathcal{D})$-inferencia $\mathbf{DF}_{\mathcal{P}}$, el cual es también un nodo crítico en $\mathcal{T}$. Sea N cualquier nodo crítico de $\mathcal{AT}$ y $N'_1, \ldots, N'_p$ los hijos de N en $\mathcal{AT}$, y sean φ, φ'_j $(1 \leq j \leq p)$ las c-sentencias que etiquetan a N y a los N'_j. Como N es un nodo crítico, todas las φ'_j son válidas en $\mathcal{I}$ mientras que φ es inválida en $\mathcal{I}$. Debido a la construcción de $\mathcal{AT}$ a partir de $\mathcal{T}$, hay dos posibles casos para la relación entre N y los N'_j en $\mathcal{T}$:*

- Caso 1*: N es la raíz de $\mathcal{T}$ y N'_j son los descendientes independientes más cercanos de N en $\mathcal{T}$ que son nodos con caja.*
- Caso 2*: N es un nodo con caja de $\mathcal{T}$ y N'_j son los descendientes independientes más cercanos de N en $\mathcal{T}$ que son nodos con caja.*

En el Caso 1*, todos los pasos de inferencia que van desde $\varphi'_1, \ldots, \varphi'_m$ a φ en $\mathcal{T}$ podrían usar una $CPPC(\mathcal{D})$-inferencia distinta de $\mathbf{DF}_{\mathcal{P}}$. Como se ha mostrado en el Teorema 11, todas las reglas de inferencia en $CPPC(\mathcal{D})$, con la sola excepción de $\mathbf{DF}_{\mathcal{P}}$, preservan la validez en interpretaciones arbitrarias. Como todas las φ'_j son válidas en $\mathcal{I}$ pero φ no lo es, este caso es imposible. En el* Caso 2*, cada uno de los hijos N_i $(1 \leq i \leq m)$ de N en $\mathcal{T}$ está etiquetado mediante alguna c-sentencia φ_i, la cual se sigue de $\varphi'_1, \ldots, \varphi'_j$ por medio de $CPPC(\mathcal{D})$-inferencias distintas de $\mathbf{DF}_{\mathcal{P}}$, preservando así la validez en interpretaciones arbitrarias. En consecuencia, φ_i es válido en $\mathcal{I}$ para todo $1 \leq i \leq m$ y N es un nodo crítico en $\mathcal{T}$.*

Examinando ahora el nodo con caja N en $\mathcal{T}$ encontramos que:

(1) N está etiquetado por un c-hecho $f\bar{t}_n \to s \Leftarrow \Pi$, que es inválido en $\mathcal{I}$. Por tanto, $\mathcal{I} \not\models_{\mathcal{D}} f\bar{t}_n \to s \Leftarrow \Pi$ y entonces $Sol_{\mathcal{D}}(\Pi) \not\subseteq Sol_{\mathcal{I}}(f\bar{t}_n \to s)$.

(2) N tiene hijos etiquetados por $P \square \Delta \Leftarrow \Pi$ que son válidos en $\mathcal{I}$. En consecuencia, $\mathcal{I} \models_{\mathcal{D}} P \square \Delta \Leftarrow \Pi$ y entonces $Sol_{\mathcal{D}}(\Pi) \subseteq Sol_{\mathcal{I}}(P \square \Delta)$.

(3) N tiene un hijo etiquetado mediante $r \to s \Leftarrow \Pi$ que es válido en $\mathcal{I}$. Por tanto, $\mathcal{I} \models_{\mathcal{D}} r \to s \Leftarrow \Pi$ y entonces $Sol_{\mathcal{D}}(\Pi) \subseteq Sol_{\mathcal{I}}(r \to s)$.

(4) $(f\bar{t}_n \to r \Leftarrow P \square \Delta) \in [\mathcal{P}]_\perp$. Por la definición de $[\mathcal{P}_\perp]$, ha de existir una regla de programa con restricciones $(f\overline{t'}_n \to r' \Leftarrow P' \square \Delta') \in \mathcal{P}$ y una sustitución $\theta \in Sub_{\mathcal{D}}$ tal que $(f\overline{t'}_n \to r' \Leftarrow P' \square \Delta')\theta \equiv (f\bar{t}_n \to r \Leftarrow P \square \Delta) \in [\mathcal{P}]_\perp$.

Sea $\eta \in Sol_{\mathcal{D}}(\Pi)$ una solución de Π elegida arbitrariamente. Entonces, de acuerdo con los apartados (2) y (3), $\eta \in Sol_{\mathcal{I}}(P \square \Delta)$ (o equivalentemente, por el apartado (4), $\theta\eta \in Sol_{\mathcal{I}}(P' \square \Delta')$) y $\eta \in Sol_{\mathcal{I}}(r \to s)$ (o equivalentemente también, por el apartado (4), $\eta \in Sol_{\mathcal{I}}(r'\theta \to s)$). Por la Definición 28, se tiene que $\mathcal{I} \Vdash_{\mathcal{D}} (P \square \Delta)\eta$ (o equivalentemente, por el apartado (4), $\mathcal{I} \Vdash_{\mathcal{D}} (P' \square \Delta')\theta\eta$) e $\mathcal{I} \Vdash_{\mathcal{D}} r\eta \to s\eta$ (o equivalentemente también, por el apartado (4), $\mathcal{I} \Vdash_{\mathcal{D}} r'\theta\eta \to s\eta$). Puesto que η es una valoración, $\theta\eta \in Sub_{\mathcal{D}}$ es una sustitución cerrada, $s\eta \in Pat_{\mathcal{D}}$ es un patrón cerrado y $(f\overline{t'}_n \to r' \Leftarrow P' \square \Delta')\theta\eta$ es cerrado. En esta situación, si suponemos que la regla de programa con restricciones $(f\overline{t'}_n \to r' \Leftarrow P' \square \Delta') \in \mathcal{P}$ es válida en $\mathcal{I}$, de acuerdo con el apartado (2) de la Definición 13 en el Capítulo 3, se obtiene que $\mathcal{I} \Vdash_{\mathcal{D}} (f\overline{t'}_n)\theta\eta \to s\eta$ (o equivalentemente, por el apartado (4) anterior, $\mathcal{I} \Vdash_{\mathcal{D}} (f\bar{t}_n)\eta \to s\eta$) y entonces $\eta \in Sol_{\mathcal{I}}(f\bar{t}_n \to s)$. Se concluye así que $Sol_{\mathcal{D}}(\Pi) \subseteq Sol_{\mathcal{I}}(f\bar{t}_n \to s)$, en contradicción con el apartado (1) *anterior. Por consiguiente, la regla de programa $(f\overline{t'}_n \to r' \Leftarrow \Delta')$ en $\mathcal{P}$ es incorrecta en $\mathcal{I}$.*

□

5.6. Una herramienta para el diagnóstico declarativo de respuestas incorrectas en $\mathcal{TOY}$

En esta sección damos una breve presentación de $\mathcal{DDT}$ [Cab05], una herramienta gráfica de depuración declarativa que proporciona un prototipo de implementación del método de diagnosis presentado en este capítulo para el dominio de restricciones $\mathcal{R}$, el cual permite manejar restricciones aritméticas sobre números reales además de restricciones de igualdad y de desigualdad, como se vio en el Capítulo 2. La herramienta $\mathcal{DDT}$ se presenta como una extensión de una herramienta previa [CR04] desarrollada por nuestro grupo de investigación que sin embargo no soportaba restricciones que no fueran de igualdad y desigualdad, y se encuentra disponible dentro de la distribución pública del sistema de programación lógico funcional $\mathcal{TOY}$, el cual puede ser descargado en `http://toy.sourceforge.net`. La principal aportación del

trabajo del doctorando en el desarrollo de esta extensión de la herramienta ha sido la de precisar la forma en que se han implementado los nuevos árboles de cómputo con restricciones mediante la utilización de la lógica de reescritura $CRWL(\mathcal{D})$, estableciendo y demostrando para ello los resultados teóricos que nos permiten justificar su elección.

En adición a la interfaz gráfica implementada en Java, $\mathcal{DDT}$ soporta dos estrategias diferentes para la navegación del CT durante una sesión de depuración. El lector interesado puede referirse a [Cab05] para una mayor explicación del uso de la herramienta, una discusión sobre la implementación y su eficiencia, y una colección de programas incorrectos (disponible en la carpeta *examples/debugger* de la distribución de $\mathcal{TOY}$) que pueden ser diagnosticadas usando $\mathcal{DDT}$.

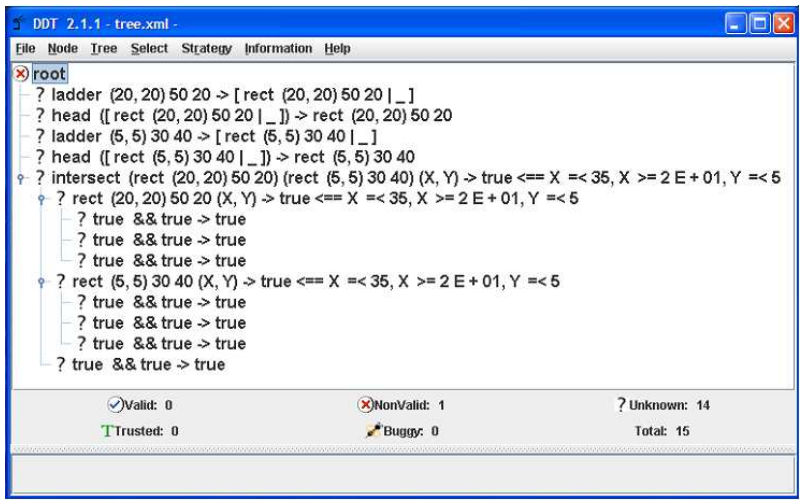
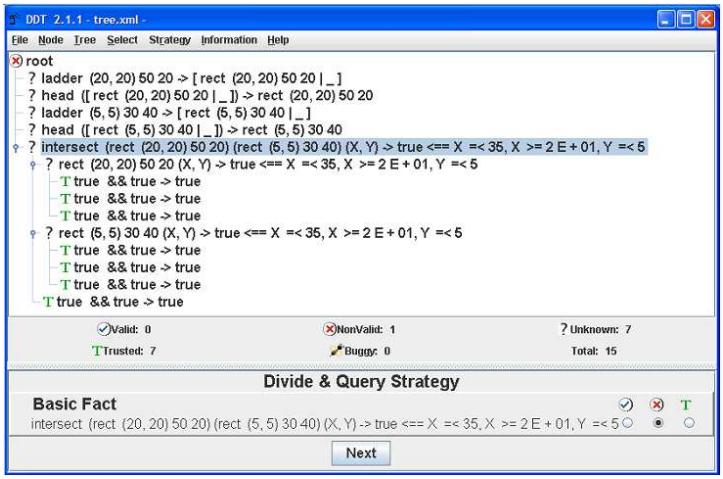


Figura 5.5: $APPT(\mathcal{R})$ correspondiente al $PPT(\mathcal{R})$ de la Figura 5.2

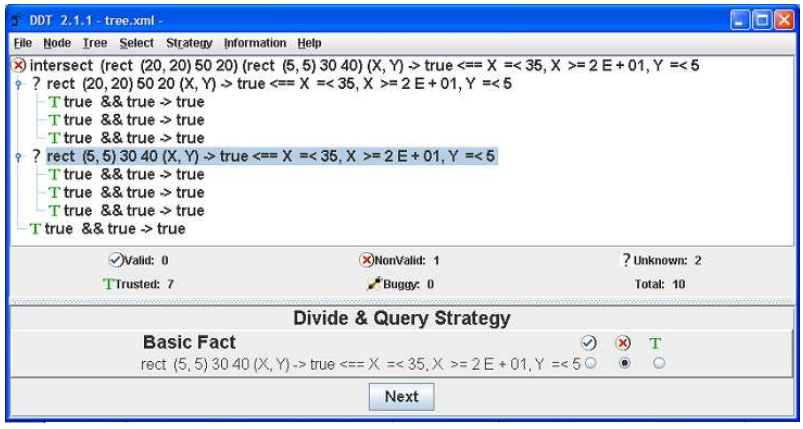
La Figura 5.5 muestra el $APPT$ asociado al PPT de la Figura 5.2 como es mostrado por la herramienta $\mathcal{DDT}$. Aunque en teoría todos los c-hechos en un PPT deberían incluir la misma conjunción de restricciones Π, en la práctica la herramienta permite simplificar Π en cada c-hecho $f\,\bar{t}_n \rightarrow t \Leftarrow \Pi$, guardando sólo aquellas restricciones atómicas que estén relacionadas con las variables que aparecen en $f\,\bar{t}_n \rightarrow t$. Puede justificarse fácilmente que esta simplificación no afecta realmente al significado pretendido de los c-hechos.

Antes de comenzar una *sesión de depuración*, el usuario puede inspeccionar y simplificar el árbol usando varias facilidades. Por ejemplo, el usuario podría marcar cualquier nodo correspondiente a la función infija && como *trusted*, indicando que la definición de && es con seguridad no errónea. Esto hace que todos los nodos correspondientes a && sean automáticamente válidos. Los nodos válidos pueden ser eliminados del árbol de manera segura (el conjunto de nodos críticos no va a cambiar)

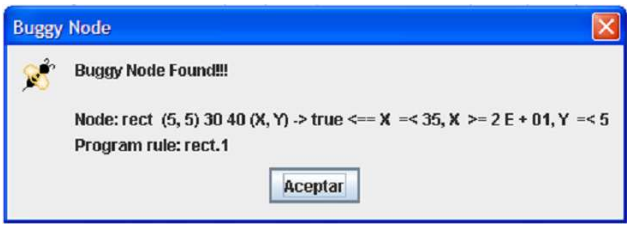
usando un menú de opciones adecuado. A continuación, el usuario puede comenzar una sesión de depuración seleccionando una de las dos posibles estrategias en $\mathcal{DDT}$: la estrategia *descendente* (*top-down*) o la estrategia *pregunta y divide* (*divide and query*) (véase [CR04] para una comparativa detallada entre ambas estrategias en una versión más antigua de $\mathcal{DDT}$ que no soportaba restricciones). Después de seleccionar la estrategia *divide and query*, la cual conduce usualmente a sesiones más cortas, $\mathcal{DDT}$ pregunta por la validez del siguiente nodo:



El modelo pretendido del programa corresponde a las intuiciones explicadas en la Sección 5.1. Por tanto, la cuestión que se plantea al usuario debe entenderse como: *¿Es* (X, Y) *un punto en la intersección de los dos rectángulos para todos los posibles valores de* X, Y *que satisfacen las restricciones* $X \leq 35, X \geq 20, Y \leq 5$*?* La respuesta es *no*, porque con estas restricciones Y puede tomar cualquier valor menor que 5 y algunos de estos valores podrían dar lugar a un par (X, Y) fuera de la intersección para cualquier X. Así, el usuario marca la cruz, dando a entender que el c-hecho por el que se le está preguntando no es válido. La siguiente cuestión es:



La cual también es calificada como no válida por el usuario. En este punto, la herramienta encuentra un nodo crítico, el cual apunta a la regla de programa que es incorrecta y finaliza la sesión de depuración.



El *APPT* asociado a una respuesta incorrecta se construye mediante una transformación de programa, utilizando técnicas similares a las presentadas en [CR04, Cab04] para el caso de programas *FLP* sin restricciones. El árbol que se obtiene se muestra entonces a través de la interfaz gráfica del depurador implementada en Java.

El lector interesado en el uso práctico de $\mathcal{DDT}$ y de las diversas estrategias utilizables para localizar nodos críticos en un *CT* dado puede consultar [CR04, Cab04, Cab05, Sil06, Sil07].

Capítulo 6

Depuración declarativa de respuestas perdidas

En este capítulo vamos a complementar el estudio de las técnicas de depuración declarativa en el esquema $CFLP(\mathcal{D})$ iniciado en el capítulo anterior considerando la depuración como el diagnóstico de las causas que dan lugar a resultados inesperados observados después de la compleción de algún cómputo. Para ello, presentamos un método declarativo de diagnosis de *respuestas computadas incompletas*, siguiendo un esquema similar al ya presentado en el capítulo previo para el diagnóstico de respuestas incorrectas. A pesar de su similitud, mientras que los métodos y herramientas considerados en el Capítulo 5 suponen un progreso incremental con respecto a resultados previamente conocidos para lenguajes *CLP* [TF00, FLT03], *FLP* [NB95, CLR01, CR02, CR04, Cab04] y *CFLP* [Cab05, CRV06a], los resultados que se proponen ahora en este capítulo son más novedosos. De hecho, no conocemos ninguna investigación relativa al diagnóstico declarativo de respuestas perdidas en lenguajes *CFLP* tan expresivos como los ofrecidos a través del esquema $CFLP(\mathcal{D})$, excepto en nuestras publicaciones [CRV07, CRV08], a pesar de que las respuestas perdidas son un problema común en la práctica (que puede surgir incluso en ausencia de respuestas incorrectas).

En la siguiente sección vamos a motivar nuestro enfoque, basado principalmente en [CRV08], y a ilustrar sus principales características a través de la presentación de varios ejemplos concretos de depuración de respuestas perdidas. En el resto del capítulo presentamos nuestro enfoque, exponiendo sus fundamentos y describiendo brevemente un prototipo de implementación.

6.1. Ejemplos motivadores

Depuración de relaciones de familia perdidas en $CFLP(\mathcal{H})$. Como primer ejemplo de depuración de respuestas perdidas vamos a considerar el dominio de restricciones de Herbrand $\mathcal{H}$, mediante el cual, como ya se vio en el Capítulo 2,

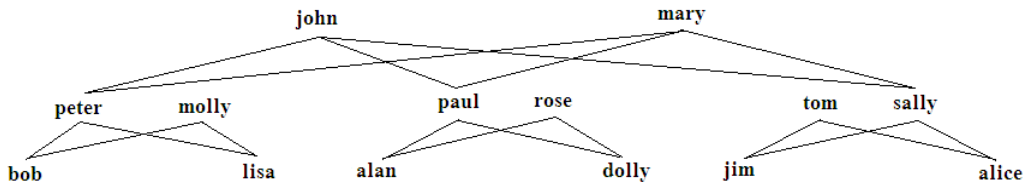


Figura 6.1: Relaciones de familia

podemos incorporar y manejar restricciones de igualdad y desigualdad simbólica sobre patrones dentro del código de los programas. El fragmento de $CFLP(\mathcal{H})$-programa escrito en sintaxis $\mathcal{TOY}$ que se muestra a continuación permite generar relaciones de familia basadas en el árbol familiar que se muestra en la Figura 6.1, el cual sirve como ilustración gráfica del significado pretendido de este programa.

```
% Tipo de datos enumerado para representar personas

data person = john | mary | peter | ... | dolly | jim  | alice

% Tipo sinónimo para representar relaciones interpersonales como
% funciones indeterministas

type relation = person -> person

% Función de composición

(.) :: (B -> C) -> (A -> B) -> (A -> C)
(F . G) X --> F (G X)

% Función auxiliar

otherThan :: A -> A -> A
otherThan X Y --> Y <== Y /= X

% Datos familiares básicos

maleChildOf :: (person,person) -> person
maleChildOf(john,mary)   --> peter
maleChildOf(john,mary)   --> paul
maleChildOf(peter,molly) --> bob
maleChildOf(paul,rose)   --> alan
maleChildOf(tom,sally)   --> jim
```

```
femaleChildOf :: (person,person) -> person
femaleChildOf(john,mary)   --> sally
femaleChildOf(peter,molly) --> lisa
femaleChildOf(paul,rose)   --> dolly
femaleChildOf(tom,sally)   --> alice

% Relaciones familiares básicas

fatherOf,motherOf,sonOf,daughterOf,brotherOf,sisterOf :: relation
fatherOf   X --> Y <== maleChildOf(Y,Z)   == X
fatherOf   X --> Y <== femaleChildOf(Y,Z) == X
motherOf   X --> Y <== maleChildOf(Z,Y)   == X
motherOf   X --> Y <== femaleChildOf(Z,Y) == X
sonOf      X --> maleChildOf(X,Y)
sonOf      X --> maleChildOf(Y,X)
daughterOf X --> femaleChildOf(X,Y)
daughterOf X --> femaleChildOf(Y,X)
brotherOf  X --> otherThan X (maleChildOf(fatherOf X,motherOf X))
sisterOf   X --> otherThan X (femaleChildOf(fatherOf X,motherOf X))

% Generación de relaciones familiares

basicFamilyRelation :: relation
basicFamilyRelation --> fatherOf
basicFamilyRelation --> motherOf
basicFamilyRelation --> sonOf
basicFamilyRelation --> daughterOf
basicFamilyRelation --> brotherOf
basicFamilyRelation --> sisterOf

familyRelation :: relation
familyRelation --> basicFamilyRelation
familyRelation --> basicFamilyRelation . basicFamilyRelation
```

Se observa, en primer lugar, que el $CFLP(\mathcal{H})$-programa anterior permite representar las relaciones familiares como funciones indeterministas, aprovechando además la capacidad del sistema $\mathcal{TOY}$ para poder representar valores de tipo funcional mediante *patrones de orden superior*. En segundo lugar, se observa que el siguiente objetivo falla inesperadamente, debido a la existencia de relaciones familiares entre `alice` y `alan`, las cuales puede ser observadas directamente a través del árbol de familia de la Figura 6.1.

```
Toy> familyRelation == R, R alice == alan
       no
       Elapsed time: 438 ms.
```

En este caso, es posible realizar diferentes *diagnósticos de respuestas perdidas*:

- La respuesta `R -> sonOf . brotherOf . motherOf` (es decir, `alan` es hijo de un hermano de la madre de `alice`) puede haberse perdido debido a una definición incompleta de `familyRelation`. Esta función podría ser extendida, en consecuencia, añadiendo la nueva regla:

```
familyRelation = familyRelation . basicFamilyRelation
```

- Otras respuestas como `R -> cousinOf` o `R -> sonOf . uncleOf` podrían haberse perdido debido a una definición incompleta de `basicFamilyRelation`. Esta función podría ser extendida del modo siguiente:

```
basicFamilyRelation = .../// cousinOf // uncleOf
```

Nos referiremos a este fragmento de programa y al objetivo previos como ejemplo recurrente en el resto de este capítulo. En este punto, una herramienta para la diagnosis declarativa debería ser capaz de encontrar un *nodo crítico* en un árbol de cómputo adecuado, con el fin de detectar alguna definición de función que sea incompleta y posiblemente responsable de las respuestas perdidas.

La aproximación que utilizaremos en este capítulo consistirá en usar CTs cuyos nodos tengan asociados lo que denominaremos como *aserciones de recolección de respuestas*, representadas abreviadamente con el nombre de *aca*s (del inglés, *answer collection assertions*).

- El *aca* en el nodo raíz va a tener la forma $G_0 \Rightarrow \bigvee_{i \in I} \hat{S}_i$, asegurando que todas las soluciones del objetivo inicial G_0 quedan cubiertas por la disyunción finita de respuestas computadas $\bigvee_{i \in I} \hat{S}_i$. Designaremos por S_i a cada objetivo en forma resuelta $\Pi \square \sigma$ que represente una de estas respuestas computadas, de forma que la notación $\hat{S}_i$ indique la cuantificación existencial explícita $\exists \overline{U}.\ (\Pi \square \sigma)$ con respecto a las variables de S_i que no aparezcan en el objetivo inicial G_0. En el caso particular en el que $I = \varnothing$, es decir, en el caso en el que la disyunción quede vacía debido a un objetivo que falla finitamente, utilizaremos el símbolo $\blacklozenge$ para representar esta disyunción.
- Los *aca*s en nodos internos tienen la forma $f\overline{t}_n \rightarrow t \square S \Rightarrow \bigvee_{i \in I} \hat{S}_i$, asegurando que $\bigvee_{i \in I} \hat{S}_i$ cubre todas las posibles soluciones para un objetivo intermedio

$f\bar{t}_n \rightarrow t \square S$, mediante el cual se pretende computar resultados de la llamada a la función $f\bar{t}_n$ que encajan el patrón t y satisfacen las restricciones de S. En tales llamadas de función, ni los parámetros $\bar{t}_n$ ni el resultado t necesitan ser totalmente evaluados; en su lugar, serán evaluados hasta alcanzar la extensión necesaria para resolver el objetivo más externo bajo una estrategia de evaluación perezosa.

Más aún, el CT total debe ser tal que la validez del *aca* en cada nodo se siga a partir de la validez de los *aca*s de sus hijos, bajo la suposición de que la definición de función que relaciona el nodo padre con los nodos hijos es completa con respecto a la semántica pretendida del programa. Conseguiremos que se satisfaga este requisito construyendo el CT como un árbol de prueba abreviado con respecto a un sistema de inferencia lógicamente correcto para derivar *aca*s.

Una vez que un CT haya sido construido, la búsqueda de un *nodo crítico* podrá ser implementada con la ayuda de un *oráculo* externo (usualmente el programador), que tenga un conocimiento declarativo de la validez de los *aca*s basado en una interpretación pretendida conocida del programa. Cualquier CT con un *aca* inválido en su raíz va a tener con seguridad al menos un nodo crítico etiquetado con un *aca* inválido y cuyos hijos están todos etiquetados con *aca*s válidos. Cada nodo crítico N estará relacionado con alguna función particular f_N cuyas reglas de programa son responsables de la computación del *aca* en N a partir de los *aca*s en los N hijos. En consecuencia, las reglas de programa para f_N serán diagnosticadas como incompletas. La detección de nodos críticos depende de las decisiones tomadas por el oráculo sobre la validez de los *aca*s con respecto a la interpretación pretendida del programa. Puesto que el oráculo es usualmente el programador, éste puede decidir experimentar con diferentes elecciones del modelo pretendido con el fin de obtener diferentes diagnósticos de funciones posiblemente incompletas.

Consideremos, por ejemplo, el CT de la Figura 6.2 que se obtendría para el ejemplo de las relaciones familiares (más adelante, en la Sección 6.4 se explicará detalladamente la construcción de este CT). Inicialmente, el programador juzgará el *aca* en la raíz de este CT como inválido, debido a que no esperaba un fallo finito para el objetivo de partida.

- A partir del conocimiento que el usuario tiene del árbol familiar en el que se basa la interpretación pretendida de este programa, podría echar en falta la respuesta `R -> sonOf . brotherOf . motherOf` (es decir, `alan` es un hijo de un hermano de la madre de `alice`). Si decide considerar entonces que el *aca* en el nodo B del árbol es válido, el nodo A podría llegar a ser un nodo crítico, y en consecuencia, la función `familyRelation` sería diagnosticada como incompleta, sugiriendo la adición de nuevas reglas de programa para

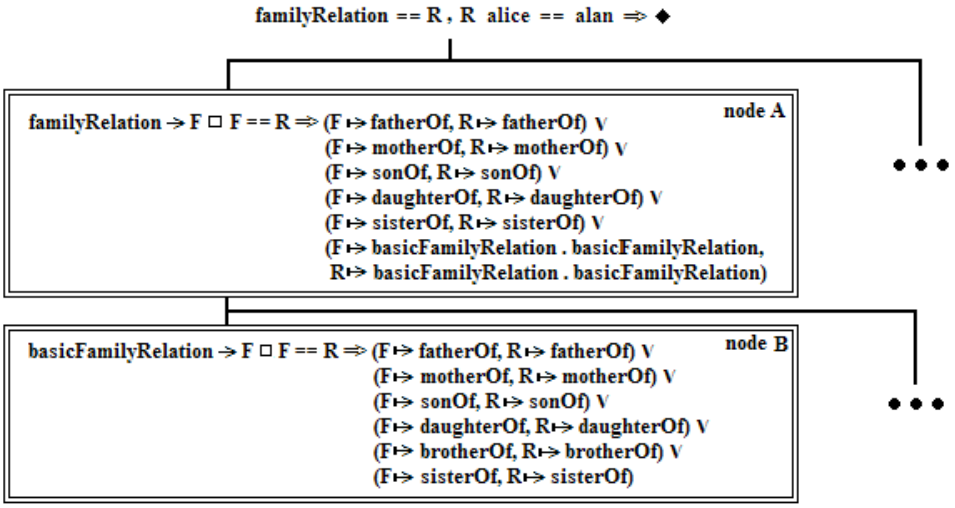


Figura 6.2: *CT* para el ejemplo de las relaciones de familia

computar relaciones de familia como composiciones de más de dos relaciones básicas de familia. Siguiendo este diagnóstico y añadiendo estas nuevas reglas, la ejecución del objetivo devuelve ya el resultado esperado:

```
Toy> familyRelation == R, R alice == alan
        { R -> sonOf . brotherOf . motherOf }
        Elapsed time: 9609 ms.

     sol.1, more solutions (y/n/d/a) [y]? y
        no
        Elapsed time: 10922 ms.
```

- Sin embargo, el programador podría también echar en falta otras respuestas tales como `R -> cousinOf` o `R -> sonOf . uncleOf`, y decidir entonces ver el *aca* en el nodo B como inválido. En este caso, el nodo B podría llegar a ser crítico, la función `basicFamilyRelation` sería diagnosticada como incompleta y el programador podría reaccionar añadiendo nuevas reglas de programa tales que `basicFamilyRelation = cousinOf` y `basicFamilyRelation = uncleOf`, junto con definiciones de programa adecuadas para `cousinOf` y `uncleOf`.

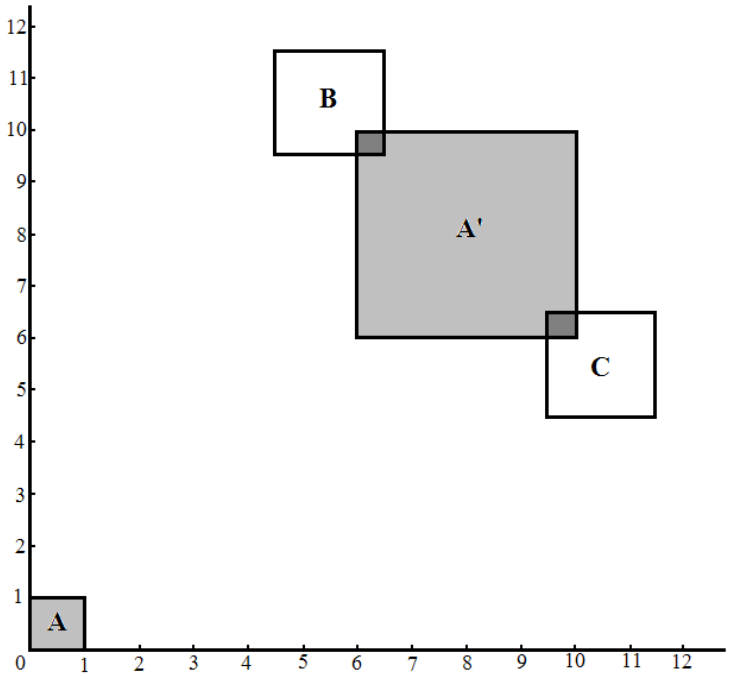


Figura 6.3: Transformaciones geométricas

Depuración de transformaciones geométricas perdidas en $CFLP(\mathcal{R})$. Examinamos ahora un segundo ejemplo de diagnosis de respuestas perdidas en el que hacemos uso del dominio de restricciones $\mathcal{R}$ presentado en el Capítulo 2. En concreto, el siguiente $CFLP(\mathcal{R})$-programa, escrito también en la sintaxis de $\mathcal{TOY}$, permite generar *transformaciones geométricas* en el plano basadas en los cuadrados A, B y C que se muestran en la Figura 6.3.

```
% Definiciones de tipos

type point  = (real,real)
type figure = point -> bool
type side   = real

% Definición de cuadrado

square :: point -> side -> figure
square (X0,Y0) S (X,Y) --> true <==
                           X0 <= X, X <= X0 + S, Y0 <= Y, Y <= Y0 + S
```

```
% Relaciones de pertenencia y de intersección

isIn :: figure -> point -> bool
isIn F P --> F P

commonPoint :: figure -> figure -> point -> bool
commonPoint F G P --> true <== isIn F P, isIn G P

% Transformaciones

type transformation = figure -> figure

simpleTrans :: transformation
simpleTrans --> translateTo NewPoint
simpleTrans --> halfSize
simpleTrans --> doubleSize

translateTo :: point -> transformation
translateTo P (square P0 S) --> square P S

halfSize :: transformation
halfSize (square P S) --> square P (S/2)

doubleSize :: transformation
doubleSize (square P S) --> square P (2*S)

transfor :: transformation
transfor --> simpleTrans
transfor --> simpleTrans . simpleTrans

% Datos del problema

sqA, sqB, sqC :: figure
sqA --> square (0,0) 1
sqB --> square (4.5,9.5) 2
sqC --> square (9.5,4.5) 2
```

Un posible usuario de este $CFLP(\mathcal{R})$-programa podría estar interesado en experimentar con diferentes posibilidades de componer las transformaciones simples `simpleTrans` dadas en el programa (es decir, traslaciones a un determinado punto del plano `translateTo P` y homotecias que duplican `doubleSize` o reducen a la mitad `halfSize` el cuadrado sobre el que se aplican). Por ejemplo, el usuario po-

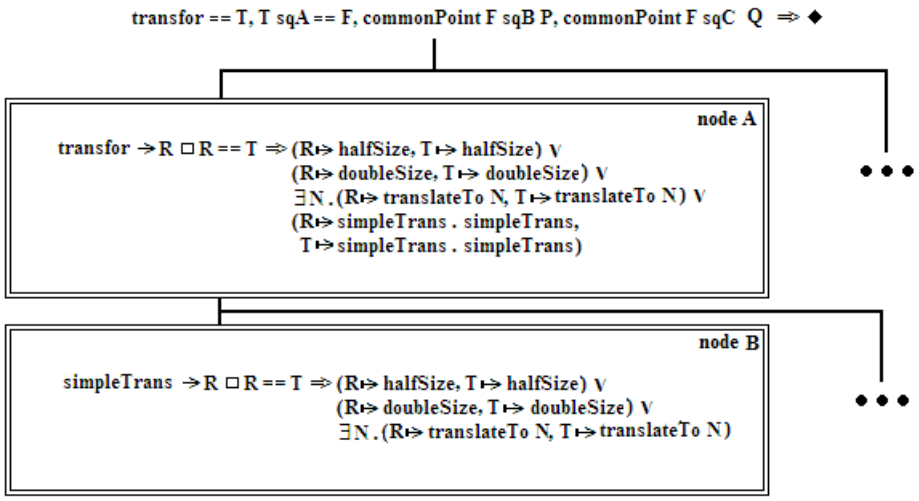


Figura 6.4: CT para el ejemplo de las transformaciones geométricas

dría estar interesado en averiguar cómo es posible transformar el cuadrado unidad A dado en la Figura 6.3 en otro cuadrado A' que interseque simultáneamente con los otros dos cuadrados B y C dados en el programa, pero de forma que se use la menor cantidad posible de transformaciones simples para conseguir este propósito. Inicialmente, el usuario puede tratar de resolver el problema usando tan solo la composición de dos transformaciones simples `simpleTrans`, con lo que puede proponer la resolución en $\mathcal{TOY}(\mathcal{R})$ del siguiente objetivo:

```
% Objetivo en Toy(R)

  Toy(R)> transfor == T, T sqA == F, commonPoint F sqB P,
          commonPoint F sqC Q

            no
            Elapsed time: 16 ms.
```

El objetivo planteado, sin embargo, falla finitamente, con lo que el usuario puede tratar de usar el diagnóstico declarativo de respuestas perdidas con el fin de detectar definiciones de funciones que posiblemente sean incompletas en su programa. En este caso, es posible argumentar de un modo análogo al expuesto en el ejemplo de las relaciones de familia, utilizando para ello el CT que se muestra en la Figura 6.4. El usuario puede tratar de completar la función `transfor` añadiendo otra transformación `simpleTrans` a la correspondiente regla de programa.

```
transfor :: transformation
transfor -> simpleTrans
transfor -> simpleTrans . simpleTrans . simpleTrans
```

Ahora el usuario obtiene la siguiente respuesta, mediante la cual aprende y complementa la parte de conocimiento que inicialmente tenía sobre la interpretación pretendida del programa:

```
Toy(R)> transfor == T, T sqA == F, commonPoint F sqB P,
        commonPoint F sqC Q

       { T -> translateTo (_A,_B) . doubleSize . doubleSize,
         F -> square (_A,_B) 4,
         P -> (_C,_D),
         Q -> (_E,_F) }

       { _C =< 6.5,
         _F =< 6.5,
         _B - _F =< -0.0,
         _D - _B =<  4.0,
         _E >= 9.5,
         _D >= 9.5,
         _A - _E >= -4.0,
         _C - _A >=  0.0 }

       Elapsed time: 15 ms.
```

Por tanto, es posible resolver el problema usando tan solo la composición de tres transformaciones simples. Si por el contrario hubiésemos utilizado, como en el ejemplo anterior de las familias, la regla más general:

```
transfor --> transfor . simpleTrans
```

entonces se producirían infinitas respuestas, como consecuencia de que se pueden usar infinitas transformaciones elementales para poder alcanzar la solución de este problema.

Depuración de respuestas perdidas y evaluación perezosa. Finalizamos esta sección considerando un tercer y último ejemplo de $CFLP(\mathcal{H})$-programa también escrito en sintaxis $\mathcal{TOY}$, mediante el cual es posible ilustrar el comportamiento del método de depuración de respuestas perdidas que proponemos en el caso de cómputos en los que intervengan funciones indeterministas y evaluación perezosa.

El siguiente programa, al que denominamos $\mathcal{P}_{\texttt{fD}}$, va a incluir reglas de programa tanto para funciones de programa indeterministas, `(//)` y `fDiff`, como para funciones deterministas, `gen` y `even`. Obsérvese que usamos la sintaxis infija para `(//)`, así como también el símbolo de igualdad `=` en lugar de la flecha de reescritura `-->` para las reglas de programa de aquellas funciones que son vistas como deterministas por el usuario. Esto es sólo a nivel de información del usuario, puesto que el sistema $\mathcal{TOY}$ no lo comprueba, tratando así a todas las funciones definidas en el programa como posiblemente indeterministas.

```
infixr 40 //    % operador de elección indeterminista

(//) :: A -> A -> A
X // _ --> X
_ // Y --> Y

fDiff :: [A] -> A
fDiff [X]      --> X
fDiff (X:Y:Zs) --> X // fDiff (Y:Zs) <== X /= Y
fDiff (X:Y:Zs) --> X                 <== X == Y

gen :: A -> A -> [A]
gen X Y = X : Y : gen Y X

even :: int -> bool
even N = true <== (mod N 2) == 0
```

La función `fDiff` permite devolver cualquier elemento que pertenezca al prefijo `Xs` de mayor longitud de la lista dada como parámetro que no incluya dos elementos idénticos en posiciones consecutivas. En general, existirá más de un elemento que satisfaga esta condición, por lo que `fDiff` es una función indeterminista. La función `gen` es, sin embargo, determinista, y devuelve una lista potencialmente infinita de la forma $[d_1, d_2, d_2, d_1, d_1, d_2, ...]$, donde los elementos d_1 y d_2 vienen dados como parámetros. Por tanto, la evaluación perezosa de `(fDiff (gen 1 2))` se espera que de lugar a dos posibles resultados, `1` y `2`, en cómputos alternativos, con lo que el objetivo inicial $G_{\texttt{fD}}$: `even (fDiff (gen 1 2)) == true` para $\mathcal{P}_{\texttt{fD}}$ se espera que tenga éxito, puesto que `(fDiff (gen 1 2))` es capaz de devolver el número par `2`.

Asumamos ahora que la tercera regla de programa para la función `fDiff` no aparece en el programa $\mathcal{P}_{\texttt{fD}}$ porque, sencillamente, nos hemos olvidado de ella. Entonces, la expresión `(fDiff (gen 1 2))` debería devolver sólo el valor numérico `1` y, por tanto, el objetivo $G_{\texttt{fD}}$ debería fallar inesperadamente. En este punto podríamos usar un diagnóstico de respuestas perdidas, con el fin de detectar un nodo crítico en el árbol de cómputo y localizar así alguna definición de función que sea incompleta (en este caso, la función `fDiff`) como causante de la pérdida de respuestas.

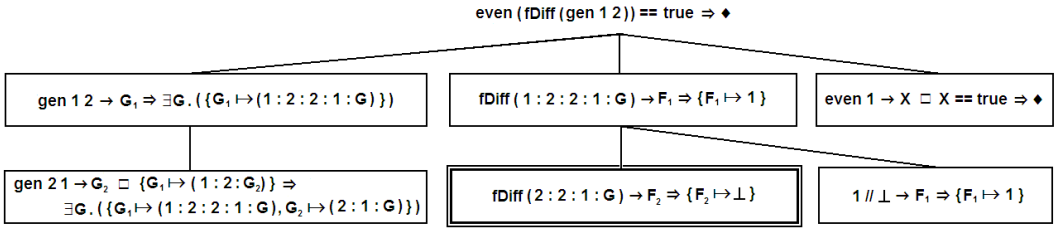


Figura 6.5: CT para la depuración de respuestas perdidas con evaluación perezosa

Un CT correspondiente al objetivo $G_{\texttt{fD}}$ y al programa $\mathcal{P}_{\texttt{fD}}$ (con la tercera regla de programa para la función `fDiff` omitida) se muestra en la Figura 6.5. Más adelante, en la Sección 6.4, se explicará detalladamente la construcción de este CT. En este caso, el programador juzgará el *aca* en la raíz del árbol correspondiente al objetivo inicial `even (fDiff (gen 1 2)) == true` $\Rightarrow \blacklozenge$ como inválido, puesto que no esperaba el fallo finito para este objetivo. Más aún, utilizando su conocimiento de la interpretación pretendida del programa, decidirá considerar los *aca*s para las funciones `gen`, `even` y `(//)` como válidas. Obsérvese que los correspondientes *aca*s del árbol `gen 1 2` $\rightarrow G_1 \Rightarrow \exists G.\ (\{G_1 \mapsto (1 : 2 : 2 : 1 : G)\})$ y `gen 2 1` $\rightarrow G_2 \ \square\ \{G_1 \mapsto (1 : 2 : G_2)\} \Rightarrow \exists G.\ (\{G_1 \mapsto (1 : 2 : 2 : 1 : G),\ G_2 \mapsto (2 : 1 : G)\})$ representan la recolección de resultados parcialmente evaluados durante el cómputo para las listas potencialmente infinitas generadas mediante `gen`. Sin embargo, el *aca* `fDiff (2:2:1:G)` $\rightarrow F_2 \Rightarrow \{F_2 \mapsto \bot\}$ asegura que el valor indefinido $\bot$ es el único resultado posible para la llamada a la función `fDiff (2:2:1:G)`, mientras que el usuario esperaba también el resultado 2. Por tanto, el usuario juzgará este *aca* como inválido. El nodo donde se sitúa (resaltado en la figura mediante un rectángulo doble) no tiene hijos, y de este modo es crítico, dando lugar como resultado del diagnóstico a que la definición de la función `fDiff` sea incompleta. Este síntoma particular de incompletitud puede ser corregido si de nuevo escribimos la tercera regla de programa para `fDiff` dentro del programa.

6.2. Teoría negativa asociada a un programa

En el resto de este capítulo supondremos *aca*s de la forma $G \Rightarrow \bigvee_{i \in I} \hat{S}_i$, donde G es un objetivo no necesariamente inicial y cada S_i es una forma resuelta. Los *aca*s que pueden aparecer en nodos de CTs tienen la forma particular ya explicada en la Sección 6.1. Nos proponemos que los CTs se puedan construir como *árboles de prueba abreviados* con respecto a un sistema de inferencia lógicamente correcto que nos permita derivar *aca*s. En concreto, vamos a presentar un sistema de inferencia cuyos *árboles de prueba* permitan representar la deducción de *aca*s a partir de la *teoría negativa* $\mathcal{P}^-$ asociada a un $CFLP(\mathcal{D})$-programa $\mathcal{P}$ dado.

Sea $\mathcal{P}$ un *CFLP*($\mathcal{D}$)-programa. La *Teoría Negativa* $\mathcal{P}^-$ asociada a $\mathcal{P}$ se obtiene en dos pasos:

1) Cada regla de programa $f\ \overline{t}_n \to r \Leftarrow P \square \Delta$ se reemplaza por una forma *estandarizada* $f\ \overline{X}_n \to Y \Leftarrow \hat{R}$, donde $\overline{X}_n, Y$ son variables nuevas, $\hat{R} = \exists \overline{U}.\ R$ con $\overline{U} = var(R) \setminus \{\overline{X}_n, Y\}$ y la condición R es $X_1 \to t_1, \ldots, X_n \to t_n, r \to Y$, $P \square \Delta$.

2) $\mathcal{P}^-$ se construye tomando un *axioma* $(f)^-_{\mathcal{P}}$ de la forma $\forall \overline{X}_n, Y.\ (f\ \overline{X}_n \to Y \Rightarrow (\bigvee_{i \in I}\ \hat{R}_i) \vee (\bot \to Y))$ para cada símbolo de función f cuyas reglas estandarizadas sean $\{f\ \overline{X}_n \to Y \Leftarrow \hat{R}_i\}_{i \in I}$. Por convenio, usaremos la notación D_f para representar la disyunción $(\bigvee_{i \in I}\ \hat{R}_i) \vee (\bot \to Y)$, dejando la cuantificación universal de las variables $\overline{X}_n, Y$ implícita.

Intuitivamente, cada axioma $(f)^-_{\mathcal{P}}$ afirma que cualquier resultado computado para f debe ser obtenido por medio de alguna de las reglas de f en el programa. La última alternativa $(\bot \to Y)$ dentro de D_f expresa que Y queda ligada al resultado indefinido $\bot$ en el caso en el que ninguna regla de programa para f tenga éxito a la hora de computar un resultado más definido.

Ejemplo 21 *Sea $\mathcal{P}$ el CFLP($\mathcal{H}$)-programa de las relaciones de familia presentado en la Sección 6.1. Entonces, $\mathcal{P}^-$ incluye (entre otros) los siguientes axiomas:*

$$
\begin{array}{l}
(femaleChildOf)^-_{\mathcal{P}} : \forall C, F.\ (femaleChildOf\ C \to F \Rightarrow \\
\quad (C \to (john, mary) \wedge sally \to F) \vee \\
\quad (C \to (peter, molly) \wedge lisa \to F) \vee \\
\quad (C \to (paul, rose) \wedge dolly \to F) \vee \\
\quad (C \to (tom, sally) \wedge alice \to F) \vee \\
\quad (\bot \to F))
\end{array}
$$

$$
\begin{array}{l}
(familyRelation)^-_{\mathcal{P}} : \forall F.\ (familyRelation \to F \Rightarrow \\
\quad (basicFamilyRelation \to F) \vee \\
\quad (basicFamilyRelation\ .\ basicFamilyRelation \to F) \vee \\
\quad (\bot \to F))
\end{array}
$$

$$
\begin{array}{l}
(basicFamilyRelation)^-_{\mathcal{P}} : \forall F.\ (basicFamilyRelation \to F \Rightarrow \\
\quad (fatherOf \to F) \vee (motherOf \to F) \vee \\
\quad (sonOf \to F) \vee (daughterOf \to F) \vee \\
\quad (brotherOf \to F) \vee (sisterOf \to F) \vee \\
\quad (\bot \to F))
\end{array}
$$

Sea ahora $\mathcal{P}_{\mathtt{fD}}$ el CFLP($\mathcal{H}$)-programa dado en la sección 6.1, con la omisión de la tercera regla de programa para fDiff. Entonces, $\mathcal{P}^-_{\mathtt{fD}}$ incluye (entre otros) el siguiente axioma para el símbolo de función fDiff:

$$\begin{aligned}(fDiff)^-_{\mathcal{P}_{\mathtt{fD}}} : \forall\, L, F.\ (fDiff\ L \to F \Rightarrow \\ \exists X.\ (L \to [X] \wedge X \to F) \vee \\ \exists X, Y, Zs.\ (L \to (X : Y : Zs) \wedge X \mathrel{/=} Y \wedge X \mathbin{//} fDiff\ (Y : Zs) \to F) \vee \\ (\bot \to F))\end{aligned}$$

Con el fin de determinar cómo se interpretan en una c-interpretación dada $\mathcal{I}$ sobre un dominio de restricciones $\mathcal{D}$ tanto la teoría negativa $\mathcal{P}^-$ asociada a un *CFLP*($\mathcal{D}$)-programa $\mathcal{P}$ como los *aca*s, terminamos esta sección precisando la semántica de modelos que vamos a utilizar. Al igual que en el capítulo anterior, vamos a optar por usar la *semántica débil* presentada en el Capítulo 3 y en [LRV07] para describir los principales resultados teóricos del método de depuración declarativa de respuestas perdidas. Así, y con el propósito de facilitar la lectura, convenimos en utilizar la notación semántica $\models_{\mathcal{D}}$ para representar realmente que $\models^w_{\mathcal{D}}$ en el sentido de la semántica débil. La siguiente definición permite entender este significado pretendido.

Definición 30 (Semántica Dependiente de una Interpretación) *Sea $\mathcal{P}$ un CFLP($\mathcal{D}$)-programa, $\mathcal{I}$ una c-interpretación sobre $\mathcal{D}$ y $G \Rightarrow \bigvee_{i\in I}\ \hat{S}_i$ un aca.*

(1) $\mathcal{I}$ es un modelo *de $\mathcal{P}^-$ (en símbolos, $\mathcal{I} \models_{\mathcal{D}} \mathcal{P}^-$) si y sólo si cualquier axioma $(f)^-_{\mathcal{P}} : (f\ \overline{X}_n \to Y \Rightarrow D_f) \in \mathcal{P}^-$ satisface $Sol_{\mathcal{I}}(f\ \overline{X}_n \to Y) \subseteq Sol_{\mathcal{I}}(D_f)$. Cuando esta inclusión se cumpla diremos que $(f)^-_{\mathcal{P}}$ es* válido *en $\mathcal{I}$, o equivalentemente también que la definición de f, tal y como está dada en $\mathcal{P}$, es* completa *con respecto a $\mathcal{I}$.*

(2) $\mathcal{I}$ es un modelo *de $G \Rightarrow \bigvee_{i\in I}\ \hat{S}_i$ (en símbolos, $\mathcal{I} \models_{\mathcal{D}} G \Rightarrow \bigvee_{i\in I}\ \hat{S}_i$) si y sólo si $Sol_{\mathcal{I}}(G) \subseteq Sol_{\mathcal{D}}(\bigvee_{i\in I}\ \hat{S}_i) = \bigcup_{i\in I}\ Sol_{\mathcal{D}}(\hat{S}_i)$.*

(3) $G \Rightarrow \bigvee_{i\in I}\ \hat{S}_i$ es consecuencia lógica *de $\mathcal{P}^-$ (en símbolos, $\mathcal{P}^- \models_{\mathcal{D}} G \Rightarrow \bigvee_{i\in I}\ \hat{S}_i$) si y sólo si cualquier modelo $\mathcal{I}$ de $\mathcal{P}^-$ es también modelo de $G \Rightarrow \bigvee_{i\in I}\ \hat{S}_i$. Cuando esto suceda diremos también que la disyunción de respuestas $\bigvee_{i\in I}\ \hat{S}_i$ es* completa *para G con respecto a $\mathcal{P}$.*

6.3. Síntomas y errores negativos

De manera similar a como ocurría en el capítulo anterior con la depuración de respuestas incorrectas, la depuración declarativa de respuestas perdidas comienza con la observación de un síntoma por parte del usuario, en este caso denominado *síntoma*

de incompletitud, y finaliza con el correspondiente *diagnóstico de incompletitud* mediante la aplicación de las técnicas de depuración explicadas en este capítulo. La descripción formal de este nuevo escenario de depuración presupone también un conocimiento previo del usuario de una interpretación pretendida del programa que se pretende depurar, entendida como una c-interpretación que permite representar el comportamiento de las funciones definidas en el programa tal y como el programador espera que sea. De forma más precisa, definimos los conceptos de síntoma y de diagnóstico apropiados para el escenario de depuración de la depuración declarativa de respuestas perdidas en el esquema $CFLP(\mathcal{D})$:

Definición 31 (Escenario de Depuración de Respuestas Perdidas) *Sea $\mathcal{I}$ la interpretación pretendida sobre $\mathcal{D}$ de un $CFLP(\mathcal{D})$-programa $\mathcal{P}$ dado. Se tiene entonces que:*

(1) Un **síntoma de incompletitud** *ocurre si el sistema de resolución de objetivos computa una cantidad finita de formas resueltas $\{\hat{S}_i\}_{i\in I}$ como respuestas para un objetivo admisible inicial $\hat{G}_0$ y el programador juzga que $Sol_{\mathcal{I}}(G_0) \not\subseteq \bigcup_{i\in I} Sol_{\mathcal{D}}(\hat{S}_i)$, mostrando así que la disyunción $\bigvee_{i\in I} \hat{S}_i$ de respuestas computadas pierde alguna solución esperada por la interpretación pretendida del programa. En consecuencia, el aca $G_0 \Rightarrow \bigvee_{i\in I} \hat{S}_i$ no será válido en $\mathcal{I}$.*

(2) Un **diagnóstico de incompletitud** *se proporciona apuntando a algún símbolo de función definida f tal que el axioma $(f)^-_{\mathcal{P}}$ para f en $\mathcal{P}^-$ no es válido en $\mathcal{I}$, mostrando así que la definición de f, tal y como está dada en $\mathcal{P}$, es incompleta con respecto a $\mathcal{I}$.*

En cualquier escenario de depuración de respuestas perdidas como el que acabamos de fijar supondremos además que el cómputo que ha dado lugar a la observación de respuestas perdidas se ha producido sin errores de tipo y sin invocaciones incompletas al resolutor.

Como ejemplos concretos de escenarios de depuración de respuestas perdidas, el lector puede remitirse a los ya presentados en la Sección 6.1.

6.4. El cálculo negativo $CNPC(\mathcal{D})$

Como se ha explicado al comienzo de este capítulo, el método de diagnosis declarativa de respuestas perdidas que vamos a proponer se apoya en la construcción de CTs como árboles de prueba abreviados con respecto a un sistema de inferencia lógicamente correcto que permite derivar *aca*s. En esta sección vamos a presentar este sistema de inferencia.

Asumamos que un síntoma de incompletitud ha sido observado por el programador para un objetivo inicial con respecto a un $CFLP(\mathcal{D})$-programa $\mathcal{P}$. En el contexto de la depuración de respuestas perdidas que nos ocupa vamos a representar los objetivos en la forma $G = R \,\square\, S$, donde $R = P \,\square\, \Delta$ es la parte *no resuelta* del objetivo G y $S = \Pi \,\square\, \sigma$ es su correspondiente *almacén de restricciones*, siendo σ una sustitución idempotente. Al igual que en el caso particular de los objetivos en forma resuelta presentados en la Sección 6.1, usaremos la notación $\hat{G}$ para representar los objetivos en la forma usual $\hat{G} = \exists \overline{U}.\ (R \,\square\, S)$, donde se indica explícitamente la cuantificación existencial de aquellas variables que se consideren intermedias o locales en G. Puesto que el sistema de resolución de objetivos ha computado una disyunción de respuestas $D = \bigvee_{i \in I} \hat{S}_i$, el *aca* $G \Rightarrow D$ que asegura que las respuestas computadas cubren todas las soluciones de G debería ser derivable a partir de $\mathcal{P}^-$. El *Cálculo de Prueba Negativo con Restricciones CNPC*$(\mathcal{D})$ (del inglés, *Constraint Negative Proof Calculus*) está formado por las reglas de inferencia que se muestran a continuación y está diseñado con el propósito de poder derivar pruebas lógicas de *aca*s de la forma $\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} G \Rightarrow D$.

Presentamos las reglas del cálculo seguidas de un comentario que incluye la definición del operador de conjunción & utilizado en la regla **CJ**. Intuitivamente, la notación $\hat{G}$ & $\hat{S}'$ representa el resultado de adjuntar a un objetivo dado $\hat{G}$ una forma resuelta $\hat{S}'$ resultante de una computación previa, de modo que la computación pueda continuar a partir del nuevo objetivo $\hat{G}$ & $\hat{S}'$. Adicionalmente, si $S = \Pi \,\square\, \sigma$ representa una forma resuelta, usamos la notación S @ σ' con σ' una sustitución idempotente para representar la nueva forma resuelta $\Pi\sigma' \,\square\, \sigma\sigma'$, y análogamente S @ π' con π' una restricción atómica primitiva para representar la nueva forma resuelta $\Pi \wedge \pi' \,\square\, \sigma$.

SF Forma Resuelta

$$\frac{}{R \,\square\, S \Rightarrow D}$$

si $Sol_{\mathcal{D}}(S) \subseteq Sol_{\mathcal{D}}(D)$.

CJ Conjunción

$$\frac{R_1 \,\square\, S \Rightarrow \bigvee_{i \in I} \exists \overline{Z}_i.\ S_i \quad \ldots \quad R_2 \,\&\, S_i \Rightarrow \bigvee_{j \in J_i} \exists \overline{Z}_{ij}.\ S_{ij} \quad \ldots \quad (i \in I)}{(R_1 \wedge R_2) \,\square\, S \Rightarrow \bigvee_{i \in I} \bigvee_{j \in J_i} \exists \overline{Z}_i, \overline{Z}_{ij}.\ S_{ij}}$$

si $\overline{Z}_i \notin var((R_1 \wedge R_2) \,\square\, S)$ y $\overline{Z}_{ij} \notin var((R_1 \wedge R_2) \,\square\, S) \cup \overline{Z}_i$ para todo $i \in I$, $j \in J_i$.

TS Deducción Trivial

$$\frac{}{G \Rightarrow D}$$

si G es atómico y se tiene garantizado que $Sol_{\mathcal{I}}(G) = \emptyset$ o bien $Sol_{\mathcal{D}}(D) = Val_{\mathcal{D}}$.

DC Descomposición

$$\frac{\overline{e_m \to t_m} \ \square \ S \Rightarrow D}{h\overline{e}_m \to h\overline{t}_m \ \square \ S \Rightarrow D}$$

si $h\overline{e}_m$ no es un patrón, donde $\overline{e_m \to t_m}$ abrevia la conjunción $e_1 \to t_1, \ldots, e_m \to t_m$.

IM Imitación

$$\frac{\overline{e_m \to X_m} \ \square \ (S \mathbin{@} \{X \mapsto h\overline{X}_m\}) \Rightarrow \bigvee_{i\in I} \exists \overline{Z}_i.\ S_i}{h\overline{e}_m \to X \ \square \ S \Rightarrow \bigvee_{i\in I} \exists \overline{X}_m, \overline{Z}_i.\ S_i}$$

si $h\overline{e}_m$ no es un patrón, $X \in \mathcal{Var}$ y $\overline{X}_m \notin var(h\overline{e}_m \to X \ \square \ S)$.

$(\mathbf{AR})_p$ Reducción de Argumentos para Funciones Primitivas

$$\frac{\overline{e_n \to X_n} \ \square \ (S \mathbin{@} p\overline{X}_n \to! \ t) \Rightarrow \bigvee_{i\in I} \exists \overline{Z}_i.\ S_i}{p\overline{e}_n \to^? t \ \square \ S \Rightarrow S_t \vee (\bigvee_{i\in I} \exists \overline{X}_n, \exists \overline{Z}_i.\ S_i)}$$

si $p \in PF^n$, $e_i \notin Pat_{\mathcal{D}}$ (para algún $1 \leq i \leq n$), $\overline{X}_n \notin var(p\overline{e}_n \to^? t \ \square \ S)$ y $\to^?$ es $\to$ (representando una producción) o bien $\to!$ (representando una restricción). Usaremos la notación abreviada S_t para representar $S \mathbin{@} \{R \mapsto \bot\}$ si t es una variable R, o bien para representar la disyunción vacía $\blacklozenge$ en otro caso.

$(\mathbf{AR})_f$ Reducción de Argumentos para Funciones Definidas

$$\frac{(\overline{e_n \to X_n} \wedge f \ \overline{X}_n \to t) \ \square \ S \Rightarrow \bigvee_{i\in I} \exists \overline{Z}_i.\ S_i}{f \ \overline{e}_n \to t \ \square \ S \Rightarrow \bigvee_{i\in I} \exists \overline{X}_n, \overline{Z}_i.\ S_i}$$

si $f \in DF^n$, $e_i \notin Pat_{\mathcal{D}}$ (para algún $1 \leq i \leq n$) y $\overline{X}_n \notin var(f\overline{e}_n \to t \ \square \ S)$.

$$\frac{(\overline{e_n \to X_n} \wedge f \ \overline{X}_n \to Y \wedge Y\overline{a}_k \to t) \ \square \ S \Rightarrow \bigvee_{i\in I} \exists \overline{Z}_i.\ S_i}{f \ \overline{e}_n\overline{a}_k \to t \ \square \ S \Rightarrow \bigvee_{i\in I} \exists \overline{X}_n, Y, \overline{Z}_i.\ S_i}$$

si $f \in DF^n$ $(k > 0)$, $e_i \notin Pat_{\mathcal{D}}$ (para algún $1 \leq i \leq n$) y $\overline{X}_n, Y \notin var(f\overline{e}_n\overline{a}_k \to t \ \square \ S)$.

(DF)$_f$ Función Definida

$$\frac{\cdots\ R_i[\ \overline{X_n \mapsto t_n},\ Y \mapsto t\ ]\ \square\ S \Rightarrow D_i\ \cdots\ (i \in I)}{\boxed{f\ \bar{t}_n\ \rightarrow\ t\ \square\ S\ \Rightarrow\ S_t\ \vee\ (\bigvee_{i\in I} D_i)}}$$

si $f \in DF^n$, $\overline{X}_n, Y \notin var(f\ \bar{t}_n \rightarrow t\ \square\ S)$ y $(f\ \overline{X}_n \rightarrow Y \Rightarrow (\bigvee_{i\in I} \hat{R}_i) \vee (\bot \rightarrow Y)) \in_{var} \mathcal{P}^-$, variante del axioma $(f)^-_{\mathcal{P}}$ elegido con variables nuevas. Asimismo, la notación S_t utilizada en la conclusión de esta regla es la misma que ya ha sido definida anteriormente para la regla **(AR)**$_p$.

En el cálculo $CPPC(\mathcal{D})$ utilizado en el Capítulo 5 se derivaban c-sentencias de la forma $G \Leftarrow \Pi$, con el convenio de que si G era una conjunción, derivar $G \Leftarrow \Pi$ se reducía a derivar $\alpha \Leftarrow \Pi$ para cada parte atómica α de G. En el cálculo $CNPC(\mathcal{D})$ que ahora nos ocupa se derivan *aca*s de la forma $G \Rightarrow D$, en las cuales hay que tratar adecuadamente a G como conjunción en el caso de que no sea atómico. Es por esta razón por la que se ha introducido la regla de inferencia **CJ** en el cálculo, la cual permite inferir un *aca* para un objetivo cuya parte no resuelta está compuesta por una conjunción de la forma $(R_1 \wedge R_2) \square S$ a partir de *aca*s para objetivos con partes no resueltas de la forma $R_1 \square S$ y R_2 & S_i. Gracias a la utilización de esta regla, las demás reglas de inferencia del cálculo pueden formularse con la finalidad de poder tratar con tipos diferentes de objetivos cuya parte no resuelta sea ya atómica.

Para la formalización de la regla de conjunción **CJ** hemos hecho uso de un operador especial de conjunción, denotado mediante el símbolo &. Mediante el operador & es posible expresar de una manera cómoda el resultado de adjuntar a un objetivo dado G una forma resuelta $\hat{S}'$ resultante de una computación previa, de modo que la computación pueda continuar a partir del nuevo objetivo $\hat{G}$ & $\hat{S}'$. Debido a la importancia posterior de este operador en el diseño de un cálculo de resolución de objetivos admisible para $CNPC(\mathcal{D})$, precisamos a continuación con mayor detalle la definición formal de este operador y analizamos cuales son sus principales propiedades.

Definición 32 (Operador de Conjunción &) *Sea $\hat{G} = \exists\overline{U}$. G un objetivo admisible con $G = R \square S$ y $S = \Pi \square \sigma$ tal que*

- $vdom(\sigma) \cap \overline{U} = \emptyset$,
- $R\sigma = R$,
- $\Pi\sigma = \Pi$.

Sea $\hat{S}' = \exists\overline{U}'$. S' un objetivo en forma resuelta con $S' = \Pi' \square \sigma'$ tal que

- $\overline{U} \setminus vdom(\sigma') \subseteq \overline{U}'$,

- $\sigma\sigma' = \sigma'$,
- $Sol_{\mathcal{D}}(\Pi') \subseteq Sol_{\mathcal{D}}(\Pi\sigma')$.

Definimos $G \;\&\; S' = R\sigma' \sqcap S'$ *y* $\hat{G} \;\&\; \hat{S}' = \exists\overline{U}'.\ (R\sigma' \sqcap S')$.

Las siguientes propiedades básicas del operador & se demuestran utilizando la Definición 32 y la Definición 11 del Capítulo 3:

Proposición 9 (Propiedades Básicas del Operador &) *Sean* $\hat{G}$, $\hat{S}'$ *y* $\hat{G} \;\&\; \hat{S}'$ *como aparecen en la Definición 32 y sea* $\mathcal{I}$ *cualquier c-interpretación sobre el dominio de restricciones* $\mathcal{D}$. *Se verifican las siguientes propiedades:*

(1) $Sol_{\mathcal{I}}(\hat{G} \;\&\; \hat{S}') = Sol_{\mathcal{I}}(\hat{G}) \cap Sol_{\mathcal{D}}(\hat{S}')$.

(2) $(\hat{G} \;\&\; \hat{S}') \;\&\; \hat{S}' = \hat{G} \;\&\; \hat{S}'$.

(3) $\hat{G} = \hat{R} \;\&\; \hat{S}$.

Demostración 20

(1) *(*$\subseteq$*) Sea* $\mu \in Sol_{\mathcal{I}}(\hat{G} \;\&\; \hat{S}')$. *Por definición, existe* $\mu' =_{\setminus\overline{U}'} \mu$ *tal que* $\mu' \in Sol_{\mathcal{I}}(R\sigma' \sqcap S')$. *Por tanto,* $\mu' \in Sol_{\mathcal{I}}(R\sigma')$ *y* $\mu' \in Sol_{\mathcal{D}}(S')$. *Más aún,* $\mu \in Sol_{\mathcal{D}}(\hat{S}')$ *debido a que* $\mu' =_{\setminus\overline{U}'} \mu$ *y* $\hat{S}' = \exists\overline{U}'.\ S'$. *Por otra parte, puesto que* $S' = \Pi' \sqcap \sigma'$, *se sigue que* $\mu' \in Sol_{\mathcal{D}}(\Pi')$ *y* $\mu' \in Sol(\sigma')$, *y entonces* $\sigma'\mu' = \mu'$ *(para cada vínculo* $\{X \mapsto t\} \in \sigma'$, $X\sigma'\mu' = t\mu' = X\mu'$ *debido a que* $\mu' \in Sol(\sigma')$, *y para cualquier* $X \notin vdom(\sigma')$, $X\sigma'\mu' = X\mu'$*). Más aún, puesto que* $\mu' \in Sol_{\mathcal{I}}(R\sigma')$ *también tenemos que* $\sigma'\mu' \in Sol_{\mathcal{I}}(R)$ *y entonces* $\mu' \in Sol_{\mathcal{I}}(R)$ *por ser* $\sigma'\mu' = \mu'$. *Más aún, puesto que* $Sol_{\mathcal{D}}(\Pi') \subseteq Sol_{\mathcal{D}}(\Pi\sigma')$ *por definición y* $\mu' \in Sol_{\mathcal{D}}(\Pi')$, *también tenemos que* $\mu' \in Sol_{\mathcal{D}}(\Pi\sigma')$, *y entonces* $\sigma'\mu' \in Sol_{\mathcal{D}}(\Pi)$, *o equivalentemente* $\mu' \in Sol_{\mathcal{D}}(\Pi)$. *Puesto que por definición* $\sigma' = \sigma\sigma'$ *y* $\mu' \in Sol(\sigma')$ *entonces* $\mu' \in Sol(\sigma)$. *Deducimos que* $\mu' \in Sol_{\mathcal{D}}(\Pi \sqcap \sigma)$ *y entonces* $\mu' \in Sol_{\mathcal{D}}(S)$. *Por tanto,* $\mu' \in Sol_{\mathcal{I}}(R \sqcap S)$. *Finalmente, puesto que se verifica que* $\mu' =_{\setminus\overline{U}'} \mu$, *donde* $\mu' = \sigma'\mu'$ *y* $\overline{U} \setminus vdom(\sigma') \subseteq \overline{U}'$, *deducimos que* $\mu =_{\setminus\overline{U}} \mu'$ *y podemos por tanto concluir que* $\mu \in Sol_{\mathcal{I}}(\hat{G})$ *debido a que* $\hat{G} = \exists\overline{U}.\ (R \sqcap S)$.

($\supseteq$*) Sea* $\mu \in Sol_{\mathcal{I}}(\hat{G}) \cap Sol_{\mathcal{D}}(\hat{S}')$. *Entonces,* $\mu \in Sol_{\mathcal{I}}(\hat{G})$ *y* $\mu \in Sol_{\mathcal{D}}(\hat{S}')$. *Por definición, existe* $\mu' =_{\setminus\overline{U}'} \mu$ *tal que* $\mu' \in Sol_{\mathcal{D}}(S')$ *y entonces* $\mu' \in Sol(\sigma')$. *Más aún, existe* $\mu'' =_{\setminus\overline{U}} \mu$ *tal que* $\mu'' \in Sol_{\mathcal{I}}(R \sqcap S)$ *y entonces* $\mu'' \in Sol_{\mathcal{I}}(R)$. *Además,* $\mu'' =_{\setminus\overline{U}'} \mu$ *puesto que* $\overline{U} \setminus vdom(\sigma') \subseteq \overline{U}'$, *y por tanto* $\mu'' =_{\setminus\overline{U}'} \mu'$ *ya que* $\mu'' =_{\setminus\overline{U}'} \mu =_{\setminus\overline{U}'} \mu'$. *Entonces,* $\mu' \in Sol_{\mathcal{I}}(R)$ *y* $\mu' \in Sol(\sigma')$. *Se sigue*

que $\sigma'\mu' = \mu'$ *(para cada vínculo* $\{X \mapsto t\} \in \sigma'$, $X\sigma'\mu' = t\mu' = X\mu'$ *porque* $\mu' \in Sol(\sigma')$, *y para cualquier* $X \notin vdom(\sigma')$, $X\sigma'\mu' = X\mu'$*) y entonces* $\mu' \in Sol_{\mathcal{I}}(R\sigma')$. *Más aún, puesto que* $\mu' \in Sol_{\mathcal{D}}(S')$, *obtenemos* $\mu' \in Sol_{\mathcal{I}}(R\sigma' \square S')$. *Finalmente concluimos que* $\mu \in Sol_{\mathcal{I}}(\hat{G} \;\&\; \hat{S}')$ *por definición, debido a que* $\mu =_{\setminus \overline{U}'} \mu'$.

(2) Por definición del operador &, se tiene que $\hat{G} \;\&\; \hat{S}' = \exists\overline{U}'.\ (R\sigma' \square S')$. *Como* $\overline{U}' \setminus vdom(\sigma') \subseteq \overline{U}'$, $\sigma'\sigma' = \sigma'$ *debido a que* σ' *es una sustitución idempotente, y* $Sol_{\mathcal{D}}(\Pi') \subseteq Sol_{\mathcal{D}}(\Pi'\sigma') = Sol_{\mathcal{D}}(\Pi')$ *debido a que* $\Pi'\sigma' = \Pi'$ *por las condiciones de admisibilidad, podemos aplicar la definición del operador & y obtener* $(\hat{G} \;\&\; \hat{S}') \;\&\; \hat{S}' = \exists\overline{U}'.\ (R\sigma'\sigma' \square S')$. *Puesto que* σ' *es una sustitución idempotente,* $\sigma'\sigma' = \sigma'$, *y entonces* $(\hat{G} \;\&\; \hat{S}') \;\&\; \hat{S}' = \exists\overline{U}'.\ (R\sigma' \square S') = \hat{G} \;\&\; \hat{S}'$.

(3) Sea $\hat{G} = \exists\overline{U}.\ (R \square S)$ *un objetivo admisible, donde* $\hat{R} = \exists\overline{U}.\ R$ *(o equivalentemente,* $\hat{R} = \exists\overline{U}.\ (R \square (\Diamond \square \{\}))$, *puesto que* $\hat{R}$ *es la parte no resuelta del objetivo* $\hat{G}$ *con una parte resuelta vacía* $(\Diamond \square \{\})$ *tal que* $vdom(\{\}) \cap \overline{U} = \emptyset$, $R\{\} = R$ *y* $\Diamond\{\} = \Diamond$*) y* $\hat{S} = \exists\overline{U}.\ S$ *con* $S = \Pi \square \sigma$. *Claramente,* $\overline{U} \setminus vdom(\sigma) \subseteq \overline{U}$, $\{\}\sigma = \sigma$ *y* $Sol_{\mathcal{D}}(\Pi) \subseteq Sol_{\mathcal{D}}(\Diamond\sigma) = Sol_{\mathcal{D}}(\Diamond) = Val_{\mathcal{D}}$. *Entonces, podemos aplicar la definición del operador & para obtener* $\hat{R} \;\&\; \hat{S} = \exists\overline{U}.\ (R\sigma \square S)$. *Puesto que* $R\sigma = R$ *por las condiciones de admisibilidad del objetivo* $\hat{G}$, *obtenemos* $\hat{R} \;\&\; \hat{S} = \exists\overline{U}.\ (R \square S) = \hat{G}$.

□

6.4.1. Árboles de prueba negativos

Cualquier derivación $\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} G \Rightarrow D$ en el cálculo $CNPC(\mathcal{D})$ puede ser representada en la forma de un *Árbol de Prueba Negativo* (abreviadamente *NPT*, del inglés *Negative Proof Tree*) con restricciones sobre un dominio $\mathcal{D}$. Este árbol de prueba tiene *aca*s en sus nodos, de tal manera que el *aca* en cualquiera de sus nodos se puede inferir a partir de los *acas* de sus hijos usando alguna de las reglas de inferencia del cálculo $CNPC(\mathcal{D})$.

La Figura 6.6 muestra un *NPT* que permite representar un fragmento de la $CNPC(\mathcal{H})$-derivación que sirve de testigo al cómputo con respuestas perdidas del ejemplo de las relaciones de familia presentado en la Sección 6.1. Análogamente, la Figura 6.7 muestra el *NPT* completo que usamos en el diagnóstico de respuestas perdidas en el ejemplo también presentado al final de la Sección 6.1. Los nodos críticos se resaltan rodeando el *aca* asociado dentro de un rectángulo doble.

Nuestro propósito será el de usar una forma abreviada de *NPT*s como *CT*s útiles en la depuración de respuestas perdidas. Por ejemplo, el *CT* que se muestra en la Figura 6.2 se corresponde con el árbol abreviado construido a partir del *NPT* de la Figura 6.6, mientras que el *CT* de la Figura 6.5 se corresponde con el árbol abreviado del *NPT* de la Figura 6.7.

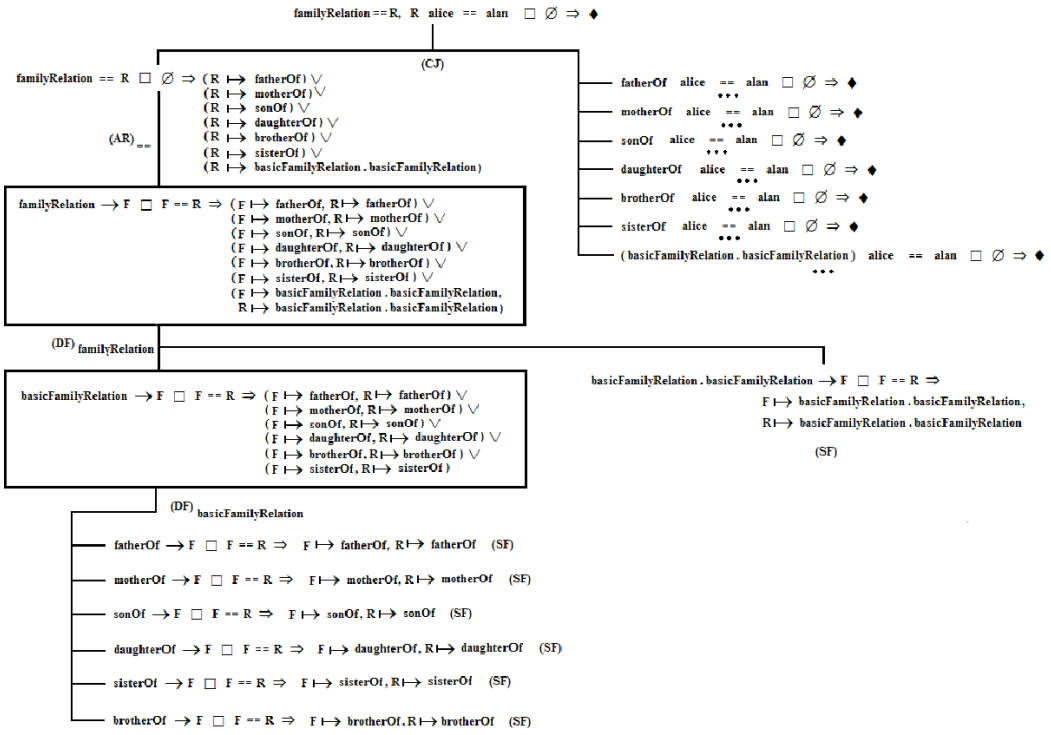


Figura 6.6: *NPT* para el ejemplo de las relaciones de familia perdidas

El siguiente teorema permite mostrar que cualquier *aca* que haya sido derivado por medio de un *NPT* es consecuencia lógica de la teoría negativa asociada al programa correspondiente. Este resultado será usado para demostrar la corrección de nuestro método de diagnóstico.

Teorema 15 (Corrección Semántica del Cálculo $CNPC(\mathcal{D})$**)** *Sea $G \Rightarrow D$ un aca cualquiera para un $CFLP(\mathcal{D})$-programa $\mathcal{P}$ dado. Si $\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} G \Rightarrow D$ (con un cierto NPT como testigo de la derivación) entonces $\mathcal{P}^- \models_{\mathcal{D}} G \Rightarrow D$ (en el sentido explicado en el apartado (2) de la Definición 30).*

Demostración 21 *Comenzamos observando que cada una de las reglas de inferencia* **(RI)** *del cálculo $CNPC(\mathcal{D})$ es de la forma:*

$$\frac{G_1 \Rightarrow D_1 \quad \ldots \quad G_m \Rightarrow D_m}{G \Rightarrow D} \qquad \textbf{(RI)}$$

donde $G_i \Rightarrow D_i$ $(1 \leq i \leq m)$ son las premisas *de la regla y $G \Rightarrow D$ es la* conclusión*. Cada una de estas reglas de inferencia es semánticamente correcta por separado en*

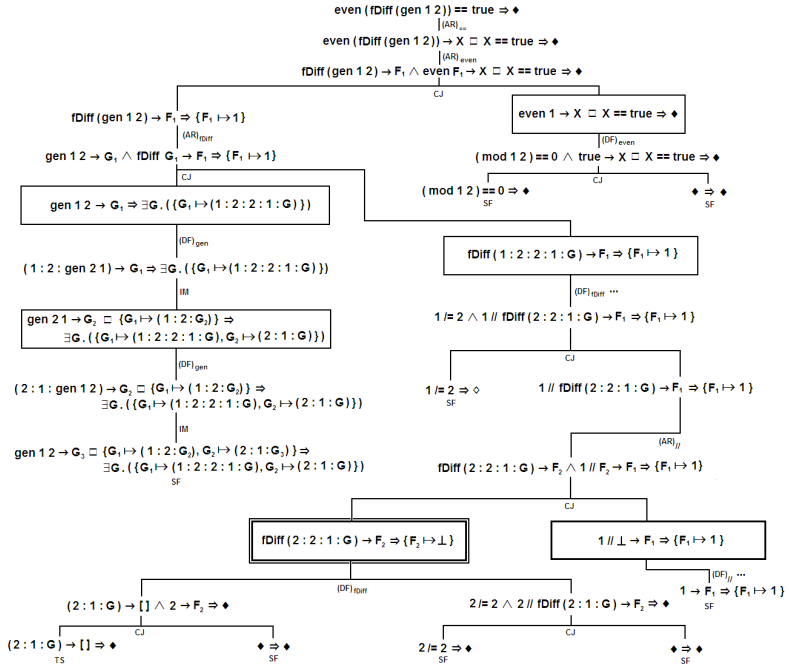


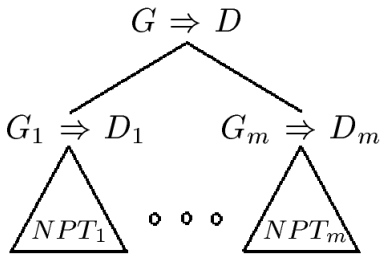
Figura 6.7: *NPT* para la depuración de respuestas perdidas y evaluación perezosa

el siguiente sentido: cualquier modelo $\mathcal{I} \models_{\mathcal{D}} \mathcal{P}^-$ *que satisfaga las premisas (es decir, tal que* $Sol_{\mathcal{I}}(G_i) \subseteq Sol_{\mathcal{D}}(D_i)$ *para todo* $1 \leq i \leq m$*) también verifica la conclusión (es decir,* $Sol_{\mathcal{I}}(G) \subseteq Sol_{\mathcal{D}}(D)$*). Una demostración de esta propiedad de corrección de manera individual para cada una de las reglas de inferencia se da en el Apéndice A. Más aún, la suposición de que la c-interpretación* $\mathcal{I}$ *es un modelo de* $\mathcal{P}^-$ *es necesaria sólo para la regla de inferencia* $\mathbf{(DF)}_f$*. Cualquier otra regla de inferencia* **(RI)** *del cálculo es correcta con respecto a una c-interpretación arbitraria* $\mathcal{I}$ *sobre el dominio.*

Asumamos ahora que la derivación $\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} G \Rightarrow D$ *ha sido probada con un NPT de una cierta profundidad p. Con el fin de poder concluir que* $\mathcal{P}^- \models_{\mathcal{D}} G \Rightarrow D$*, como se pide en el enunciado del teorema, y debido a la Definición 30, debemos demostrar que* $G \Rightarrow D$ *se satisface en* $\mathcal{I}$ *(es decir, la inclusión* $Sol_{\mathcal{I}}(G) \subseteq Sol_{\mathcal{D}}(D)$ *se cumple), siendo* $\mathcal{I}$ *cualquier c-interpretación arbitrariamente fijada tal que* $\mathcal{I} \models_{\mathcal{D}} \mathcal{P}^-$*. Esto puede ser probado fácilmente razonando por inducción sobre p, usando la ya demostrada corrección individual de cada regla de inferencia* **(RI)** *del cálculo.*

Caso base*:* $p = 0$*. En este caso, el aca* $G \Rightarrow D$ *ha sido inferido mediante la aplicación de una regla de inferencia* **(RI)** *con cero premisas. Puesto que esta regla es correcta y todas sus (cero) premisas se satisfacen trivialmente en* $\mathcal{I}$*, podemos concluir que* $G \Rightarrow D$ *también se satisface en* $\mathcal{I}$*.*

Paso inductivo*:* $p > 0$*. En este caso, el NPT que sirve de testigo a la derivación* $\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} G \Rightarrow D$ *es de la forma:*



donde la profundidad de cada NPT_i *es* $p_i < p$ *y el aca* $G \Rightarrow D$ *ha sido inferido a partir de los* m *acas* $G_i \Rightarrow D_i$ *por medio de una regla de inferencia* **(RI)** *con* m *premisas. Entonces,* $G_i \Rightarrow D_i$ *se satisface en* $\mathcal{I}$ *para todo* $1 \leq i \leq m$ *por* hipótesis de inducción*, y puesto que el paso* **(RI)** *es correcto, podemos concluir que* $G \Rightarrow D$ *también se satisface en* $\mathcal{I}$*.*

□

6.4.2. Cálculos de resolución de objetivos $CNPC(\mathcal{D})$-admisibles

Decimos que un sistema de resolución de objetivos para el esquema $CFLP(\mathcal{D})$ es $CNPC(\mathcal{D})$-*admisible* si siempre que una cantidad finita de formas resueltas $\{\hat{S}_i\}_{i \in I}$ son computadas como respuestas para un objetivo admisible inicial $\hat{G}_0$ cuyo espacio de búsqueda es finito se tiene una derivación $\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} G_0 \Rightarrow \bigvee_{i \in I} \hat{S}_i$ en el cálculo $CNPC(\mathcal{D})$ con testigo un NPT sobre el dominio $\mathcal{D}$.

Debido a lo explicado en la subsección anterior, el buen comportamiento de nuestro método de depuración de respuestas perdidas se puede justificar formalmente para un escenario de depuración en el cual las respuestas observadas por el usuario hayan sido calculadas por un sistema de resolución de objetivos $CNPC(\mathcal{D})$-admisible. Por analogía con el Teorema 12 del capítulo anterior, cabe conjeturar que los dos cálculos formales de resolución de objetivos presentados en el Capítulo 4 (a saber, $CLNC(\mathcal{D})$ y $CDNC(\mathcal{D})$) son $CNPC(\mathcal{D})$-admisibles. Verificar esta conjetura para estos cálculos concretos es sin embargo complicado, debido a las dificultades técnicas asociadas a la aplicación de la regla **CJ** del cálculo $CNPC(\mathcal{D})$, que no se corresponde exactamente con la aplicación de reglas de transformación de objetivos en estos cálculos operacionales. En su lugar, hemos optado por demostrar la $CNPC(\mathcal{D})$-admisibilidad de un cálculo de resolución de objetivos más laxo que los presentados en el Capítulo 4, al que denominaremos $RGSC(\mathcal{D})$ (del inglés, *Relaxed Goal Solving Calculus*). En $RGSC(\mathcal{D})$ no se utilizan árboles definicionales, no se incluye ninguna regla para vincular variables libres de orden superior, y se reemplaza la invocación a un resolutor de restricciones por una regla más abstracta (véase **SC** más abajo) que pretende representar el comportamiento ideal de un resolutor completo. Pese a estas limitaciones pensamos que $RGSC(\mathcal{D})$ proporciona una caracterización formal de la

resolución de objetivos adecuada al escenario de depuración de respuestas perdidas que pretendemos tratar. La omisión de la regla de vinculación de variables de orden superior se justifica por el hecho de que dicha regla da lugar a espacios de búsqueda infinitos, que caen fuera de nuestro escenario. La regla **SC** de simplificación de restricciones se justifica por la intención de diagnosticar respuestas perdidas a causa de la insuficiencia de las reglas del programa, bajo el supuesto de un comportamiento ideal del resolutor.

El cálculo $RGSC(\mathcal{D})$ se compone de las siguientes reglas de transformación de objetivos:

DC Descomposición

$$\exists \overline{U}.\ ((h\overline{e}_m \to h\overline{t}_m \wedge R) \sqcap S) \vdash_{DC_1} \exists \overline{U}.\ ((\overline{e_m \to t_m} \wedge R) \sqcap S)$$

si $h\overline{e}_m$ es pasivo.

$$\exists \overline{U}.\ ((t \to t \wedge R) \sqcap S) \vdash_{DC_2} \exists \overline{U}.\ (R \sqcap S) \quad \text{si } t \in Pat_{\mathcal{D}}.$$

SP Producción Simple

$$\exists X, \overline{U}.\ ((t \to X \wedge R) \sqcap S) \vdash_{SP_1} \hat{R}\ \&\ \hat{S}'$$

donde $S' \equiv S \mathbin{@} \{X \mapsto t\}$.

$$\exists \overline{U}.\ ((X \to t \wedge R) \sqcap S) \vdash_{SP_2} \hat{R}\ \&\ \hat{S}'$$

si $t \notin \mathcal{V}ar$, donde $S' \equiv S \mathbin{@} \{X \mapsto t\}$.

IM Imitación

$$\exists \overline{U}.\ ((h\overline{e}_m \to X \wedge R) \sqcap S) \vdash_{IM} \exists \overline{X}_m, \overline{U}.\ ((\overline{e_m \to X_m} \wedge h\overline{X}_m \to X \wedge R) \sqcap S)$$

si $h\overline{e}_m \notin Pat_{\mathcal{D}}$ es pasivo y $X \in dvar_{\mathcal{D}}(R \sqcap S)$.

EL Eliminación

$$\exists X, \overline{U}.\ (e \to X \wedge R) \sqcap S \vdash_{EL} \exists \overline{U}.\ (R \sqcap S)$$

si $X \notin var(R \sqcap \Pi)$, donde $S \equiv \Pi \sqcap \sigma$.

$\mathbf{PF}_p$ Función Primitiva

$$\exists\overline{U}.\ ((p\overline{e}_n \to^? t \wedge R) \sqcap S) \Vdash_{PF_p} (\exists\overline{X}_n, \overline{U}.\ (\overline{e_n \to X_n} \wedge R)) \ \&\ \hat{S}'$$

si $p \in PF^n$, $t \notin \mathcal{V}ar$ o $t \in dvar_{\mathcal{D}}(R \sqcap S)$, y $S' \equiv S \ @\ p\overline{X}_n \to! t$.

$\mathbf{DF}_f$ Función Definida

$$\exists\overline{U}.\ ((f\overline{e}_n \to t \wedge R) \sqcap S) \Vdash_{DF_{f,1}} \exists\overline{X}_n, Y, \overline{Y}_{i_0}, \overline{U}.\ ((\overline{e_n \to X_n} \wedge Y \to t \wedge R_{i_0} \wedge R) \sqcap S)$$

$$\exists\overline{U}.\ (f\overline{e}_n\overline{a}_k \to t \wedge R) \sqcap S \Vdash_{DF_{f,2}} \exists\overline{X}_n, Y, \overline{Y}_{i_0}, \overline{U}.\ ((\overline{e_n \to X_n} \wedge Y\overline{a}_k \to t \wedge R_{i_0} \wedge R) \sqcap S) \quad \text{si } k > 0$$

donde $f \in DF^n$, $t \notin \mathcal{V}ar$ o $t \in dvar_{\mathcal{D}}(R \sqcap S)$, $(f\overline{X}_n \to Y \Rightarrow \bigvee_{i\in I} \hat{R}_i) \in_{var} \mathcal{P}^-$ siendo $\hat{R}_i = \exists\overline{Y}_i.\ R_i$ para cada $i \in I$, e $i_0 \in I$.

SC Simplificación de Restricciones

$$\exists\overline{U}.\ (R \sqcap S) \Vdash_{SC} \hat{R}\ \&\ \hat{S}_{i_0}$$

si $Sol_{\mathcal{D}}(\hat{S}) = \bigcup_{i\in I} Sol_{\mathcal{D}}(\hat{S}_i)$ e $i_0 \in I$.

CF Fallo por Conflicto

$$\exists\overline{U}.\ ((h\overline{e}_p \to h'\overline{t}_q \wedge R) \sqcap S) \Vdash_{CF_1} \blacksquare \quad \text{si } h \neq h' \text{ o } p \neq q.$$

$$\exists\overline{U}.\ ((u \to u' \wedge R) \sqcap S) \Vdash_{CF_2} \blacksquare \quad \text{si } u, u' \in \mathcal{B}^{\mathcal{D}} \text{ pero } u \neq_{\mathcal{B}^{\mathcal{D}}} u'.$$

$$\exists\overline{U}.\ ((h\overline{e}_p \to u' \wedge R) \sqcap S) \Vdash_{CF_3} \blacksquare \quad \text{si } u' \in \mathcal{B}^{\mathcal{D}}.$$

$$\exists\overline{U}.\ ((u \to h\overline{t}_q \wedge R) \sqcap S) \Vdash_{CF_4} \blacksquare \quad \text{si } u \in \mathcal{B}^{\mathcal{D}}.$$

FC Fallo en la Resolución de Restricciones

$$\exists\overline{U}.\ (R \sqcap S) \Vdash_{FC} \blacksquare \quad \text{si } Sol_{\mathcal{D}}(\hat{S}) = \emptyset.$$

Escribiremos $\hat{G} \Vdash_{\mathcal{P},RGSC(\mathcal{D})} \hat{G}'$ (respectivamente, $\hat{G} \Vdash^*_{\mathcal{P},RGSC(\mathcal{D})} \hat{G}'$) para indicar que el objetivo $\hat{G}$ puede ser reducido a un nuevo objetivo $\hat{G}'$ en un paso (respectivamente, en una cantidad finita de pasos) usando el programa $\mathcal{P}$ y las reglas de

transformación de objetivos del cálculo $RGSC(\mathcal{D})$. Observamos que las condiciones técnicas requeridas para $\hat{G}$ & $\hat{S}'$ en la Definición 32 quedan garantizadas trivialmente en el caso en el que $\hat{G} \Vdash^{*}_{\mathcal{P},RGSC(\mathcal{D})} \exists\overline{U}'.\ (R' \square S')$.

Un paso de transformación de objetivos $\hat{G} \Vdash_{\mathcal{P},RGSC(\mathcal{D})} \hat{G}'$ puede interpretarse como un espacio de búsqueda en forma de árbol si se piensa en $\hat{G}'$ como un hijo de $\hat{G}$. En los nodos de este árbol se tendrían objetivos, situándose en la raíz el objetivo inicial del cómputo y en las hojas del árbol formas resueltas. Más aún, escribiremos $(\underline{\widehat{R_1 \wedge R_2}})$ & $\hat{S} \Vdash^{p}_{\mathcal{P},RGSC(\mathcal{D})} D$ para indicar que el objetivo $\hat{G} = (\underline{\widehat{R_1 \wedge R_2}})$ & $\hat{S}$ tiene un *espacio de búsqueda parcialmente desarrollado* $\mathcal{S}$ de *tamaño* finito p, siendo p el número de nodos del árbol que representa este espacio de búsqueda construido usando el programa $\mathcal{P}$, el cálculo de resolución de objetivos laxo $RGSC(\mathcal{D})$ y una cierta *estrategia de selección* que permite seleccionar sólo descendientes atómicos de la parte R_1 del objetivo inicial, y tal que la disyunción de las partes resueltas $\hat{S}_i$ de las *hojas* de $\mathcal{S}$ es D. Esta notación jugará un papel clave en el resto de esta sección. Remarcamos ahora varios puntos importantes en relación a la notación que acabamos de introducir:

- La notación $(\underline{R_1} \wedge R_2)$ debe ser interpretada *modulo* la conmutatividad y asociatividad de la conjunción, lo que significa que la parte no resuelta R de un objetivo ha de ser vista como un conjunto (y no como una secuencia) de partes atómicas, dividida en dos partes, R_1 y R_2.

- Puesto que $\mathcal{S}$ no es necesariamente un espacio de búsqueda terminado, algunas de sus hojas podrían no estar en forma resuelta. Cada miembro de la disyunción D es exactamente la parte resuelta de una hoja.

- Si un objetivo $\hat{G}$ tiene un *espacio de búsqueda finito* totalmente desarrollado $\mathcal{S}$ de *tamaño* p éste puede ser escrito también como $\hat{G} \Vdash^{p}_{\mathcal{P},RGSC(\mathcal{D})} D$. Cuando usamos esta notación no se impone ningún requerimiento especial a la estrategia de selección, de forma que D representa la disyunción finita de las hojas $\hat{S}_i$ de $\mathcal{S}$, las cuales están todas ellas en forma resuelta. En el caso en el que $\mathcal{S}$ sea un espacio de búsqueda correspondiente a una situación de *fallo finito* sin ninguna hoja con éxito, entonces D se reduciría a $\blacklozenge$ (es decir, a una disyunción vacía de formas resueltas, indicando así el fallo).

El siguiente lema técnico es también una herramienta clave para la demostración del Teorema 16, que nos va a permitir probar que el cálculo de resolución de objetivos $RGSC(\mathcal{D})$ es $CNPC(\mathcal{D})$-admisible. De manera informal, el lema afirma que las respuestas computadas para una conjunción $(R_1 \wedge R_2)$ pueden ser descompuestas en dos pasos: primero computando respuestas para R_1 y a continuación procediendo con la computación de respuestas para R_2 como una extensión de cada respuesta particular de R_1. Formalmente:

Lema 16 (Lema de Conjunción) *Asumamos que* $(\widehat{R_1 \wedge R_2}) \;\&\; \hat{S} \Vdash^{p}_{\mathcal{P},RGSC(\mathcal{D})} D$ *(con un espacio de búsqueda parcialmente desarrollado de tamaño* p*). Entonces, es posible encontrar espacios de búsqueda parcialmente desarrollados del modo siguiente:*

(1) $(\widehat{\underline{R_1} \wedge R_2}) \;\&\; \hat{S} \Vdash^{q}_{\mathcal{P},RGSC(\mathcal{D})} \bigvee_{i\in I} \hat{S}_i$ *(espacio de búsqueda parcialmente desarrollado de tamaño* $q \leq p$*).*

(2) $\forall i \in I : (\widehat{R_1 \wedge \underline{R_2}}) \;\&\; \hat{S}_i \Vdash^{q_i}_{\mathcal{P},RGSC(\mathcal{D})} D_i$ *(espacios de búsqueda parcialmente desarrollados de tamaños* $q_i \leq p$*).*

(3) $\bigvee_{i\in I} D_i = D$.

La demostración de este lema, basada a su vez en dos lemas auxiliares, puede encontrarse en el Apéndice A.

El siguiente teorema trata de proporcionar una base teórica a la suposición pragmática de que los cálculos de resolución de objetivos presentados en el Capítulo 4, los cuales sirven de base formal a los métodos de resolución de objetivos en sistemas *CFLP* actuales como *Curry* [Han06] o $\mathcal{TOY}$ [ALR07], son $CNPC(\mathcal{D})$-admisibles.

Teorema 16 (Existencia de Cálculos de Resolución de Objetivos $CNPC(\mathcal{D})$ **Admisibles)** *El cálculo de resolución de objetivos laxo* $RGSC(\mathcal{D})$ *presentado en esta sección es* $CNPC(\mathcal{D})$*-admisible.*

Demostración 22 *Por las propiedades del operador & vistas en la Proposición 9, sabemos que un objetivo admisible* $\hat{G} = \exists \overline{U}.\ (R \sqcap S)$ *se puede escribir equivalentemente en cualquiera de las dos formas siguientes:* $\widehat{R \sqcap S}$ *o* $\hat{R} \;\&\; \hat{S}$*. Tanto aquí como en las demostraciones del Apéndice A.4, usaremos implícitamente la equivalencia entre las dos notaciones,* $\widehat{R \sqcap S}$ *y* $\hat{R} \;\&\; \hat{S}$*, en muchos casos referidas a un objetivo cuya parte no resuelta está expresada en forma de conjunción.*

El Teorema 16 puede verse como un caso particular de un resultado más general, enunciado como sigue: si $(\widehat{\underline{R} \wedge R'}) \;\&\; \hat{S} \Vdash^{p}_{\mathcal{P},RGSC(\mathcal{D})} D$ *(con un espacio de búsqueda parcialmente desarrollado de tamaño finito* p*, construido usando el programa* $\mathcal{P}$*, el cálculo de resolución de objetivos* $RGSC(\mathcal{D})$ *y una estrategia de selección que solo permite seleccionar átomos descendientes de la parte* R*), entonces existe una derivación* $\mathcal{P}^{-} \vdash_{CNPC(\mathcal{D})} R \sqcap S \Rightarrow D$ *en el cálculo de prueba negativo* $CNPC(\mathcal{D})$*, con un cierto árbol de prueba negativo* NPT *que sirve de testigo.*

Nótese que el Teorema 16 se obtiene como caso particular de este resultado cuando R' *es vacío,* $\hat{R} \;\&\; \hat{S}$ *es un objetivo admisible inicial* $\hat{G}_0$*, el espacio de búsqueda está totalmente desarrollado y es finito, y* $D = \bigvee_{i\in I} \hat{S}_i$ *es la disyunción finita de todas las formas resueltas calculadas por* $RGSC(\mathcal{D})$ *a partir de* $\hat{G}_0$.

La demostración del resultado más general procede razonando por inducción sobre p, usando el Lema de Conjunción *para poder tratar objetivos compuestos cuya parte no resuelta sea una conjunción. Los detalles concretos de esta demostración se dan en el Apéndice A.4.*

□

6.5. Depuración declarativa de respuestas perdidas mediante árboles de prueba negativos abreviados

En este punto ya estamos preparados para presentar un método completo de diagnosis declarativa de respuestas perdidas basado en *NPT*s que nos permita proporcionar diagnósticos correctos para cualquier sistema de resolución de objetivos que sea $CNPC(\mathcal{D})$-admisible. En primer lugar, vamos a demostrar que los síntomas de incompletitud son causados por conjuntos incompletos de reglas de programa. Esto queda garantizado a través del siguiente teorema:

Teorema 17 (Síntomas de Incompletitud) *Asumamos que se ha observado un síntoma de incompletitud para un $CFLP(\mathcal{D})$-programa $\mathcal{P}$, tal y como se explica en la Definición 31, con una interpretación pretendida $\mathcal{I}_0$, un objetivo admisible inicial $\hat{G}_0$ y una disyunción finita de respuestas computadas $D = \bigvee_{i \in I} \hat{S}_i$. Asumamos también que el cómputo ha sido realizado por un sistema de resolución de objetivos $CNPC(\mathcal{D})$-admisible. Entonces, existe algún símbolo de función definida f tal que el axioma $(f)^-_{\mathcal{P}}$ para f en $\mathcal{P}^-$ no es válido en $\mathcal{I}_0$. De este modo, la definición de f, tal y como viene dada en $\mathcal{P}$, es incompleta con respecto a $\mathcal{I}_0$.*

Demostración 23 *Debido a la $CNPC(\mathcal{D})$-admisibilidad del sistema de resolución de objetivos, podemos asumir que $\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} G_0 \Rightarrow D$ (con un cierto NPT). Más aún, aplicando el Teorema 15, se tiene que $\mathcal{P}^- \models_{\mathcal{D}} G_0 \Rightarrow D$. Por la Definición 30, podemos concluir que $Sol_{\mathcal{I}}(G_0) \subseteq Sol_{\mathcal{D}}(D)$ se cumple para cualquier modelo $\mathcal{I} \models_{\mathcal{D}} \mathcal{P}^-$. Sin embargo, también sabemos que $Sol_{\mathcal{I}_0}(G_0) \not\subseteq Sol_{\mathcal{D}}(D)$, debido a que la disyunción D de respuestas computadas es un síntoma de incompletitud con respecto a $\mathcal{I}_0$. Por tanto, podemos concluir que $\mathcal{I}_0 \not\models_{\mathcal{D}} \mathcal{P}^-$, y en consecuencia, que el axioma de completitud $(f)^-_{\mathcal{P}}$ de algún símbolo de función definida f debe ser inválido en $\mathcal{I}_0$.*

□

El teorema anterior no proporciona un método práctico para poder encontrar un símbolo de función definida cuyas reglas de programa sean incompletas. Como ya se ha explicado en la Sección 6.1, el método de diagnosis declarativa debe ser capaz de poder encontrar tales reglas de programa incompletas mediante la inspección y búsqueda en un *CT*. En esta subsección vamos a proponer el uso de árboles de prueba abreviados obtenidos mediante el cálculo $CNPC(\mathcal{D})$ como *CT*s.

De entre todas las reglas del cálculo $CNPC(\mathcal{D})$, la regla $\mathbf{(DF)}_f$ es la única regla de inferencia que depende del programa, mientras que todas las demás reglas de inferencia van a ser correctas con respecto a c-interpretaciones arbitrarias. Por esta razón, los árboles de prueba abreviados que vamos a utilizar como CTs omitirán los pasos de inferencia que estén relacionados con cualquier regla de inferencia del cálculo $CNPC(\mathcal{D})$ que sea distinta de la regla $\mathbf{(DF)}_f$. De manera más precisa, dado un NPT $\mathcal{T}$ que sirve de testigo a una $CNPC(\mathcal{D})$-derivación $\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} G \Rightarrow D$, su *Árbol de Prueba Abreviado Negativo* asociado $\mathcal{AT}$ (abreviadamente $ANPT$, del inglés *Abbreviated Negative Proof Tree*), se construye como sigue:

- La raíz de $\mathcal{AT}$ es la raíz de $\mathcal{T}$.
- Los hijos de cualquier nodo N en $\mathcal{AT}$ son los descendientes más cercanos de N en $\mathcal{T}$ correspondientes a *acas con caja*, introducidos por los pasos de inferencia $\mathbf{(DF)}_f$.

Como ejemplos concretos, el $ANPT$ de la Figura 6.2 se corresponde con el árbol abreviado construido a partir del NPT de la Figura 6.6, mientras que el $ANPT$ de la Figura 6.5 se corresponde con el árbol abreviado del NPT de la Figura 6.7.

Nuestro último resultado es un refinamiento del Teorema 17. Este resultado garantiza que la diagnosis declarativa con $ANPT$s usados como CTs conduce a la correcta detección de funciones de programa incompletas.

Teorema 18 (Diagnosis Declarativa de Respuestas Incompletas) *Como en el enunciado del Teorema 17, asumimos que un síntoma de incompletitud ha sido observado para un $CFLP(\mathcal{D})$-programa $\mathcal{P}$ tal y como se ha explicado en la Definición 31, con interpretación pretendida $\mathcal{I}_0$, objetivo admisible inicial $\hat{G}_0$ y una disyunción finita de respuestas $D = \bigvee_{i \in I} \hat{S}_i$ computada mediante un sistema de resolución de objetivos $CNPC(\mathcal{D})$-admisible. Entonces, es posible construir una derivación de la forma $\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} G_0 \Rightarrow D$ para la cual el $ANPT$ construido a partir de cualquier NPT que sirva de testigo a esta derivación tiene siempre algún nodo crítico. Más aún, cada uno de los nodos críticos apunta a un axioma $(f)^-_{\mathcal{P}}$ que es incompleto con respecto a la interpretación pretendida $\mathcal{I}_0$ del usuario.*

Demostración 24 *Comenzamos como en la demostración del Teorema 17: la admisibilidad del sistema de resolución de objetivos con respecto a $CNPC(\mathcal{D})$ garantiza una derivación de la forma $\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} G_0 \Rightarrow D$ con un cierto NPT, digamos $\mathcal{T}$. Sea $\mathcal{AT}$ el $ANPT$ construido a partir de $\mathcal{T}$. La raíz de $\mathcal{AT}$ es la misma que la de $\mathcal{T}$, y su aca asociado es inválido en la interpretación pretendida $\mathcal{I}_0$ debido a que $\mathcal{T}$ es un testigo de un síntoma de incompletitud. Razonando como en el Teorema*

14, podemos aplicar la **completitud débil de la depuración declarativa** *(véase [Nai97]) para demostrar que $\mathcal{AT}$ tiene al menos un nodo crítico N.*

Debido a la construcción de $\mathcal{AT}$ a partir de $\mathcal{T}$, N es también un nodo de $\mathcal{T}$ y los hijos de N en $\mathcal{AT}$ son los descendientes más cercanos de N en $\mathcal{T}$ correspondientes a acas con caja introducidos mediante la aplicación de $\mathbf{(DF)}_f$ pasos de inferencia. Por tanto, cada hijo de N en $\mathcal{T}$ puede ser inferido a partir de algunos de los hijos de N en $\mathcal{AT}$ por medio de reglas de inferencia del cálculo $CNPC(\mathcal{D})$ distintas de $\mathbf{(DF)}_f$. Tales inferencias son correctas con respecto a cualquier c-interpretación, como hemos visto en la demostración del Teorema 15. Por otra parte, puesto que N es un nodo crítico en $\mathcal{AT}$, el aca asociado a cada hijo de N en $\mathcal{AT}$ es válido en $\mathcal{I}_0$. A partir de toda esta información podemos concluir que N es también un nodo crítico en $\mathcal{T}$ y que la regla de $CNPC(\mathcal{D})$ que relaciona N con sus hijos en $\mathcal{T}$ debe ser $\mathbf{(DF)}_f$, para algún símbolo de función definido $f \in DF^n$ (cualquier otra $CNPC(\mathcal{D})$ regla de inferencia **(RI)** *es correcta con respecto a $\mathcal{I}_0$ y podría no inferir el aca no válido en N a partir de los acas válidos en los hijos de N). Más aún, N no puede ser el nodo raíz de $\mathcal{T}$, el cual no es nunca el resultado de una $\mathbf{(DF)}_f$ inferencia (esta afirmación se justifica por un razonamiento similar al empleado en la demostración del Teorema 14).*

Para concluir la demostración observamos que el paso $\mathbf{(DF)}_f$ que relaciona N con sus hijos en $\mathcal{T}$ infiere un aca, el cual es inválido en la interpretación pretendida $\mathcal{I}_0$ a partir de los acas que son válidos en $\mathcal{I}_0$. La demostración del Teorema 15 muestra que esta inferencia incorrecta podría no ser posible si el axioma de completitud $(f)^-_{\mathcal{P}}$ era válido en $\mathcal{I}_0$. Por tanto, este axioma $(f)^-_{\mathcal{P}}$ no puede ser válido en $\mathcal{I}_0$; en otras palabras, la función definida f asociada al nodo crítico N es incompleta con respecto a la interpretación pretendida, como pretendíamos demostrar. □

6.6. Un prototipo de herramienta para el diagnóstico declarativo de respuestas perdidas en $\mathcal{TOY}$

En esta última sección vamos a discutir las ventajas y los inconvenientes de una posible implementación en el sistema $\mathcal{TOY}$ de una herramienta de depuración declarativa de respuestas perdidas basada en el método de diagnosis presentado en las secciones precedentes. Actualmente, la implementación propuesta es un prototipo que solo permite manejar restricciones del dominio de Herbrand $\mathcal{H}$, aunque los mismos principios que han guiado su diseño pueden ser aplicados a otros dominios de restricciones. Al igual que en el caso de la herramienta de depuración presentada en el capítulo anterior, el desarrollo del prototipo actual ha sido el resultado de un trabajo en equipo, en el que la principal contribución del doctorando ha consistido en precisar la forma en la que se han implementado los nuevos árboles de cómputo así como las decisiones de diseño necesarias para representar adecuada-

mente la recolección de respuestas en la forma de *acas* en sus nodos, estableciendo y demostrando para ello suficientes resultados teóricos para justificar su integración en el sistema $\mathcal{TOY}$.

Básicamente, es posible seguir dos alternativas cuando se pretende implementar un depurador declarativo [NS97]:

(1) La primera alternativa consiste en usar una *transformación de programas* como la que se describe en trabajos relacionados para la depuración declarativa de valores incorrectos en *Haskell* [PN03a, PN03b, Pop07] y para la depuración declarativa de respuestas incorrectas en programación lógico funcional perezosa en $\mathcal{TOY}$ sin restricciones [CR02, CR04, Cab04] o con restricciones [Cab05]. El programa $\mathcal{P}$ que va a ser depurado es transformado previamente en un nuevo programa $\mathcal{P}^{\mathcal{T}}$. El cómputo que se realiza con respecto al programa transformado $\mathcal{P}^{\mathcal{T}}$ produce los mismos resultados que con respecto al programa de partida $\mathcal{P}$, pero gracias a la transformación es posible emparejar cada resultado con un árbol de cómputo que sirva de testigo para la depuración.

(2) La segunda alternativa se basa en *modificar el compilador* del lenguaje de programación utilizado, en la línea de trabajos previos como [Nil01b] sobre depuración declarativa de valores incorrectos en programación funcional. Después de la modificación, el compilador ha de ser capaz de producir los árboles de cómputo durante la ejecución del programa.

La primera de las dos alternativas planteadas es, sin duda, más independiente de la estructura del compilador, y es por tanto más portable. Sin embargo, la segunda alternativa se propone como un modo más eficiente de producir los CTs, los cuales son usualmente muy grandes y costosos en recursos del sistema. En cualquier caso, el programa (transformado) ha de ser re-ejecutado con el fin de poder obtener el correspondiente CT. En el desarrollo de nuestro prototipo hemos optado por la primera alternativa, aunque se han introducido algunas modificaciones con el fin de permitir reducir el coste en memoria del programa transformado y acelerar el proceso de construcción de los CTs.

En primer lugar, vamos a resumir el proceso que habitualmente se sigue en el sistema $\mathcal{TOY}$ cuando se compila un programa fuente $\mathcal{P}$*.toy* con el fin de resolver un objetivo inicial G con respecto a un programa $\mathcal{P}$. Durante el proceso de compilación normal (descrito en la parte *a*) de la Figura 6.8), el sistema traduce el programa fuente $\mathcal{P}$*.toy* en un programa Prolog equivalente $\mathcal{P}$*.pl* en el que se incluye un predicado para cada una de las funciones que se definen en $\mathcal{P}$ siguiendo la técnica que se describe en [LLR93] y en [Cab05]. Por ejemplo, la función `even` definida en la Sección 6.1 es transformada en el siguiente predicado:

```
even(N,R,IC,OC) :- ... código para 'even' ... .
```

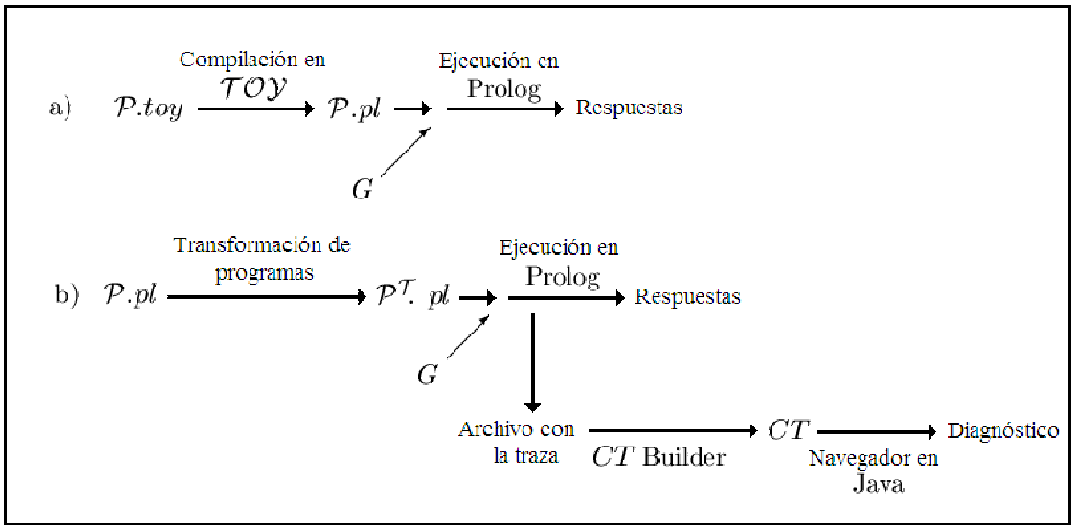


Figura 6.8: Esquema del prototipo de respuestas perdidas

donde la variable `N` corresponde al parámetro de entrada de la función, `R` al resultado de la función (obsérvese que el predicado correspondiente a cada función calcula resultados en *forma normal de cabeza* con el fin de poder implementar el estrechamiento perezoso, como se explica en el Apéndice B) e `IC,OC` representan, respectivamente, los almacenes de restricciones de entrada y de salida. Más aún, cada objetivo G de $\mathcal{P}$ se transforma también en un objetivo Prolog equivalente y es resuelto con respecto a $\mathcal{P}.pl$ mediante el sistema Prolog subyacente a $\mathcal{TOY}$. El resultado es una colección de respuestas que se le presentan al usuario en una cierta secuencia, como consecuencia del mecanismo de "vuelta atrás" (*backtracking*) de Prolog.

Si la computación de respuestas para G finaliza después de haber recogido una cantidad finita de respuestas, el usuario puede entonces decidir si hay o no respuestas perdidas (es decir, un *síntoma de incompletitud* en la terminología de la Definición 31), y en consecuencia ejecutar el comando `/missing` con el fin de iniciar una *sesión de depuración*. El depurador procede en este caso ejecutando los siguientes pasos:

1) El programa $\mathcal{P}.pl$ es transformado en un nuevo programa Prolog $\mathcal{P}^{\mathcal{T}}.pl$, como se muestra en la parte *b*) de la Figura 6.8. El depurador puede asumir de manera segura que $\mathcal{P}.pl$ es un programa ya existente en el sistema, puesto que la herramienta siempre se inicializa *después* de que alguna respuesta perdida haya sido detectada por el usuario. El programa transformado $\mathcal{P}^{\mathcal{T}}$ se comporta de manera idéntica a $\mathcal{P}$, con la única diferencia de que este programa permite producir una *traza* para representar la computación en un archivo de texto.

Esta traza está formada por la información de los valores de los argumentos y los contenidos del almacén de restricciones, y también por los valores de los mismos argumentos, el resultado y los contenidos del almacén de restricciones cuando la función tiene éxito. Por ejemplo, mostramos a continuación un fragmento del código que se obtendría para la función `even` en el programa transformado:

```
 1 % esta cláusula sustituye al predicado original
 2 even(N,R,IC,OC) :-
 3     % muestra los valores de entrada para 'even'
 4     write(' begin('), write(' even,'), writeq(N), write(','),
 5     write(R), write(', '), writeq(IC), write(').'), nl,
 6     % 'evenBis' corresponde al predicado original para 'even'
 7     evenBis(N,R,IC,OC),
 8     % muestra un resultado de salida
 9     write(' output('), write(' even,'), writeq(N), write(','),
10     write(R), write(', '), writeq(OC), write(').'), nl.
11 % cuando todas las posibles salidas han sido producidas
12 even(N,R,IC,OC) :-
13          nl, write(' end(even).'), nl,
14          !,
15          fail.
16 evenBis(N,R,IC,OC) :- ... código original para 'even' ... .
```

Como se observa en el ejemplo, el código para cada función permite mostrar información concreta sobre los valores de los argumentos y de los contenidos del almacén de restricciones en cada momento en el que se haga uso de una función definida (líneas 4-5). Entonces, el predicado correspondiente a la función original, ahora renombrado con el sufijo `Bis`, es invocado (línea 7). Después de cada llamada con éxito a una función, la traza muestra de nuevo los valores de los argumentos y el resultado, el cuál podría haber cambiado, y los contenidos del almacén de restricciones de salida (líneas 9, 10). Una segunda cláusula (líneas 12-15) muestra el valor `end` cuando la función ha agotado todas sus posibles salidas. La cláusula falla con el fin de poder asegurar que el flujo de programa no ha sido modificado. El código original para cada función se guarda sin modificaciones en el programa transformado, excepto el renombramiento (`evenBis` en lugar de `even` en la línea 16 del ejemplo). Esto permite asegurar que el programa se comportará de modo equivalente al programa original, excepto para la traza que ha sido producida como un efecto lateral.

2) Con el propósito de obtener el archivo con la traza, el depurador ha de repetir el cómputo de todas las respuestas para el objetivo G con respecto a $\mathcal{P}^{\mathcal{T}}$. Tras cada cómputo finalizado con éxito, el depurador fuerza un `fail` con el objetivo de activar el mecanismo de "vuelta atrás" y producir la siguiente solución para

el objetivo. La salida del programa es entonces redirigida a un archivo, donde la traza está siendo almacenada.

3) El archivo con la traza es entonces analizado por el módulo *CT builder* de la herramienta. El resultado es el *árbol de cómputo* (un *ANPT*), el cuál se muestra mediante una *interfaz gráfica* implementada en Java.

4) El árbol puede ser recorrido por el usuario, bien manualmente, proporcionando información sobre la validez de los *aca*s contenidos en el árbol, o bien usando alguna de las estrategias semi-automáticas que están incluidas en la herramienta, las cuales tratan de minimizar el número de nodos que el usuario ha de examinar (para una descripción detallada de algunas de estas estrategias y su eficiencia, el lector interesado puede consultar [Sil06]). El proceso acaba cuando se encuentra un *nodo crítico* y la herramienta indica que una definición incompleta de función ha sido localizada en el programa.

La implementación actual del prototipo en el sistema $\mathcal{TOY}$ se encuentra disponible en la dirección de internet `http://gpd.sip.ucm.es/rafa/missing`. La generación de los archivos con la traza funciona satisfactoriamente, aunque el módulo de construcción de árboles de cómputo *CT builder* y la interfaz gráfica en Java todavía necesitan mejoras.

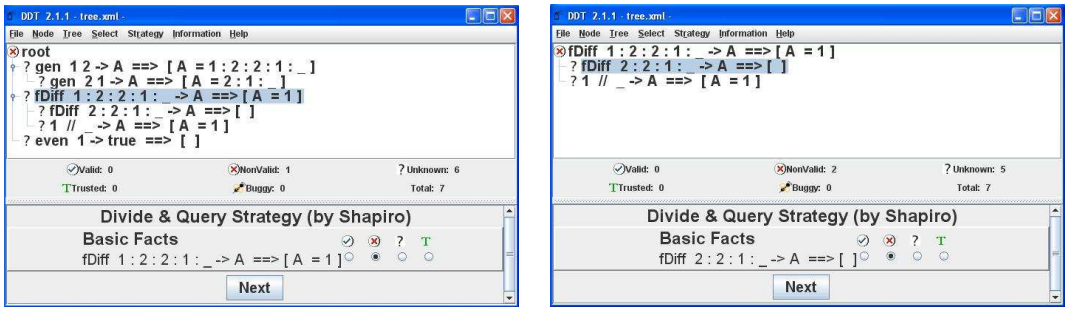


Figura 6.9: Capturas del prototipo de depurador de respuestas perdidas

La Figura 6.9 muestra la forma en la que la herramienta representa el *CT* que corresponde al escenario de depuración declarativa de respuestas perdidas descrito en la Sección 6.1. Aunque el objetivo inicial no se muestra, el resto del *CT* se correspondería con el de la Figura 6.5, cuya construcción como un *ANPT* ha sido explicada en las secciones precedentes. Cuando se muestra un *aca* de la forma $f\,\bar{t}_n \rightarrow t \,\square\, S \Rightarrow \bigvee_{i\in I} \hat{S}_i$, la herramienta usa notación de listas para representar la disyunción finita $\bigvee_{i\in I} \hat{S}_i$, además de realizar las siguientes simplificaciones:

- Los vínculos de variables inútiles dentro de los almacenes S y S_i son eliminados, como ocurre en el caso particular del *aca* que se muestra en la Figura 6.9 en la forma `gen 2 1 -> A ==> [A = 2:1:_]`.

- Si t es una variable X, el caso $\{X \mapsto \bot\}$ se omite de la disyunción finita $\bigvee_{i \in I} \hat{S}_i$, de modo que el usuario ha de interpretar el *aca* como la colección de todos los posibles resultados para X que sean distintos del valor indefinido $\bot$.

- La herramienta también muestra el símbolo especial `_` en algunos lugares. Dentro de un *aca* cualquiera, las apariciones de `_` en el lado derecho de la implicación $\Rightarrow$ han de ser entendidas como variables diferentes cuantificadas existencialmente, mientras que cada aparición de `_` en el lado izquierdo de $\Rightarrow$ debe ser interpretada como $\bot$. Por ejemplo, `1 // _ -> A ==> [A = 1]` es el *aca* `1 //` $\bot \rightarrow A \Rightarrow \{A \mapsto 1\}$ tal y como lo mostraría la herramienta. Interpretar las apariciones de `_` en el lado izquierdo de $\Rightarrow$ como variables diferentes cuantificadas universalmente podría ser incorrecto. En particular, el *aca* `1 //` $\bot \rightarrow A \Rightarrow \{A \mapsto 1\}$ es válido con respecto a la interpretación pretendida $\mathcal{I}_{\mathcal{P}_{\mathtt{fD}}}$ de $\mathcal{P}_{\mathtt{fD}}$, mientras que la sentencia $\forall X.$ (`1 //` $X \rightarrow A \Rightarrow \{A \mapsto 1\}$) tiene un significado diferente y no es válida en $\mathcal{I}_{\mathcal{P}_{\mathtt{fD}}}$.

En la sesión de depuración que se muestra en la Figura 6.9, el usuario ha seleccionado la estrategia *pregunta y divide* (*Divide & Query*) [Sil06] con el propósito de encontrar un nodo crítico. La parte inferior de la pantalla mostrada en el lado izquierdo de la Figura 6.9 muestra la primera de las preguntas que la herramienta plantea al usuario tras haber seleccionado esta estrategia, es decir, la validez del *aca* `fDiff 1:2:2:1:_ -> A ==>[A=1]`. De acuerdo con el conocimiento que el usuario tiene de la interpretación pretendida $\mathcal{I}_{\mathcal{P}_{\mathtt{fD}}}$ del programa, marca este *aca* como inválido. La estrategia ahora permite podar el CT guardando solo el subárbol con raíz el *aca* inválido del paso anterior (recordemos que cada CT con una raíz inválida debe contener al menos un nodo crítico). La segunda cuestión, mostrada en la pantalla del lado derecho de la Figura 6.9, pregunta al usuario por la validez del *aca* `fDiff 2:2:1:_ -> A ==>[]` (el cual representa `fDiff 2:2:1:`$\bot \rightarrow A \Rightarrow \{A \mapsto \bot\}$, como se ha explicado anteriormente). De nuevo, el conocimiento de $\mathcal{I}_{\mathcal{P}_{\mathtt{fD}}}$ conduce al usuario a esperar que la llamada `fDiff 2:2:1:`$\bot$ pueda devolver algún resultado definido, por lo que el *aca* es marcado como inválido. Tras responder a esta cuestión, el depurador señala a `fDiff` como una función incompleta, finalizando así la sesión de depuración de respuestas perdidas.

El siguiente ejemplo sencillo permite complementar la presentación del comportamiento del prototipo implementado en $\mathcal{TOY}$, aunque su finalidad no sea exactamente la de llegar al diagnóstico de ninguna función incompleta como ocurría en el ejemplo anterior. La Figura 6.10 muestra el CT correspondiente al objetivo `sonOf mary == A` para el fragmento de programa sobre relaciones de familia presentado y discutido en la Sección 6.1. El nodo raíz del $ANPT$ correspondiente a este CT se

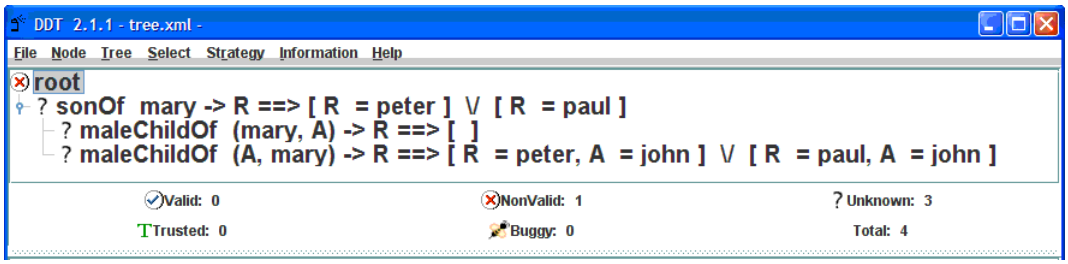


Figura 6.10: Otra captura del prototipo de depurador de respuestas perdidas

asume que es un síntoma de incompletitud (ya que el depurador ha sido invocado), y por tanto, no se muestra por pantalla. El *aca* que aparece en la raíz de la parte mostrada en la Figura 6.10 proporciona la información de que `mary` tiene dos hijos, `peter` y `paul`. Los dos hijos del nodo raíz están asociados a dos llamadas de función diferentes para `maleChildOf`, realizadas a través de la función `sonOf`:

- El primer nodo hijo corresponde al caso en el que `mary` es el padre, y la lista vacía en el lado derecho representa una disyunción vacía de respuestas computadas, puesto que el intento de probar que `mary` es el padre de alguien mediante la aplicación de las definiciones de función en el programa falla finitamente.

- El segundo nodo hijo corresponde al caso en el que `mary` es la madre. Este *aca* da lugar a dos respuestas, cada una de ellas incluyendo el nombre del padre (`john` en ambos casos) así como también el nombre de un hijo (`peter` y `paul`, respectivamente). Estos nombres se muestran como vínculos de variables.

Este cómputo particular no ha involucrado ningún tipo de resolución de restricciones. En otro caso, las restricciones apropiadas se mostrarían dentro de las respuestas de los *aca*s además de los vínculos de variables correspondientes de una manera semejante a como se ha mostrado en las secciones precedentes.

Finalmente, en relación a la eficiencia de este método de depuración, los experimentos preliminares que hemos efectuado permiten evidenciar las siguientes conclusiones:

1) Generar el programa transformado $\mathcal{P}^{\mathcal{T}}$. *pl* a partir de $\mathcal{P}$.*pl* es proporcional en tiempo al número de funciones en el programa. Esta operación requiere una cantidad insignificante de memoria del sistema, puesto que cada predicado es transformado de manera separada.

2) El cómputo del objetivo con respecto a $\mathcal{P}^{\mathcal{T}}$. *pl* requiere a lo sumo los mismos recursos del sistema que con respecto al programa $\mathcal{P}.pl$, ya que escribir la traza no da lugar a una sobrecarga significativa en nuestros experimentos.

3) Producir el CT a partir de la traza no es inmediato y requiere varios recorridos de la traza. Aunque esta dificultad algorítmica implica un mayor tiempo de consumo de recursos, este proceso sólo guarda porciones de la traza en la memoria del sistema en cada momento.

4) La fase más ineficiente en nuestra implementación actual es, sin duda, la que se correspondería con la interfaz gráfica. Aunque sea posible guardar en memoria sólo la porción del árbol que se muestra en cada momento, nuestra interfaz gráfica debe cargar el CT al completo en memoria principal. Planeamos mejorar en un futuro próximo esta limitación. Sin embargo, el prototipo actual permite trabajar con CTs que contienen miles de nodos, lo que es suficiente para permitir la depuración de errores experimentando con cómputos de tamaño mediano.

5) Como es usual en depuración declarativa, la eficiencia de la herramienta depende directamente del tamaño del árbol de cómputo, el cual depende a su vez del tamaño de las estructuras de datos requeridas y no del tamaño del programa. Esto también nos permite concluir la conveniencia de utilizar datos de tamaño pequeño o mediano para los cómputos planteados con la intención de depurar errores.

Una cuestión diferente es la dificultad de responder a las preguntas que se le formulan al usuario. De hecho, en programas complejos que involucran restricciones, los *aca*s pueden ser grandes e intrincados, como también ocurre con otras herramientas de depuración para lenguajes *CLP* [TF00]. No obstante, nuestro prototipo trabaja razonablemente bien en todos aquellos casos en los que el espacio de búsqueda del objetivo es de tamaño pequeño o mediano, por lo que creemos que trabajar con nuestra herramienta sobre estos objetivos puede ser de utilidad a la hora de detectar muchos de los errores de programación que tienen lugar en la práctica. Por supuesto, el diseño de técnicas específicas que nos permitan simplificar CTs es uno de los trabajos futuros más importantes a desarrollar sobre nuestro prototipo actual para favorecer la aplicabilidad real de la herramienta y su uso en otros entornos de programación. Por ejemplo, pedir al usuario una instancia concreta de respuesta perdida del objetivo inicial que está depurando y comenzar la sesión de diagnosis para esa instancia del objetivo podría ser de gran utilidad.

Capítulo 7

Conclusiones y trabajo futuro

En este último capítulo resumimos cuáles han sido los principales resultados que se han obtenido en la tesis, tanto a nivel teórico como con respecto a su aplicación práctica. También planteamos algunas de las posibilidades más interesantes de trabajo futuro que surgen a partir de los resultados obtenidos.

7.1. Conclusiones

En esta memoria de tesis doctoral hemos presentado un marco teórico que permite caracterizar la semántica declarativa y operacional de los lenguajes de programación lógico funcionales perezosos con restricciones, a través del desarrollo de un nuevo esquema genérico $CFLP(\mathcal{D})$ sobre un dominio de restricciones $\mathcal{D}$ y un resolutor sobre ese dominio paramétricamente dado. Como aplicación de este esquema se ha conseguido formalizar un método de diagnosis de programas con restricciones que integra, de una forma cómoda y natural, aproximaciones previas que en el campo de la depuración declarativa se han desarrollado por separado para los paradigmas lógico funcional y lógico con restricciones.

7.1.1. Resultados teóricos

Resumimos a continuación, de una manera más detallada, cuáles han sido los principales resultados teóricos en los que se fundamentan cada una de las dos partes en las que se divide la presente memoria.

El esquema $CFLP(\mathcal{D})$ de programación lógico funcional con restricciones

El nuevo esquema genérico $CFLP(\mathcal{D})$ propuesto en la primera parte de esta tesis nos ha permitido fundamentar, de una manera uniforme, la semántica de programas lógico funcionales con restricciones. Para lograr este objetivo se ha proporcionado una nueva formalización del concepto de dominio de restricciones, una nueva noción

de c-interpretación que ha dado lugar a una semántica débil y a una semántica fuerte para programas, y una nueva lógica para la reescritura con restricciones $CRWL(\mathcal{D})$, también parametrizada por el dominio $\mathcal{D}$, para la que ha sido posible demostrar su corrección y completitud con respecto a ambos tipos de semántica.

Todos estos resultados pueden verse como una extensión natural, aunque no trivial, de resultados semánticos ya conocidos en el esquema $CLP(\mathcal{D})$ para la programación lógica con restricciones [JMMS98, GDL95] y para la programación lógico funcional perezosa sin restricciones basada en la lógica para la reescritura *CRWL* [GHLR96, GHLR99]. En comparación con otros trabajos anteriores y aproximaciones previas en programación lógico funcional con restricciones, como por ejemplo el esquema $CFLP(\mathcal{D})$ desarrollado en [Lop92, Lop94], se han conseguido resultados teóricos destacables por su importancia teórica, especialmente en lo referente a mejorar el planteamiento formal de la semántica declarativa para *CFLP*. En este sentido, las mejoras introducidas por el nuevo esquema $CFLP(\mathcal{D})$ proporcionan un fundamento muy satisfactorio para todos aquellos trabajos previamente desarrollados sobre programación lógico funcional con restricciones de desigualdad [KLMR92, AGL94, LS99a] y un punto de partida sólido para posibles mejoras de los resultados obtenidos en trabajos anteriores sobre programación lógico funcional con restricciones sobre multiconjuntos [ALR98, ALR99]. Las restricciones sobre multiconjuntos están fuera del alcance del presente trabajo debido a su utilización de constructoras de datos algebraicas, mientras que el esquema $CFLP(\mathcal{D})$ que hemos presentado aquí asume constructoras de datos libres.

En segundo lugar, hemos demostrado la existencia de varios cálculos teóricos de resolución de objetivos que son correctos y fuertemente completos con respecto a la semántica declarativa de $CFLP(\mathcal{D})$-programas proporcionada por la lógica para la reescritura con restricciones $CRWL(\mathcal{D})$. Algunos de estos cálculos pueden ser considerados incluso como un modelo formal razonable para la especificación del comportamiento actual de implementaciones de sistemas $CFLP(\mathcal{D})$. En concreto, hemos presentado un nuevo cálculo de estrechamiento perezoso con restricciones $CLNC(\mathcal{D})$ que puede parametrizarse por un dominio de restricciones $\mathcal{D}$ arbitrario. Este cálculo se apoya en una nueva noción formal de resolutor de restricciones, mediante la cual es posible garantizar las propiedades de corrección y completitud fuerte. En segundo lugar, hemos presentado un nuevo cálculo de estrechamiento dirigido por demanda con restricciones $CDNC(\mathcal{D})$, el cual también puede ser parametrizado por un dominio de restricciones arbitrario $\mathcal{D}$ y por un resolutor sobre el dominio, pero que además incorpora árboles definicionales con restricciones para poder guiar la elección de redexes demandados (y por tanto necesarios) en el cómputo. Este segundo cálculo da lugar a mejoras de eficiencia con respecto al primer cálculo $CLNC(\mathcal{D})$, puesto que además de conservar las propiedades de corrección y de completitud fuerte permite mantener la eficiencia de otras estrategias de estrechamiento necesario en programación lógico funcional para las que se ha demostrado su optimalidad

gracias a la utilización de árboles definicionales. Todas estas propiedades sin duda convierten al cálculo $CDNC(\mathcal{D})$ en una adecuada especificación del comportamiento computacional en sistemas $CFLP(\mathcal{D})$ existentes como por ejemplo $Curry$ [Han06] y $\mathcal{TOY}$ [ALR07, FHS03c].

Técnicas de depuración declarativa en el esquema $CFLP(\mathcal{D})$

En la segunda parte de esta memoria de tesis hemos presentado los fundamentos teóricos de dos métodos de depuración declarativa en el esquema $CFLP(\mathcal{D})$, el primero de ellos para el diagnóstico de respuestas incorrectas y el segundo para la diagnosis de respuestas incompletas. Ambos métodos se apoyan en la técnica clásica de representación de la computación que ha dado lugar al error mediante un árbol de cómputo cuya inspección conduce al descubrimiento de alguna regla de programa responsable de tal error. Hemos demostrado la corrección lógica de ambos métodos gracias a la conexión que se establece entre los árboles de cómputo, construidos como árboles de prueba mediante cálculos de inferencia adecuados para la representación de respuestas incorrectas y perdidas (el cálculo de prueba positivo y el cálculo de prueba negativo, respectivamente) y la semántica de los $CFLP(\mathcal{D})$-programas, formalizada mediante la lógica para la reescritura con restricciones $CRWL(\mathcal{D})$. Hemos demostrado también la existencia de cálculos de resolución de objetivos a partir de los cuales se pueden construir los árboles de cómputo necesarios para los dos métodos de depuración propuestos.

Creemos que, hasta el momento, ningún otro resultado teórico comparable había sido previamente demostrado para un marco formal tan expresivo como el que ofrece la programación lógico funcional con restricciones a través del nuevo esquema genérico $CFLP(\mathcal{D})$ propuesto en la primera parte de esta memoria de tesis doctoral.

7.1.2. Aplicación práctica

Una de las expectativas más importantes que suscita la propuesta teórica del nuevo esquema $CFLP(\mathcal{D})$, así como de las técnicas de depuración declarativas asociadas que han sido presentadas en esta memoria, es la de determinar si realmente se traducen en algún tipo de aplicación práctica implementable en sistemas reales de programación. En esta sección vamos a tratar de resumir cómo se han llevado a la práctica todas estas ideas teóricas, tanto en lo que se refiere a la implementación en el sistema $\mathcal{TOY}$ de instancias concretas del esquema $CFLP(\mathcal{D})$ como al diseño de herramientas de depuración declarativa de respuestas incorrectas y de respuestas perdidas.

El sistema $\mathcal{TOY}(\mathcal{FD})$

En primer lugar, y en lo que respecta a la aplicación práctica del esquema $CFLP(\mathcal{D})$, se ha desarrollado una implementación en el sistema $\mathcal{TOY}$ siguiendo las ideas teóri-

cas presentadas en los Capítulos 2, 3 y 4, que permite trabajar con la instancia concreta $CFLP(\mathcal{FD})$ correspondiente al dominio finito $\mathcal{FD}$. Las características más destacables de esta implementación, a la que hemos denominado $\mathcal{TOY}(\mathcal{FD})$, son las que se recogen tanto en el Apéndice B de esta memoria como en nuestra publicación de revista [FHSV07]. En resumen, y gracias a $\mathcal{TOY}(\mathcal{FD})$, se ha conseguido proporcionar un fundamento tanto teórico como práctico a trabajos anteriormente iniciados en [FHS03a, FHS03b] sobre programación lógico funcional con restricciones de dominio finito. Vamos a detallar a continuación algunas de las características principales y de los rasgos más significativos en los que se basa el sistema de programación $\mathcal{TOY}(\mathcal{FD})$:

- La instancia $CFLP(\mathcal{FD})$ permite definir una aproximación lógico funcional a la resolución de restricciones en el dominio $\mathcal{FD}$. De esta forma, el lenguaje $CFLP(\mathcal{FD})$ no es sólo una alternativa a $CLP(\mathcal{FD})$ sino que consigue extender sus principales capacidades con nuevas características no habituales o no existentes en $CLP(\mathcal{FD})$, como por ejemplo, la notación funcional y currificada, la utilización de un sistema de tipos, funciones de orden superior (y por tanto, restricciones de orden superior), composición de restricciones, patrones de orden superior, evaluación perezosa y polimorfismo, entre otras. Como consecuencia, $CFLP(\mathcal{FD})$ proporciona mayores recursos, cuando se compara con $CLP(\mathcal{FD})$, para una programación declarativa productiva que implícitamente permite una mayor expresividad. En el Apéndice B se han explicado cuáles son las contribuciones más importantes de $\mathcal{TOY}(\mathcal{FD})$ a través de las capacidades adicionales que $CFLP(\mathcal{FD})$ ofrece con respecto a $CLP(\mathcal{FD})$. Mediante esta comparativa ha sido posible evidenciar los principales beneficios a los que da lugar la integración de FLP y de $CLP(\mathcal{FD})$ en un solo lenguaje declarativo de programación.

- El sistema $\mathcal{TOY}(\mathcal{FD})$ permite conectar el sistema lógico funcional perezoso $\mathcal{TOY}$ con el eficiente resolutor de restricciones $\mathcal{FD}$ de SICStus Prolog [SIC03]. Como resultado, $\mathcal{TOY}(\mathcal{FD})$ es un sistema lógico funcional perezoso que soporta la resolución de restricciones $\mathcal{FD}$. Es más, hemos demostrado que la resolución de restricciones en $\mathcal{TOY}(\mathcal{FD})$ es suficientemente eficiente, puesto que, en general, se comporta de una manera muy cercana a SICStus Prolog, lo que significa que la sobrecarga de SICStus Prolog en $\mathcal{TOY}$ no incrementa significativamente el tiempo de cómputo real en el sistema $\mathcal{TOY}(\mathcal{FD})$. Como consecuencia, $\mathcal{TOY}(\mathcal{FD})$ supera claramente a otros sistemas $CLP(\mathcal{FD})$ existentes, como *SWI Prolog* y *Ciao Prolog*, y es también competitivo con respecto a *GNU Prolog*, uno de los sistemas $CLP(\mathcal{FD})$ más eficientes. Los resultados experimentales obtenidos en [FHSV07] han conseguido probar incluso que $\mathcal{TOY}(\mathcal{FD})$ es entre uno y tres veces más rápido que *PAKCS*, la implementación de $CFLP(\mathcal{FD})$ más próxima a $\mathcal{TOY}(\mathcal{FD})$ en *Curry*.

- Otras aplicaciones prácticas de la resolución de problemas que involucran restricciones $\mathcal{FD}$ en el sistema $\mathcal{TOY}(\mathcal{FD})$ pueden consultarse en [FHS02] y en [FHS03a]. En todas estas aplicaciones queda suficientemente claro que una de las ventajas inherentes a la implementación que ofrece $\mathcal{TOY}(\mathcal{FD})$ de $CFLP(\mathcal{FD})$ es la de permitir resolver, por un lado, todas aquellas aplicaciones prácticas de $CLP(\mathcal{FD})$, y por otro, permitir resolver problemas que guardan mayor relación con el contexto de la programación funcional. En este sentido, podemos afirmar que la integración de restricciones $\mathcal{FD}$ en un lenguaje FLP permite incorporar los principales beneficios de ambos mundos.

- Adicionalmente, también es posible sacar en conclusión que la idea de utilizar un lenguaje FLP como interfaz de un resolutor de restricciones puede ser extendida a otros tipos de sistemas de restricciones de interés, tales como aquellos que resuelven restricciones no lineales, restricciones sobre conjuntos, o restricciones booleanas, por nombrar tan solo unos pocos. Se observa así que $\mathcal{TOY}(\mathcal{FD})$ puede ser visto como un procedimiento de resolución de restricciones integrado en un sofisticado mecanismo de ejecución de estrechamiento perezoso que permite resolver no solo restricciones primitivas sino también restricciones definidas por el usuario. Operacionalmente hablando podríamos decir que $\mathcal{TOY}(\mathcal{FD})$ permite compilar $CFLP(\mathcal{FD})$-programas en programas Prolog en un sistema equipado con un resolutor de restricciones $\mathcal{FD}$. Esto hace que tanto la evaluación perezosa como la resolución de restricciones sean características inherentes al propio sistema $\mathcal{TOY}(\mathcal{FD})$.

Herramientas de depuración declarativa en el sistema $\mathcal{TOY}$

En la segunda parte de esta tesis hemos presentado una herramienta de depuración declarativa de respuestas incorrectas en el sistema $\mathcal{TOY}$, denominada $\mathcal{DDT}$, mediante la cual hemos conseguido implementar la técnica propuesta de diagnosis en el dominio concreto $\mathcal{R}$ de restricciones aritméticas sobre los números reales. Esta herramienta ha sido diseñada como una extensión no trivial de herramientas de depuración previamente existentes. La principal ventaja de la herramienta $\mathcal{DDT}$ es que proporciona diversas facilidades prácticas para reducir el número y la complejidad de las preguntas que se le presentan al usuario durante una sesión de depuración.

Con respecto al método propuesto de depuración declarativa de respuestas perdidas, desde un punto de vista práctico, la técnica puede ser aplicada a sistemas $CFLP$ actuales como *Curry* o $\mathcal{TOY}$, dando lugar a un diagnóstico correcto, siempre bajo la suposición de que estos se comportan como sistemas admisibles de resolución de objetivos. Esta suposición es plausible en tanto que estos sistemas están basados en los procedimientos formales de resolución de objetivos que hemos demostrado como admisibles en el Capítulo 6. Hasta el momento hemos conseguido realizar una implementación en el sistema $\mathcal{TOY}$ (aún en fase de desarrollo) de un prototipo de

depurador basado en el método de depuración declarativa de respuestas perdidas presentado en el Capítulo 6 y en [CRV08], el cuál nos permite trabajar con restricciones del dominio de Herbrand $\mathcal{H}$. Pensamos que, con los mismos principios de diseño, sería también posible aplicar esta herramienta sobre otros dominios de restricciones de interés práctico, como por ejemplo $\mathcal{R}$ y $\mathcal{FD}$.

7.2. Trabajo futuro

El trabajo desarrollado en esta memoria de tesis deja aún mucho espacio para futuras investigaciones, tanto desde el punto de vista teórico como de su utilidad práctica. Entre las posibilidades de trabajo futuro que tienen que ver más con el desarrollo teórico del nuevo esquema $CFLP(\mathcal{D})$ destacamos la *cooperación de resolutores* y la *verificación de programas declarativos multi-paradigma con restricciones*. Desde el punto de vista más práctico de la diagnosis declarativa de $CFLP(\mathcal{D})$-programas proponemos la mejora de las herramientas actuales de depuración implementadas en el sistema $\mathcal{TOY}$, especialmente a través de la utilización de *aserciones*.

Seguidamente detallamos en mayor profundidad algunas de las líneas de investigación relacionadas con el trabajo de esta tesis doctoral, en cuyo desarrollo participamos junto con otros miembros del Grupo de Programación Declarativa del Departamento de Sistemas Informáticos y Computación. De algunas de ellas éramos conscientes al inicio del presente trabajo, mientras que otras han ido surgiendo durante el desarrollo del mismo.

7.2.1. Cooperación de resolutores en el esquema $CFLP(\mathcal{D})$

El esquema $CFLP(\mathcal{D})$ para programación lógico funcional con restricciones que se ha presentado en esta memoria de tesis continúa una larga historia de intentos por combinar el poder expresivo de la programación lógica y de la programación funcional con las mejoras en ejecución proporcionadas por la utilización de resolutores de restricciones específicos. Así como ocurre con el esquema *CLP* [JM94] para programación lógica con restricciones, hemos definido varias instancias concretas del esquema $CFLP(\mathcal{D})$ que se corresponden con diferentes dominios de restricciones $\mathcal{D}$ que podemos usar como parámetros específicos. Al margen de todos los beneficios que la utilización de este esquema conlleva y que ya han sido extensamente comentados, el uso de un dominio fijo $\mathcal{D}$ en cada momento supone una limitación importante, puesto que muchos problemas prácticos involucran restricciones pertenecientes a más de un dominio.

Por supuesto, una posible solución consistiría en diseñar resolutores "híbridos" que permitiesen trabajar con este tipo de restricciones definidas sobre primitivas pertenecientes a más de un dominio. Ahora bien, el desarrollo eficiente de tales resolutores no es una tarea sencilla, y en muchos casos constituye una labor muy

costosa, sino imposible, si se pretenden utilizar de una manera competitiva en la práctica real de la programación. Una solución práctica de este problema en el contexto de la programación lógica con restricciones y el esquema genérico *CLP* es la que se basa en la utilización del concepto de *cooperación de resolutores* [GMB01], el cual ha dado lugar a un campo de investigación que en los últimos años ha motivado un interés creciente en la comunidad de la programación con restricciones y que ha sido ampliamente estudiado. En líneas generales, la cooperación de resolutores propone superar, básicamente, dos problemas:

1) La falta de declaratividad en las soluciones, es decir, la interacción entre resolutores debería hacer más sencillo el que sea posible expresar problemas compuestos por restricciones pertenecientes a más de un dominio.

2) Un pobre rendimiento de los sistemas, esto es, la comunicación entre resolutores debería poder mejorar la eficiencia del proceso de resolución de restricciones.

El objetivo principal de la cooperación de diferentes resolutores en Programación con Restricciones es, por tanto, el de poder resolver de una forma eficiente problemas híbridos que no pueden ser resueltos mediante un solo resolutor de restricciones sobre un dominio particular. En el contexto del esquema $CFLP(\mathcal{D})$, las principales instancias que se han propuesto en esta tesis y que han sido implementadas en el sistema $\mathcal{TOY}$ [ALR07], permiten hacer uso de los siguientes resolutores particulares: un resolutor para el dominio de Herbrand $\mathcal{H}$, con restricciones de igualdad y desigualdad; un resolutor para el dominio $\mathcal{FD}$, el cual soporta restricciones de dominio finito sobre el conjunto de los números enteros $\mathbb{Z}$; y un resolutor para el dominio $\mathcal{R}$, el cual soporta restricciones aritméticas sobre el conjunto de los números reales $\mathbb{R}$. En el contexto de la cooperación de resolutores en el esquema $CFLP(\mathcal{D})$, un problema interesante se centraría en la combinación de estos tres resolutores. Esta combinación permitiría utilizar, por una parte, las restricciones de $\mathcal{H}$ para tratar información estructurada, y por otra, las restricciones híbridas de $\mathcal{FD}$ y $\mathcal{R}$ que aparecen en múltiples aplicaciones prácticas en programación con restricciones.

En los trabajos [EFHR+07a, EFHR+07b, EFHR+08] hemos presentado una propuesta para la cooperación de resolutores basada en la comunicación entre los dominios $\mathcal{H}$, $\mathcal{FD}$ y $\mathcal{R}$ mediante la utilización de restricciones especiales de comunicación denominadas *puentes*. Un puente, expresado en la forma `X #== Y`, permite restringir una variable de tipo entero `X :: int` y una variable de tipo real `Y :: real` a tomar el mismo valor entero.

Ejemplo 22 (Cooperación entre $\mathcal{FD}$ y $\mathcal{R}$) *Como ejemplo particular, consideramos un programa genérico escrito en $\mathcal{TOY}$ que permite resolver el problema de la búsqueda de un punto bidimensional en la* `interseccion` *entre una región discreta de puntos* `cuadrado` *de dimensión* `N` $\times$ `N` *y una región continua* `triangulo` *de vértice superior* `(RX0,RY0)`*, base* `B` *y altura* `H` *similares a las ya representadas en el*

Capítulo 2 a través del Ejemplo 1. Tanto las dos regiones como la forma en la que se determina la intersección entre ambas a través de la función booleana `pertenece`*, se representan como predicados que involucran restricciones de $\mathcal{FD}$ y $\mathcal{R}$, respectivamente. La Figura 7.1 muestra tres objetivos diferentes para este programa.*

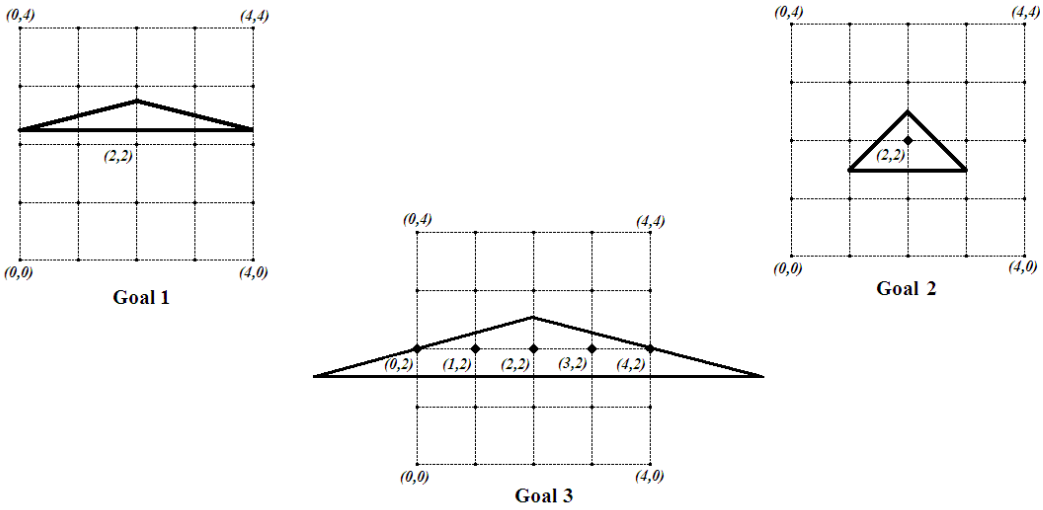


Figura 7.1: Intersección entre una rejilla cuadrada y tres regiones triangulares

```
% Región discreta de puntos en forma de rejilla cuadrada

  cuadrado :: int -> grid
  cuadrado N (X,Y) :- domain [X,Y] 0 N

% Región continua en forma triangular

  triangulo :: cPoint -> real -> real -> region
  triangulo (RX0,RY0) B H (RX,RY) :-
                      RY >= RY0 - H,
                      B * RY - 2 * H * RX <= B * RY0 - 2 * H * RX0,
                      B * RY + 2 * H * RX <= B * RY0 + 2 * H * RX0

  interseccion :: region -> rejilla -> dPoint -> bool
  interseccion Region Rejilla (X,Y) :- X #== RX, Y #== RY,
                                       pertenece Region  (RX,RY),
                                       pertenece Rejilla (X,Y),
                                       labeling  [ ] [X,Y]
```

El sistema permite guardar los puentes en un almacén especial y los utiliza para dos propósitos, denominados, respectivamente, *vinculación* y *propagación*:

- La *vinculación* es una operación muy simple que permite vincular una variable que aparece en uno de los extremos de un puente a un valor numérico adecuado, siempre que el otro extremo se haya conseguido instanciar a un valor numérico.

- La *propagación* es una operación más compleja que tiene lugar siempre que una restricción de un dominio "puro" es enviada al resolutor de $\mathcal{FD}$ o de $\mathcal{R}$. En ese momento, las reglas de propagación utilizan los puentes que están disponibles para construir una restricción en el dominio emparejado con el de partida sobre el que vamos a proyectar. En nuestro caso concreto, $\mathcal{R}$ es el dominio emparejado con $\mathcal{FD}$ y viceversa. Esta nueva restricción proyectada es enviada al resolutor del dominio correspondiente para que pueda así contribuir a la resolución global del problema híbrido inicialmente planteado.

En resumen, la propagación permite a cada uno de los dos resolutores, el de $\mathcal{FD}$ y el de $\mathcal{R}$, tomar ventaja y partido de la computación realizada por el otro. Con el fin de maximizar las oportunidades para la propagación, los procedimientos de resolución de objetivos que han sido presentados en esta memoria de tesis en el Capítulo 4 son extendidos con operaciones que permiten tanto la creación de puentes (siempre que ésta sea posible, y de acuerdo a ciertas reglas y condiciones) como la propagación o proyección de restricciones de un dominio a otro. Obviamente, la computación sobre un resolutor para un dominio particular sigue manteniéndose como un caso particular.

De una forma más detallada, la propuesta que se desarrolla en las publicaciones [EFHR+07a, EFHR+07b, EFHR+08] sigue los siguientes tres pasos:

1) En primer lugar, se introduce un dominio especial de restricciones sobre el que es posible definir la coordinación de varios dominios, denominado *dominio de coordinación* $\mathcal{C}$. Este dominio puede ser utilizado como una instancia particular del esquema $CFLP(\mathcal{D})$, con la salvedad de que no cuenta con un resolutor de restricciones propio para este dominio. De existir, entonces éste sería un resolutor híbrido, que no es el enfoque que se pretende seguir en nuestra aproximación de permitir la cooperación entre los resolutores de cada dominio involucrados a través del dominio de coordinación. Es este dominio de coordinación el que proporciona las restricciones puente utilizadas para la comunicación y el intercambio de información entre los distintos dominios particulares.

2) En segundo lugar, en vez de desarrollar un resolutor híbrido, se desarrolla un nuevo cálculo coordinado de resolución de objetivos, inspirado en los presentados en el Capítulo 4. Este cálculo permite combinar la evaluación perezosa basada en el estrechamiento perezoso, como mecanismo de cómputo principal,

con la invocación de los resolutores individuales que intervienen en la comunicación. Adicionalmente, el cálculo permite realizar las dos operaciones de comunicación anteriormente descritas, la creación de puentes y la propagación de restricciones entre los diferentes dominios involucrados en la cooperación. Usando la semántica declarativa del esquema $CFLP(\mathcal{D})$ que se describe en el Capítulo 3 sobre los dominios de coordinación $\mathcal{C}$, se han obtenido resultados de corrección y completitud para el cálculo coordinado de resolución de objetivos, referidos al cómputo de *soluciones*. Los resultados de corrección y completitud presentados en el Capítulo 4 de la tesis son más generales que estos, ya que se refieren al cómputo de *respuestas* (compárese la Definición 18 y la Definición 19 en la Subsección 4.2.1).

3) En tercer lugar, se implementa la resolución de objetivos con cooperación de resolutores en el sistema $\mathcal{TOY}$. Esta implementación permite soportar la cooperación de resolutores a través de puentes y proyecciones para el dominio de Herbrand $\mathcal{H}$ y los dos dominios numéricos $\mathcal{R}$ y $\mathcal{FD}$.

La obtención de resultados de corrección y de completitud más generales relativos al cómputo de respuestas, la obtención de una implementación eficiente y suficientemente competitiva con respecto a otros sistemas de cooperación de resolutores ya existentes y el diseño de estrategias de cooperación y coordinación entre los distintos resolutores son aún parte del trabajo futuro a realizar en esta área, que sin duda contribuirán a enriquecer, en gran medida, las posibilidades teóricas y prácticas del esquema $CFLP(\mathcal{D})$ y de las técnicas de depuración declarativa presentadas en esta tesis.

7.2.2. Verificación de programas declarativos con restricciones

La formalización de una semántica declarativa en el Capítulo 3 de esta tesis para programas con restricciones en el esquema $CFLP(\mathcal{D})$ nos ha permitido estudiar técnicas específicas de depuración declarativa de respuestas incorrectas y perdidas en los Capítulos 5 y 6, respectivamente. Utilizando el mismo punto de partida resultaría de interés proponer un marco teórico que no solo sirva para la depuración de errores, sino también para la verificación de programas declarativos multi-paradigma con restricciones. Como aplicación de este posible trabajo futuro sería posible demostrar formalmente, de un modo (semi)-automático, propiedades de programas que utilicen restricciones concretas sobre dominios conocidos ($\mathcal{H}$, $\mathcal{R}$, $\mathcal{FD}$) o incluso restricciones heterogéneas sobre dominios híbridos (como el dominio de coordinación $\mathcal{C}$ introducido en la subsección anterior).

De una manera más concreta, los pasos a desarrollar seguirían una metodología análoga a la que ya se ha llevado a cabo con éxito para programas lógico funcionales sin restricciones en [CLL04]:

1) Formulación de una traducción de $CFLP(\mathcal{D})$-programas a $CLP(\mathcal{D})$-programas que dispongan de un modelo mínimo y de las nociones de compleción y de compleción inductiva correspondientes. De esta forma sería posible sacar partido de trabajos conocidos sobre *CLP* [JM94] a través del estudio de las relaciones existentes entre las propiedades válidas en el modelo mínimo del programa original (el que realmente nos interesa) y el del programa lógico con restricciones asociado.

2) Para el desarrollo del punto anterior resultaría suficiente con tomar en consideración el *cálculo de prueba positivo* que hemos presentado en el Capítulo 5 para el esquema $CFLP(\mathcal{D})$, el cual daría cuenta de las reducciones individuales de un modo similar a como procede el cálculo de pruebas proporcionado por la lógica para la reescritura *CRWL* en programación lógico funcional perezosa sin restricciones. Sin embargo, debido al hecho de tener también disponible un *calculo de prueba negativo* en el Capítulo 6 para la depuración declarativa de respuestas perdidas en el esquema $CFLP(\mathcal{D})$, creemos que resultaría de interés explorar este hecho como vía complementaria, ahora desde el punto de vista de la verificación de $CFLP(\mathcal{D})$-programas que usen la construcción *fail* de fallo finito estudiada en trabajos previos sobre programación lógico funcional [LS00, LS01, LS02, LS03, LS04].

7.2.3. Mejoras del depurador declarativo en el sistema $\mathcal{TOY}$

Las herramientas de depuración declarativa que hasta el momento hemos implementado en base a las técnicas expuestas en los Capítulos 5 y 6 de esta memoria tienen el inconveniente de que las preguntas que se le formulan al usuario durante una sesión de depuración para el diagnóstico de un síntoma de error (positivo o negativo) son con frecuencia demasiado complejas. Para aliviar este problema se han intentado diversas soluciones, como el uso de *especificaciones parciales ejecutables* del programa que permiten evitar ciertas preguntas [DNM89, BDM97, CR04], la inferencia de ciertas respuestas a partir de respuestas anteriores [CR02] o el diseño de árboles de cómputo ajustados a las necesidades de un problema de depuración específico sobre un dominio también específico [FLT03]. La optimización de la eficiencia de las herramientas actuales de depuración es parte de los retos más importantes de la investigación actual en este campo, y en consecuencia, una parte importante del trabajo futuro al que da lugar la presente tesis.

De manera más detallada, el objetivo que se plantearía es el de integrar las distintas herramientas ya implementadas, tanto para la depuración declarativa de respuestas incorrectas como de respuestas perdidas, en un único prototipo en el sistema $\mathcal{TOY}$ que sea lo más eficiente posible. Para ello, los pasos a seguir podrían ser los siguientes:

1) Evaluación de las herramientas de depuración actualmente desarrolladas a través de una serie de casos de prueba, elegidos teniendo en cuenta las experiencias publicadas sobre evaluación de depuradores declarativos, tanto en el campo de la programación lógica con restricciones como en el campo de la programación funcional perezosa. En cada prueba, se estimaría el número y la complejidad de las preguntas formuladas al usuario, así como la sobrecarga (en términos de tiempo y espacio) causada por la depuración con respecto a la ejecución ordinaria del mismo cómputo.

2) Optimización del depurador mediante la introducción de mejoras que puedan aminorar:

   - El número y la complejidad de las preguntas planteadas al usuario mediante la utilización de técnicas de simplificación de las restricciones que aparecen en las preguntas, así como de información auxiliar útil para obviar la necesidad de algunas de las preguntas. Para ello, se puede considerar el almacenamiento de las respuestas anteriores obtenidas, y más en general, de aserciones proporcionadas por el usuario que representen información parcial sobre la semántica pretendida del programa, y de las cuales se pueda deducir en ciertos casos la respuesta a futuras preguntas. El uso de aserciones será comentado más extensamente en la siguiente subsección.
   - La sobrecarga del consumo de tiempo y espacio, considerando una modificación del depurador que permita no calcular todo el árbol de deducción antes de iniciar la sesión de depuración, sino que lo vaya calculando gradualmente, según vaya siendo demandado. La implementación de esta idea requeriría modificaciones en la representación de los árboles de prueba y en la construcción de los programas transformados que los generan.

3) Evaluación de la efectividad de las optimizaciones realizadas por medio de los mismos casos de prueba que han sido empleados inicialmente.

Por último, como una posibilidad también interesante de mejorar nuestro prototipo de depurador declarativo de respuestas perdidas en el sistema $\mathcal{TOY}$, pensamos que sería posible obtener los árboles de cómputo en los que se basa nuestra técnica de diagnosis a partir de los denominados *redex trail*, utilizados en programación funcional [BFHH+07, CL07, CD08] y en programación lógico funcional [BHHV04]. Aunque nuestra implementación se basa en la generación de trazas adecuadas a partir del programa transformado $\mathcal{P}^{\mathcal{T}}$ tal y como se ha explicado en el Capítulo 6, creemos que esta otra posibilidad de proporcionar depuradores basados en redex trail podría permitirnos razonar sobre la corrección de la implementación mediante el uso de la semántica declarativa que sirve de soporte a esta estructura, de manera semejante a como se propone en [BFHH+07, LC07, CD08].

Por otra parte, la noción de *aca* que hemos utilizado para representar la recolección de respuestas en un cómputo guarda una cierta relación con otras técnicas de programación relativas en programación lógico funcional, como por ejemplo, la *búsqueda encapsulada* [BHH04] o la manipulación de *subespacios de cómputo* [AB07]. Sin embargo, los *aca*s, tal y como se utilizan en nuestro contexto, no son una técnica de programación, sino que sirven como sentencias lógicas cuya falsedad revela la incompletitud de respuestas computadas con respecto a las respuestas esperadas. En este sentido, una idea interesante a desarrollar como posible trabajo futuro sería la de utilizar un tipo especial de sentencias lógicas, similares a los *aca*s, pero que permitieran asegurar la *igualdad* entre los conjuntos observados y esperados de respuestas computadas para un mismo objetivo con un espacio de búsqueda finito. El desarrollo de esta idea podría dar lugar al diagnóstico declarativo de un tercer tipo de resultados inesperados, denominado *conjuntos de respuesta incorrecta* como se hace actualmente en *Datalog* [CGS07]. Sin embargo, pensamos que un diagnóstico separado de respuestas incorrectas y perdidas (en este orden) es en la práctica más conveniente para los usuarios de lenguajes declarativos multi-paradigma con restricciones.

7.2.4. Utilización de aserciones

Como señalamos en la introducción, la *diagnosis abstracta*, también llamada *diagnosis declarativa*, no es un tipo de depuración declarativa sino una técnica alternativa para la depuración de programas lógicos (ver, por ejemplo, [CLMV99]). Aunque también se trata de comparar la semántica del programa con la interpretación pretendida del mismo, en diagnosis abstracta se utilizan técnicas de interpretación abstracta [CC77, CC92] para tratar de probar ciertas propiedades que se cumplen en el modelo pretendido y que el usuario indica, generalmente, mediante el uso de *aserciones*. Una ventaja con respecto a la depuración declarativa es que no se precisa de un síntoma inicial, ni habitualmente interacción alguna con el usuario durante el proceso de depuración. Un caso sencillo pero ilustrativo de esta técnica son las declaraciones de tipo indicadas por el usuario y el correspondiente análisis de tipos en tiempo de compilación.

El sistema *CIAO* [CLIP97] desarrollado en la Universidad Politécnica de Madrid propone el uso de aserciones con tres propósitos diferentes:

1) Para detectar errores en tiempo de compilación mediante diagnosis abstracta.

2) Para detectar síntomas positivos o negativos durante la ejecución de un programa sin la necesidad de la intervención del usuario.

3) Para evitar algunas de las preguntas al usuario utilizando las aserciones como oráculo cuando ello es posible.

La inclusión de aserciones en relación con la depuración declarativa ya ha sido utilizada en depuradores declarativos para programación lógica y programación lógica con restricciones [DNM89, BDM97, CLIP97, HPB98, Lai00, PBH00a, PBH00b]. La idea general es la de utilizar aserciones para conocer la validez o no validez de los nodos del árbol de cómputo sin necesidad de consultar al usuario. En particular se suelen distinguir dos tipos de aserciones:

- *Positivas*, que indican conjuntos de hechos básicos que son válidos en la interpretación pretendida. En el caso de $\mathcal{TOY}$, una posible forma de representar estas aserciones podría ser $\varphi \Rightarrow f\ \bar{t}_n \rightarrow t$ indicando que cualquier instancia del hecho básico $f\ \bar{t}_n \rightarrow t$ correspondiente a una valoración de las variables que satisfaga la fórmula φ está en la interpretación pretendida. La definición del lenguaje aceptado para las fórmulas φ depende del tipo y generalidad de las aserciones que se quieran adoptar.

- *Negativas*, que indican conjuntos de hechos básicos que no son válidos en la interpretación pretendida. Con una notación análoga a la del apartado anterior estas aserciones se podrían representar en $\mathcal{TOY}$ en la forma $f\ \bar{t}_n \rightarrow t \Rightarrow \varphi$, indicando así que cualquier instancia del hecho básico $f\ \bar{t}_n \rightarrow t$ correspondiente a una valoración de las variables que no satisfaga φ no está en la interpretación pretendida.

Las aserciones podrían incluirse directamente en el código del programa. Por ejemplo, supongamos que nuestro lenguaje de aserciones admitiera fórmulas con igualdad, operaciones aritméticas y un operador $|\cdot|$ para representar la longitud de una lista, y consideremos la siguiente versión errónea de la función que concatena dos listas:

```
% @assert append X Y -> L => |X| + |Y| = |L|
  append []     L = L
  append [X|Xs] L = append Xs L
```

El marcador `@assert` indicaría al sistema la presencia de una aserción, en este caso de una aserción negativa. Durante la sesión de depuración, un hecho básico como `append [V] [U]` $\rightarrow$ `[U]`, deducible de un programa que contenga esta función, sería reconocido por el depurador como no válido en la interpretación pretendida, sin necesidad de tener que consultar al usuario.

Apéndices

Apéndice A

Demostraciones de resultados

Incluimos en este apéndice las demostraciones de las proposiciones, lemas y teoremas enunciados en los Capítulos 2, 3, 4 y 6. Estableciendo esta separación hemos pretendido facilitar la lectura de la tesis y de sus resultados, dejando para este apéndice las pruebas de los resultados principales que resultan excesivamente largas o técnicas. Algunas de las demostraciones dependen a su vez de lemas auxiliares cuyos enunciados y demostraciones también incluimos aquí.

A.1. Resultados presentados en el Capítulo 2

En esta sección incluimos las demostraciones de los principales resultados del esquema $CFLP(\mathcal{D})$, enunciados en el Capítulo 2 y que afectan a la corrección del resolutor del dominio de Herbrand $\mathcal{H}$.

A.1.1. Teorema de corrección del resolutor de $\mathcal{H}$

Comenzamos presentando la demostración del Lema 4 sobre las propiedades generales de los sistemas de transformación de almacenes de restricciones.

Demostración 25 (Demostración del Lema 4)

(1) El sistema de transformación de almacenes $\vDash_{\mathcal{D},\mathcal{X}}$ genera un árbol con raíz $\Pi \square \varepsilon$, cuyas hojas corresponden a almacenes que pertenecen a $\mathcal{SF}_{\mathcal{D}}(\Pi, \mathcal{X})$. Puesto que $\vDash_{\mathcal{D},\mathcal{X}}$ es finitamente ramificado y terminante, este árbol es localmente finito y no tiene ramas infinitas. Aplicando el denominado Lema de König *(ver, por ejemplo, [BN98]) el árbol ha de ser finito. Por tanto, debe tener una cantidad finita de hojas, y $\mathcal{SF}_{\mathcal{D}}(\Pi, \mathcal{X})$ es finito. Para un uso posterior, observamos que $solve^{\mathcal{D}}(\Pi, \mathcal{X})$ puede caracterizarse como*

$$\bigvee\{\exists \overline{Y'}.\, (\Pi' \square \sigma') \mid \Pi \square \varepsilon \vDash_{\mathcal{D},\mathcal{X}}! \; \Pi' \square \sigma',\, \overline{Y'} = var(\Pi' \square \sigma') \setminus var(\Pi)\}$$

(2) Supongamos que el sts posee la propiedad de variables libres y la propiedad de vinculación segura. Teniendo en cuenta lo comentado al final del apartado (1), para cada $\mathcal{X}$-forma resulta $\exists\overline{Y'}.(\Pi' \sqcap \sigma')$ computada mediante la llamada al resolutor $solve^{\mathcal{D}}(\Pi, \mathcal{X})$, exista alguna secuencia de pasos $\vdash_{\mathcal{D},\mathcal{X}}$ de la forma

$$\Pi \sqcap \varepsilon = \Pi'_0 \sqcap \mu'_0 \vdash_{\mathcal{D},\mathcal{X}} \Pi'_1 \sqcap \mu'_1 \vdash_{\mathcal{D},\mathcal{X}} \ldots \vdash_{\mathcal{D},\mathcal{X}} \Pi'_n \sqcap \mu'_n$$

tal que $\Pi'_n \sqcap \mu'_n = \Pi' \sqcap \sigma'$ es irreducible, y las siguientes condiciones se satisfacen para todo $1 \leq i \leq n$: $\Pi'_i \sqcap \mu'_i$ es un almacén con variables locales frescas $\overline{Y'_i} = var(\Pi'_i \sqcap \mu'_i) \setminus var(\Pi'_{i-1} \sqcap \mu'_{i-1})$; $\mu'_i = \mu'_{i-1}\mu_i$ para alguna sustitución μ_i para la que se verifica que $vdom(\mu_i) \cup vran(\mu_i) \subseteq var(\Pi'_{i-1}) \cup \overline{Y'_i}$; y $\mu_i(X)$ es una constante para todo $X \in \mathcal{X} \cap vdom(\mu_i)$. De este modo, $\overline{Y'} = \overline{Y'_1}, \ldots, \overline{Y'_n}$, y mediante un sencillo razonamiento por inducción sobre n es posible probar que $vdom(\sigma') \cup vran(\sigma') \subseteq var(\Pi) \cup \overline{Y'}$ y que $\sigma'(X)$ es una constante para todo $X \in \mathcal{X} \cap vdom(\sigma')$. Por tanto, el resolutor $solve^{\mathcal{D}}$ también satisface la propiedad de variables locales frescas y la propiedad de vinculación segura.

(3) Asumamos que $\vdash_{\mathcal{D},\mathcal{X}}$ el localmente correcto. Debido a la observación que hemos realizado en el apartado (1), para demostrar la corrección de $solve^{\mathcal{D}}$ es suficiente con mostrar que la unión

$$\bigcup\{Sol_{\mathcal{D}}(\exists\overline{Y'}.(\Pi' \sqcap \sigma')) \mid \Pi \sqcap \sigma \vdash_{\mathcal{D},\mathcal{X}}! \ \Pi' \sqcap \sigma', \overline{Y'} = var(\Pi' \sqcap \sigma') \setminus var(\Pi \sqcap \sigma)\}$$

es un subconjunto de $Sol_{\mathcal{D}}(\Pi \sqcap \sigma)$. Con el fin de demostrar esto, asumimos que

$$\Pi \sqcap \sigma \vdash^{n}_{\mathcal{D},\mathcal{X}}! \ \Pi' \sqcap \sigma', \ \overline{Y'} = var(\Pi' \sqcap \sigma') \setminus var(\Pi \sqcap \sigma)$$

y probamos que $Sol_{\mathcal{D}}(\exists\overline{Y'}.(\Pi' \sqcap \sigma')) \subseteq Sol_{\mathcal{D}}(\Pi \sqcap \sigma)$ razonando por inducción sobre n.

$\underline{n = 0}$: En este caso $\overline{Y'} = \emptyset$, $\Pi' \sqcap \sigma' = \Pi \sqcap \sigma$. La inclusión que ha de ser demostrada en este caso es trivial.

$\underline{n > 0}$: En este caso, $\Pi \sqcap \sigma \vdash_{\mathcal{D},\mathcal{X}} \Pi'_1 \sqcap \sigma'_1 \vdash^{n-1}_{\mathcal{D},\mathcal{X}}! \ \Pi' \sqcap \sigma'$ para algún almacén $\Pi'_1 \sqcap \sigma'_1$. Sea $\overline{Y'_1} = var(\Pi'_1 \sqcap \sigma'_1) \setminus var(\Pi \sqcap \sigma)$ y $\overline{Y''} = var(\Pi' \sqcap \sigma') \setminus var(\Pi'_1 \sqcap \sigma'_1)$. Entonces, $\overline{Y'} = \overline{Y'_1}, \overline{Y''} = var(\Pi' \sqcap \sigma') \setminus var(\Pi \sqcap \sigma)$. Por hipótesis de inducción, *podemos asumir que $Sol_{\mathcal{D}}(\exists\overline{Y''}.(\Pi' \sqcap \sigma')) \subseteq Sol_{\mathcal{D}}(\Pi'_1 \sqcap \sigma'_1)$. Entonces, para cualquier $\eta \in Sol_{\mathcal{D}}(\exists\overline{Y'}.(\Pi' \sqcap \sigma'))$ dada podemos probar que $\eta \in Sol_{\mathcal{D}}(\Pi \sqcap \sigma)$ mediante el siguiente razonamiento: por la definición de $Sol_{\mathcal{D}}$, existe $\eta' \in Sol_{\mathcal{D}}(\Pi' \sqcap \sigma')$ tal que $\eta' =_{\setminus\overline{Y'}} \eta$, y de aquí $\eta' =_{var(\Pi \sqcap \sigma)} \eta$. Trivialmente se sigue que $\eta' \in Sol_{\mathcal{D}}(\exists\overline{Y''}.(\Pi' \sqcap \sigma'))$, lo que implica que $\eta' \in Sol_{\mathcal{D}}(\Pi'_1 \sqcap \sigma'_1)$ por hipótesis de inducción. Trivialmente se sigue que $\eta' \in Sol_{\mathcal{D}}(\exists\overline{Y'_1}.(\Pi'_1 \sqcap \sigma'_1))$ lo que implica que $\eta' \in Sol_{\mathcal{D}}(\Pi \sqcap \sigma)$ debido a la corrección local. Puesto que $\eta' =_{var(\Pi \sqcap \sigma)} \eta$, podemos concluir que $\eta \in Sol_{\mathcal{D}}(\Pi \sqcap \sigma)$.*

(4) Asumamos ahora un conjunto seleccionado $\mathcal{RS}$ de strs tal que el sts es localmente completo para pasos $\mathcal{RS}$-libres. Teniendo en cuenta lo comentado al final del apartado (1), para demostrar la completitud de $solve^{\mathcal{D}}$ para invocaciones $\mathcal{RS}$-libres, es suficiente con mostrar que $WTSol_{\mathcal{D}}(\Pi \square \sigma)$ es un subconjunto de la unión

$$\bigcup\{WTSol_{\mathcal{D}}(\exists\overline{Y'}.(\Pi' \square \sigma')) \mid \Pi \square \sigma \vdash_{\mathcal{D},\mathcal{X}}! \ \Pi' \square \sigma', \quad \overline{Y'} = var(\Pi' \square \sigma') \setminus var(\Pi \square \sigma)\}$$

bajo la suposición adicional de que $\Pi \square \sigma$ es hereditariamente $\mathcal{RS}$-irreducible. Esto puede ser visto como una propiedad del almacén $\Pi \square \sigma$ que puede ser probada razonando por inducción bien fundada *(véase de nuevo [BN98]) sobre la relación de terminación de transformación de almacenes $\vdash_{\mathcal{D},\mathcal{X}}$:*

Caso Base: En este caso $\Pi \square \sigma$ es irreducible con respecto a $\vdash_{\mathcal{D},\mathcal{X}}$ y la unión se reduce al conjunto $WTSol_{\mathcal{D}}(\Pi \square \sigma)$. Por tanto, la inclusión que ha de ser probada es trivial.

Caso Inductivo: En este caso $\Pi \square \sigma$ es reducible con respecto a $\vdash_{\mathcal{D},\mathcal{X}}$. Como $\Pi \square \sigma$ es hereditariamente $\mathcal{RS}$-irreducible y el sts es localmente completo para pasos $\mathcal{RS}$-libres, para cualquier $\eta \in WTSol_{\mathcal{D}}(\Pi \square \sigma)$ existe algún $(\Pi'_1 \square \sigma'_1)$ hereditariamente $\mathcal{RS}$-irreducible tal que $\Pi \square \sigma \vdash_{\mathcal{D},\mathcal{X}} \Pi'_1 \square \sigma'_1$ y $\eta \in WTSol_{\mathcal{D}}(\exists\overline{Y'_1}(\Pi'_1 \square \sigma'_1))$, donde $\overline{Y'_1} = var(\Pi'_1 \square \sigma'_1) \setminus var(\Pi \square \sigma)$. Entonces, por la definición de $Sol_{\mathcal{D}}$, ha de existir $\eta'_1 \in WTSol_{\mathcal{D}}(\Pi'_1 \square \sigma'_1)$ tal que $\eta'_1 =_{\setminus\overline{Y'_1}} \eta$. La hipótesis de inducción *puede ser entonces asumida para $\Pi'_1 \square \sigma'_1$, por lo que ha de existir algún $\Pi' \square \sigma'$ tal que $\Pi'_1 \square \sigma'_1 \vdash_{\mathcal{D},\mathcal{X}}! \Pi' \square \sigma'$, $\overline{Y''} = var(\Pi' \square \sigma') \setminus var(\Pi'_1 \square \sigma'_1)$ y $\eta'_1 \in WTSol_{\mathcal{D}}(\exists\overline{Y''}(\Pi' \square \sigma'))$. Por definición de $Sol_{\mathcal{D}}$, existe $\eta' \in WTSol_{\mathcal{D}}(\Pi' \square \sigma')$ tal que $\eta' =_{\setminus\overline{Y''}} \eta'_1$. Más aún, $\Pi \square \sigma \vdash_{\mathcal{D},\mathcal{X}}! \Pi' \square \sigma'$ y $\overline{Y'} = \overline{Y'_1}, \overline{Y''} = var(\Pi' \square \sigma') \setminus var(\Pi \square \sigma)$ es tal que $\eta' =_{\setminus\overline{Y'}} \eta$, lo que permite concluir que $\eta \in WTSol_{\mathcal{D}}(\exists\overline{Y'}(\Pi' \square \sigma'))$.*

□

El Cuadro A.1, así como los dos lemas auxiliares que se muestran a continuación, serán utilizados en la demostración del Teorema 1, el principal resultado de esta sección. Este resultado asegura que $solve^{\mathcal{H}}$ satisface los requisitos para los resolutores dados en la Definición 5 (excepto una limitación técnica concerniente a la completitud). La demostración de este teorema también se apoya en el Lema 4.

Lema 17 (Lema Auxiliar de Corrección) *Asumamos $\Pi \subseteq PCon_{\mathcal{D}}$ y $\sigma, \sigma_1 \in Sub_{\mathcal{D}}$ tal que σ es idempotente y $\Pi\sigma = \Pi$. Entonces, $Sol_{\mathcal{D}}(\Pi\sigma_1) \cap Sol(\sigma\sigma_1) \subseteq Sol_{\mathcal{D}}(\Pi) \cap Sol(\sigma)$.*

REGLAS	P_1	P_2	P_3	P_4	P_5
$\mathbf{H}_1$	$\geq$	$\geq$	$\geq$	$>$	
$\mathbf{H}_2$	$\geq$	$\geq$	$\geq$	$>$	
$\mathbf{H}_3$	$\geq$	$\geq$	$>$		
$\mathbf{H}_4$	$\geq$	$\geq$	$\geq$	$\geq$	$>$
$\mathbf{H}_5$	$>$				
$\mathbf{H}_6$	$>$				
$\mathbf{H}_7$	$\geq$	$\geq$	$>$		
$\mathbf{H}_8$	$>$				
$\mathbf{H}_9$	$>$				
$\mathbf{H}_{10}$	$\geq$	$\geq$	$\geq$	$\geq$	$>$
$\mathbf{H}_{11a}$	$\geq$	$>$			
$\mathbf{H}_{11b}$	$>$				
$\mathbf{H}_{12}$	$>$				
$\mathbf{H}_{13}$	$>$				

Cuadro A.1: Orden de progreso bien fundamentado para $>_{lex}$

Demostración 26 *La hipótesis del lema asegura que* $\sigma = \sigma\sigma$ *y que* $\Pi\sigma = \Pi$. *Por otra parte, debido al Lema 3 (Lema de Sustitución) y a la definición de* $Sol_{\mathcal{D}}$, *cualquier* $\eta \in Val_{\mathcal{D}}$ *verifica que* $\eta \in Sol_{\mathcal{D}}(\Pi\sigma_1) \cap Sol(\sigma\sigma_1)$ *si y sólo si* $\sigma_1\eta \in Sol_{\mathcal{D}}(\Pi)$ *y* $\sigma\sigma_1\eta = \eta$. *Por tanto, para demostrar el lema es suficiente con asumir*

$$(a)\ \sigma = \sigma\sigma \qquad (b)\ \Pi\sigma = \Pi \qquad (c)\ \sigma_1\eta \in Sol_{\mathcal{D}}(\Pi) \qquad (d)\ \sigma\sigma_1\eta = \eta$$

y deducir a partir de estas suposiciones que $\eta \in Sol_{\mathcal{D}}(\Pi) \cap Sol(\sigma)$.

En primer lugar, probamos que $\eta \in Sol_{\mathcal{D}}(\Pi)$ *como sigue: por* (c) *y* (b), *obtenemos* $\sigma_1\eta \in Sol_{\mathcal{D}}(\Pi\sigma)$, *lo que da lugar a* $\sigma\sigma_1\eta \in Sol_{\mathcal{D}}(\Pi)$ *por el Lema 3. Por* (d), *esto es lo mismo que* $\eta \in Sol_{\mathcal{D}}(\Pi)$.

A continuación, observamos que $\eta \in Sol(\sigma)$ *es equivalente a* $\sigma\eta = \eta$, *lo que puede ser probado a partir de la siguiente cadena de igualdades:* $\sigma\eta =_{(d)} \sigma\sigma\sigma_1\eta =_{(a)} \sigma\sigma_1\eta =_{(d)} \eta$.

□

Lema 18 (Lema Auxiliar de Completitud) *Asumamos* $\Pi \subseteq PCon_{\mathcal{D}}$, $\sigma, \sigma_1 \in Sub_{\mathcal{D}}$ *y* $\eta, \eta' \in Val_{\mathcal{D}}$ *tal que* $\eta \in Sol_{\mathcal{D}}(\Pi) \cap Sol(\sigma)$, $\sigma_1\eta' = \eta'$ *y* $\eta' =_{\setminus \overline{Y'}} \eta$, *donde* $\overline{Y'}$ *son variable frescas no pertenecientes a* $var(\Pi) \cup vdom(\sigma) \cup vran(\sigma)$. *Entonces,* $\sigma\eta' = \eta'$ *y* $\eta' \in Sol_{\mathcal{D}}(\Pi\sigma_1) \cap Sol(\sigma\sigma_1)$.

Demostración 27 *En lo que sigue podemos asumir que $\sigma\eta = \eta$ debido a la hipótesis de que $\eta \in Sol(\sigma)$.*

Probamos que $\sigma\eta' = \eta'$ mostrando que $X\sigma\eta' = X\eta'$ se cumple para cualquier variable $X \in \mathcal{V}ar$. Esto es trivial para $X \notin vdom(\sigma)$. Para $X \in vdom(\sigma)$, podemos asumir que $\overline{Y'} \cap (\{X\} \cup var(X\sigma)) = \emptyset$; por tanto, $\eta' =_{X,var(X\sigma)} \eta$, y de aquí $X\sigma\eta' = X\sigma\eta = X\eta = X\eta'$ (donde la suposición $\sigma\eta = \eta$ ha sido usada en el segundo paso).

Ahora probamos que $\eta' \in Sol_{\mathcal{D}}(\Pi\sigma_1)$. Debido al Lema 3, esto es equivalente a que $\sigma_1\eta' \in Sol_{\mathcal{D}}(\Pi)$, lo que equivale a $\eta \in Sol_{\mathcal{D}}(\Pi)$ debido a la hipótesis $\sigma_1\eta' = \eta'$, $\eta' =_{\setminus\overline{Y'}} \eta$ y a que $\overline{Y'}$ no corta a $var(\Pi)$. Pero $\eta \in Sol_{\mathcal{D}}(\Pi)$ está asegurado por las hipótesis.

Finalmente, $\eta' \in Sol(\sigma\sigma_1)$ es equivalente a $\sigma\sigma_1\eta' = \eta'$, lo que puede ser demostrado como sigue: $\sigma\sigma_1\eta' = \sigma\eta' = \eta'$ (donde el primer paso se basa en la suposición $\sigma_1\eta' = \eta'$, y el segundo paso recae en la igualdad previamente probada).

□

Demostración 28 (Demostración del Teorema 1) *Consideramos el sts para almacenes de $\mathcal{H}$ cuya relación de transición $\vdash_{\mathcal{H},\mathcal{X}}$ se especifica en la Figura 2.1 en la Subsección 2.5.1, asumiendo implícitamente que la notación usada para las distintas strs es exactamente la que aparece allí. Probamos que este sts satisface las seis propiedades enumeradas en la Definición 6. La última de ellas (denominada* **Completitud Local***) se cumple para pasos $\mathcal{URS}$-libres, donde $\mathcal{URS} = \{$**OH3**,**OH7**,**H13**$\}$ es el conjunto de $\mathcal{H}$-strs no seguras, tal y como se explica en la Subsección 2.5.1.*

(1) **Variables Locales Frescas:** *La especificación de las strs en la Figura 2.1 claramente garantiza esta propiedad.*

(2) **Vinculación Segura:** *Una inspección de la Figura 2.1 muestra que las strs* **H1** *y* **H2** *vinculan una variable a una constante, y las otras transformaciones nunca vinculan una variable $X \in \mathcal{X}$. Por tanto, esta propiedad también se satisface.*

(3) **Ramificación Finita:** *Esta propiedad se cumple debido a que aquellas strs que permiten una elección indeterminista del siguiente almacén proporcionan sólo una cantidad finita de posibilidades.*

(4) **Terminación:** *Dado un $\mathcal{H}$ almacén $\Pi \,\square\, \sigma$ y un conjunto $\mathcal{X} \subseteq cvar(\Pi)$, definimos una quíntupla de números naturales $||\Pi \,\square\, \sigma||_{\mathcal{X}} =_{def} (P_1, P_2, P_3, P_4, P_5) \in \mathbb{N}^5$ donde*

- $\mathbf{P}_1$ *es el número de apariciones de restricciones atómicas en Π que están* no resueltas *con respecto a $\mathcal{X}$. En este contexto, una restricción atómica π que aparece en Π se dice que está no resuelta con respecto a $\mathcal{X}$ si y sólo si*

alguna de las $\vdash_{\mathcal{H},\mathcal{X}}$-transformaciones pueden ser aplicadas considerando π como la restricción atómica seleccionada.

- $\mathbf{P}_2$ *es la suma de las profundidades de todas las apariciones de variables $X \in \mathcal{X}$ dentro de patrones en Π.*
- $\mathbf{P}_3$ *es la suma de los tamaños sintácticos de todos los patrones que aparecen en Π.*
- $\mathbf{P}_4$ *es el número de apariciones* no resueltas *de variables obviamente demandadas en Π. En este contexto, una aparición de una variable obviamente demandada X en Π se denomina resuelta si y sólo si X aparece en una restricción de la forma $X == X$, y no resuelta en caso contrario.*
- $\mathbf{P}_5$ *es el número de apariciones de variables* inapropiadas *en Π. En este contexto, las apariciones inapropiadas de X en Π son aquellas apariciones de la forma $t == X$ o $t /= X$, con $t \in \mathcal{V}ar$ y $X \neq t$.*

Sea $>_{lex}$ el orden lexicográfico inducido por $>_{\mathbb{N}}$ sobre $\mathbb{N}^5$. Afirmamos que:

$$(\star)\ \Pi \,\square\, \sigma \vdash_{\mathcal{H},\mathcal{X}} \Pi' \,\square\, \sigma' \ \Rightarrow\ ||\Pi \,\square\, \sigma||_{\mathcal{X}} >_{lex} ||\Pi' \,\square\, \sigma'||_{\mathcal{X}}$$

Esto está justificado por la Figura A.1, la cual muestra el comportamiento de las diferentes $\vdash_{\mathcal{H},\mathcal{X}}$-transformaciones con respecto a $>_{lex}$. Con el fin de entender la figura, observamos que dos casos diferentes han de distinguirse en la aplicación de la transformación de almacenes $\mathbf{H}_{11}$, denominados:

- $\mathbf{H}_{11a}$*: Aplicación de $\mathbf{H}_{11}$ eligiendo un valor de i tal que $\mathcal{X} \cap var(t_i) \neq \emptyset$.*
- $\mathbf{H}_{11b}$*: Aplicación de $\mathbf{H}_{11}$ eligiendo un valor de i tal que $\mathcal{X} \cap var(t_i) = \emptyset$.*

Puesto que $>_{lex}$ es un orden bien fundamentado, la terminación de $\vdash_{\mathcal{H},\mathcal{X}}$ pueden concluirse a partir de $(\star)$. El lector puede remitirse a [BN98] para obtener más información sobre esta técnica de demostración.

(5) **Corrección Local:** *Dado un $\mathcal{H}$-almacén $\Pi \,\square\, \sigma$ y un conjunto $\mathcal{X} \subseteq odvar_{\mathcal{H}}(\Pi)$, debemos probar que la unión*

$$\bigcup\{Sol_{\mathcal{H}}(\exists\overline{Y'}.(\Pi' \,\square\, \sigma')) \mid \Pi \,\square\, \sigma \vdash_{\mathcal{H},\mathcal{X}} \Pi' \,\square\, \sigma',\ \overline{Y'} = var(\Pi' \,\square\, \sigma') \setminus var(\Pi \,\square\, \sigma)\}$$

es un subconjunto de $Sol_{\mathcal{D}}(\Pi \,\square\, \sigma)$. Obviamente, es suficiente con probar la inclusión

$$(\star)\ Sol_{\mathcal{H}}(\exists\overline{Y'}.(\Pi' \,\square\, \sigma')) \subseteq Sol_{\mathcal{H}}(\Pi \,\square\, \sigma)$$

para cada $\Pi' \,\square\, \sigma'$ tal que $\Pi \,\square\, \sigma \vdash_{\mathcal{H},\mathcal{X}}! \Pi' \,\square\, \sigma'$ con $\overline{Y'} = var(\Pi' \,\square\, \sigma') \setminus var(\Pi \,\square\, \sigma)$. Se tiene que $(\star)$ es una consecuencia fácil de

$$(\star\star)\ Sol_{\mathcal{H}}(\Pi' \,\square\, \sigma') \subseteq Sol_{\mathcal{H}}(\Pi \,\square\, \sigma)$$

En efecto, asumiendo $(\star\star)$ *y una solución arbitraria* $\eta \in Sol_{\mathcal{H}}(\exists\overline{Y'}.(\Pi' \,\square\, \sigma'))$*, ha de existir alguna* $\eta' \in Sol_{\mathcal{H}}(\Pi' \,\square\, \sigma')$ *tal que* $\eta =_{\setminus\overline{Y'}} \eta'$*. Entonces,* $\eta' \in Sol_{\mathcal{H}}(\Pi \,\square\, \sigma)$ *debido a* $(\star\star)$*, y de este modo* $\eta \in Sol_{\mathcal{H}}(\Pi \,\square\, \sigma)$ *porque* $\eta =_{\setminus\overline{Y'}} \eta'$ *y* $\overline{Y'} \cap var(\Pi \,\square\, \sigma) = \emptyset$*.*

Habiendo probado que $(\star\star)$ *implica* $(\star)$*, procedemos a probar* $(\star\star)$ *mediante una distinción de casos de acuerdo a la* $\vdash_{\mathcal{H},\mathcal{X}}$*-transformación usada en el paso* $\Pi \,\square\, \sigma \vdash_{\mathcal{H},\mathcal{X}} \Pi' \,\square\, \sigma'$*. En cada caso, asumimos que los almacenes* $\Pi \,\square\, \sigma$ *y* $\Pi' \,\square\, \sigma'$ *que aparecen en* $(\star\star)$ *tienen exactamente la forma que se muestra para la transformación correspondiente en la Figura 2.1 mostrada en la Subsección 2.5.1. Por ejemplo, en el caso de la transformación* **H1***, escribimos* $(t == s) \rightarrow\!!\ R,\ \Pi \ \square\ \sigma$ *en lugar de* $\Pi \,\square\, \sigma$*. Es más, en todos los casos usamos el hecho de que las restricciones y las variables dentro de cualquier almacén no están afectadas por la sustitución guardada en ese almacén.*

H1 *Asumimos que* $\eta \in Sol_{\mathcal{H}}((t == s, \Pi)\sigma_1 \,\square\, \sigma\sigma_1)$*. Entonces,* $\eta \in Sol_{\mathcal{H}}((t == s, \Pi)\sigma_1) \cap Sol(\sigma\sigma_1)$*. Tenemos que probar que* $\eta \in Sol_{\mathcal{H}}((t == s) \rightarrow\!!\ R, \Pi \,\square\, \sigma)$*.*

Puesto que $(t == s, \Pi) = (t == s, \Pi)\sigma$*, podemos inferir que* $\eta \in Sol_{\mathcal{H}}(t == s,\ \Pi) \cap Sol(\sigma)$ *a partir de nuestras suposiciones y del Lema 17.*

Aún queda por demostrar que $\eta \in Sol_{\mathcal{H}}((t == s) \rightarrow\!! R)$*. Puesto que ya sabemos que* $\eta \in Sol_{\mathcal{H}}(t == s)$*, es suficiente con probar que* $R\eta = true$*. Sin embargo,* $\eta \in Sol(\sigma\sigma_1)$ *significa que* $\sigma\sigma_1\eta = \eta$*, y por tanto,* $R\eta = R\sigma\sigma_1\eta = R\sigma_1\eta = true\ \eta = true$*.*

H2 *Muy similar a* **H1***.*

H3 *Trivial. Claramente,* $Sol_{\mathcal{H}}(\overline{t_m == s_m}) = Sol_{\mathcal{H}}(h\,\overline{t_m} == h\,\overline{s_m})$*.*

H4 *Trivial. Claramente,* $Sol_{\mathcal{H}}(X == t) = Sol_{\mathcal{H}}(t == X)$*.*

H5 *Asumamos que* $\eta \in Sol_{\mathcal{H}}(tot(t), \Pi\sigma_1 \square \sigma\sigma_1)$*. Entonces,* $t\eta$ *es un patrón total y* $\eta \in Sol_{\mathcal{H}}(\Pi\sigma_1) \cap Sol(\sigma\sigma_1)$*. Debemos probar que* $\eta \in Sol_{\mathcal{H}}(X == t, \Pi \,\square\, \sigma)$*.*

Puesto que $\Pi = \Pi\sigma$*, podemos inferir que* $\eta \in Sol_{\mathcal{H}}(\Pi) \cap Sol(\sigma)$ *a partir de nuestras suposiciones y del Lema 17. Aún queda por probar que* $\eta \in Sol_{\mathcal{H}}(X == t)$*. Pero* $\eta \in Sol(\sigma\sigma_1)$ *significa que* $\sigma\sigma_1\eta = \eta$*. De este modo,* $X\eta = X\sigma\sigma_1\eta = X\sigma_1\eta = t\eta$*, lo que implica que* $\eta \in Sol_{\mathcal{H}}(X == t)$*, puesto que* $t\eta$ *es total.*

H6 *Trivial, puesto que* $\eta \in Sol_{\mathcal{H}}(\blacksquare)$ *es falso para cualquier* η*.*

H7 *Trivial. Claramente,* $Sol_{\mathcal{H}}(\overline{t_i /= s_i}) \subseteq Sol_{\mathcal{H}}(h\,\overline{t_m}\ /= h\,\overline{s_m})$*.*

H8 *Trivial;* $\eta \in Sol_{\mathcal{H}}(h\,\overline{t_n}\ /= h'\,\overline{s_m})$ *se cumple para cualquier* η*.*

H9 *Trivial, por el mismo razonamiento que en* **H6**.

H10 *Trivial. Claramente,* $Sol_{\mathcal{H}}(X/= t) = Sol_{\mathcal{H}}(t\ /= X)$.

H11 *Asumamos que* $\eta \in Sol_{\mathcal{H}}((Z_i\ /= t_i, \Pi)\sigma_1 \,\square\, \sigma\sigma_1)$. *Entonces,* $\eta \in Sol_{\mathcal{H}}((Z_i\ /= t_i)\sigma_1)$ *y* $\eta \in Sol_{\mathcal{H}}(\Pi\sigma_1) \cap Sol(\sigma\sigma_1)$. *Tenemos que probar que* $\eta \in Sol_{\mathcal{H}}(X\ /= c\,\bar{t}_n, \Pi \,\square\, \sigma)$.

Puesto que $\Pi = \Pi\sigma$, *podemos inferir que* $\eta \in Sol_{\mathcal{H}}(\Pi) \cap Sol(\sigma)$ *a partir de nuestras suposiciones y del Lema 17. Aún queda por demostrar que* $\eta \in Sol_{\mathcal{H}}(X\ /= c\,\bar{t}_n)$. *Puesto que* $\eta \in Sol(\sigma\sigma_1)$, *sabemos que* $\sigma\sigma_1\eta = \eta$. *Por tanto, es suficiente con probar que* $\sigma\sigma_1\eta \in Sol_{\mathcal{H}}(X\ /= c\,\bar{t}_n)$, *lo que puede ser demostrado como sigue:*

$$\begin{array}{l} \sigma\sigma_1\eta \in Sol_{\mathcal{H}}(X/= c\,\bar{t}_n) \Leftrightarrow_{(1)} \eta \in Sol_{\mathcal{H}}(X/= c\,\bar{t}_n)\sigma\sigma_1 \\ \Leftrightarrow \eta \in Sol_{\mathcal{H}}(X/= c\,\bar{t}_n)\sigma_1 \Leftarrow_{(2)} \eta \in Sol_{\mathcal{H}}(Z_i\ /= t_i\sigma_1) \\ \Leftarrow_{(3)} \eta \in Sol_{\mathcal{H}}(Z_i\ /= t_i)\sigma_1 \end{array}$$

donde (1) *se cumple debido al Lema 3,* (2) *y* (3) *se cumplen por construcción de* σ_1, *y* $\eta \in Sol_{\mathcal{H}}(Z_i\ /= t_i)\sigma_1$ *se cumple debido a las suposiciones consideradas para este caso.*

H12 *Asumamos* $\eta \in Sol_{\mathcal{H}}(\Pi\sigma_1 \square \sigma\sigma_1)$. *Entonces,* $\eta \in Sol_{\mathcal{H}}(\Pi\sigma_1) \cap Sol(\sigma\sigma_1)$. *Tenemos que probar que* $\eta \in Sol_{\mathcal{H}}(X\ /= c\,\bar{t}_n, \Pi \,\square\, \sigma)$.

Puesto que $\Pi = \Pi\sigma$, *podemos inferir que* $\eta \in Sol_{\mathcal{H}}(\Pi) \cap Sol(\sigma)$ *a partir de nuestras suposiciones y del Lema 17. Aún queda por demostrar que* $\eta \in Sol_{\mathcal{H}}(X\ /= c\,\bar{t}_n)$. *Esto es así debido a que* $X\eta = X\sigma\sigma_1\eta = X\sigma_1\eta = (d\,\overline{Z}_m)\eta$, *donde la primera igualdad se cumple debido a la suposición de que* $\eta \in Sol(\sigma\sigma_1)$, *y la tercera igualdad se satisface por la propia construcción de* σ_1.

H13 *Trivial, por el mismo razonamiento realizado para* **H6**.

(6) **Completitud Local para Pasos $\mathcal{RS}$-libres:** *Comenzamos considerando el conjunto de strs no seguras* $\mathcal{URS} = \{\mathbf{OH3}, \mathbf{OH7}, \mathbf{H13}\}$ *definido en la Subsección 2.5.1. Asumamos un almacén de* $\mathcal{H}$ *de la forma* $\Pi \,\square\, \sigma$ *y un conjunto* $\mathcal{X} \subseteq odvar_{\mathcal{H}}(\Pi)$, *tal que* $\Pi \,\square\, \sigma$ *es* $\mathcal{URS}$*-irreducible pero no está en* $\mathcal{X}$*-forma resuelta. Probaremos que* $WTSol_{\mathcal{D}}(\Pi \,\square\, \sigma)$ *es un subconjunto de la unión*

$$\bigcup\{WTSol_{\mathcal{H}}(\exists\overline{Y'}.(\Pi' \,\square\, \sigma')) \mid \Pi \,\square\, \sigma \vdash_{\mathcal{H},\mathcal{X}} \Pi' \,\square\, \sigma', \quad \overline{Y'} = var(\Pi' \,\square\, \sigma') \setminus var(\Pi \,\square\, \sigma)\}$$

Dada cualquier solución bien tipada $\eta \in WTSol_{\mathcal{H}}(\Pi \,\square\, \sigma)$ *(la cual satisface en particular que* $\sigma\eta = \eta$*), debemos encontrar* $\Pi' \,\square\, \sigma'$ *y* η' *tal que*

$$(\star)\ \Pi \,\square\, \sigma \vdash_{\mathcal{H},\mathcal{X}} \Pi' \,\square\, \sigma',\ \eta' \in WTSol_{\mathcal{H}}(\Pi' \,\square\, \sigma'),\ \eta =_{\setminus \overline{Y'}} \eta'$$

de modo que $\eta \in WTSol_{\mathcal{H}}(\exists \overline{Y'}.(\Pi' \,\square\, \sigma'))$ esté asegurado. Debido a nuestras suposiciones sobre $\Pi \,\square\, \sigma$, ha de existir alguna str $\mathbf{H}_i \notin \mathcal{URS}$ que pueda ser usada para transformar $\Pi \,\square\, \sigma$. A continuación analizamos todas las posibilidades para $\mathbf{H}_i$, considerando todas las strs que se muestran en la Figura 2.1 en la Subsección 2.5.1, excepto **OH3**, **OH7** *y* **H13**. *En todos los casos concluimos que las condiciones mostradas en $(\star)$ se verifican. Cuando sea necesario considerar diferentes strs que puedan ser alternativamente aplicadas a un mismo almacén (como, por ejemplo,* **H1** *y* **H2**), *agruparemos todas las posibilidades dentro del mismo caso, argumentando que alguna de las reglas del grupo puede ser elegida para transformar $\Pi \,\square\, \sigma$ de modo que se aseguren las condiciones fijadas en $(\star)$. Asimismo, en todos los casos, asumiremos que los almacenes $\Pi \,\square\, \sigma$ y $\Pi' \,\square\, \sigma'$ que aparecen en $(\star)$ son exactamente de la misma forma que se muestra para la transformación correspondiente en la Figura 2.1, donde la restricción atómica seleccionada será representada por Π, y donde implícitamente usaremos el hecho de que las restricciones y las variables dentro de un almacén no se ven afectadas por la sustitución guardada en ese almacén.*

H1, H2 *En este caso, π es $(t == s) \rightarrow\! !\, R$, $\eta \in WTSol_{\mathcal{H}}(t == s \rightarrow\! !\, R, \Pi \,\square\, \sigma)$ y $\overline{Y'} = \emptyset$. Puesto que $\eta \in WTSol_{\mathcal{H}}(\pi)$, uno de los dos siguientes subcasos ha de cumplirse:*

(a) $\eta(R) = true$ y $\eta \in WTSol_{\mathcal{H}}(t == s)$ o bien
(b) $\eta(R) = false$ y $\eta \in WTSol_{\mathcal{H}}(t /= s)$

Asumamos que el subcaso (a) se cumple. Entonces, $(\star)$ puede ser asegurado mediante la transformación del almacén dado con **H1** *y probando que $\eta' = \eta \in WTSol_{\mathcal{H}}(\Pi\sigma_1 \,\square\, \sigma\sigma_1)$. Observamos que el Lema 18 puede ser aplicado con $\overline{Y'} = \emptyset$, $\eta' = \eta$ y $\sigma_1 = \{R \mapsto true\}$, debido a que la condición $\sigma_1\eta = \eta$ se sigue trivialmente de $\eta(R) = true$. Entonces, $\eta \in Sol_{\mathcal{H}}(\Pi\sigma_1) \cap Sol(\sigma\sigma_1)$ queda asegurado por el Lema 18, y η sigue siendo una solución bien tipada.*

Asumamos ahora que el subcaso (b) se cumple. Entonces, un argumento similar podría ser usado, pero utilizando **H2** *en lugar de* **H1**.

H3 *En este caso π is $h\,\overline{t}_m == h\,\overline{s}_m$, y $(\star)$ puede ser asegurada si se elige transformar el almacén dado mediante la regla* **H3** *y se toma $\overline{Y'} = \emptyset$ y $\sigma' = \sigma$. Se observa que h ha de ser m-transparente debido a que estamos asumiendo $\mathcal{URS}$-libre, y a que el* Lema de Transparencia *(Lema 2) puede ser aplicado para asegurar que η continua siendo una solución bien tipada del nuevo almacén.*

H4 *En este caso* π *es* $t == X$*, donde* t *no es una variable, y* $(\star)$ *puede ser trivialmente asegurado eligiendo transformar el almacén dado mediante la regla* **H4** *y tomando* $\overline{Y'} = \emptyset$ *y* $\sigma' = \sigma$*.*

H5 *En este caso* π *es* $t == X$*, con* $X \notin \mathcal{X}, X \notin var(t) y X \neq t$*. Más aún,* $\eta \in WTSol_{\mathcal{H}}(X == t, \Pi \square \sigma)$ *y* $\overline{Y'} = \emptyset$*. Entonces* $(\star)$ *puede ser asegurada transformando el almacén dado mediante la regla* **H5** *y probando que* $\eta' = \eta \in WTSol_{\mathcal{H}}(tot(t), \Pi\sigma_1 \square \sigma\sigma_1)$*. La suposición* $\eta \in WTSol_{\mathcal{H}}(\pi)$ *significa que* $\eta(X) = t\eta$ *es un patrón total, de modo que* $\eta(Y)$ *es también un patrón total para cada variable* $Y \in var(t)$*. En estas condiciones,* $\eta \in Sol_{\mathcal{H}}(tot(t))$ *y* $\sigma_1\eta = \eta$ *se cumple para* $\sigma_1 = \{X \mapsto t\}$*. Esto permite aplicar el Lema 18 con* $\overline{Y'} = \emptyset$*,* $\eta' = \eta$ *y* σ_1*, asegurando que* $\eta \in Sol_{\mathcal{H}}(\Pi\sigma_1) \cap Sol(\sigma\sigma_1)$*. Obviamente,* η *sigue siendo una solución bien tipada.*

H7 *En este caso,* π *es* $h\,\overline{t}_m \mathrel{/}= h\,\overline{s}_m$*. Debido a que* $\eta \in WTSol_{\mathcal{H}}(\pi)$*, ha de existir algún índice* i *tal que* $1 \leq i \leq m$ *y* $\eta \in WTSol_{\mathcal{H}}(t_i \mathrel{/}= s_i)$*. Entonces* $(\star)$ *puede ser asegurado eligiendo transformar el almacén dado mediante la regla* **H7** *y este valor particular de* i*, y tomando* $\overline{Y'} = \emptyset$*,* $\sigma' = \sigma$*. De nuevo, observamos que* h *ha de ser* m*-transparente debido a que estamos asumiendo* $\mathcal{URS}$*-libre, y a que el* Lema de Transparencia *(Lema 2) puede ser aplicado para asegurar que* η *continua siendo una solución bien tipada del nuevo almacén.*

H8 *En este caso,* π *es* $h\,\overline{t}_n \mathrel{/}= h'\,\overline{s}_m$ *con* $h \neq h'$ *o* $n \neq m$*, y* $(\star)$ *puede ser trivialmente asegurada eligiendo transformar el almacén de restricciones dado mediante la regla* **H8***, tomando* $\overline{Y'} = \emptyset$ *y* $\sigma' = \sigma$*.*

H10 *Este es un caso trivial, similar a* **H4***.*

H11,H12 *En este caso* π *es* $X \mathrel{/}= c\,\overline{t}_n$*, con* $X \notin \mathcal{X}$*,* $c \in DC^n$ *y* $\mathcal{X} \cap var(c\,\overline{t}_n) \neq \emptyset$*,* $\eta \in WTSol_{\mathcal{H}}(X \mathrel{/}= c\,\overline{t}_n, \Pi \square \sigma)$*. Debido a que* $\eta \in WTSol_{\mathcal{H}}(\pi)$*, uno de los dos siguientes subcasos se ha de cumplir para* $\eta(X)$*:*

(a) $\eta(X) = c\,\overline{s}_n$*, donde* $s_i \mathrel{/}= t_i\eta$ *se cumple para algún* $1 \leq i \leq n$*.*

(b) $\eta(X) = d\,\overline{s}_m$*, donde* $d \in DC^m$ *pertenece al mismo tipo de datos que* c*, pero* $d \neq c$*.*

Asumamos que el subcaso (a) se cumple. Entonces $(\star)$ *puede ser asegurado eligiendo transformar el almacén de restricciones dado mediante la regla* **H11** *y un valor particular de* i *tal que* $s_i \mathrel{/}= t_i\eta$ *se cumple, tomando* $\overline{Y'} = \overline{Z}_n$*, definiendo* η' *como la valoración que satisface* $\eta'(Z_j) = s_j$ *para todo* $1 \leq j \leq n$ *y* $\eta'(Y) = \eta(Y)$ *para cualquier otra variable* Y*, y probando que* $\eta' \in WTSol_{\mathcal{H}}((Z_i \mathrel{/}= t_i, \Pi)\sigma_1 \square \sigma\sigma_1)$*.*

Obviamente, $\eta =_{\setminus \overline{Y'}} \eta'$*. Más aún,* $\sigma_1\eta' = \eta'$*, puesto que* $X\sigma_1\eta' = (c\,\overline{s}_n)\eta' = c\,\overline{s}_n = X\eta = X\eta'$ *y* $Y\sigma_1\eta' = Y\eta'$ *para cualquier variable* $Y \neq X$*. Por tanto, el Lema 18 puede ser aplicado con el fin de asegurar que*

$\eta' \in Sol_{\mathcal{H}}(\Pi\sigma_1) \cap Sol(\sigma\sigma_1)$. Puesto que η era una solución bien tipada, η' es claramente también bien tipada bajo suposiciones de tipos apropiadas para las nuevas variables $\overline{Y'} = \overline{Z}_n$ introducidas por el paso de transformación. Queda solo por probar que $\eta' \in Sol_{\mathcal{H}}((Z_i \mathrel{/}= t_i)\sigma_1)$. Esto puede ser razonado mediante una cadena de equivalencias, finalizando con la condición que se ha de cumplir en el subcaso (a):

$$\eta' \in Sol_{\mathcal{H}}((Z_i \mathrel{/}= t_i)\sigma_1) \Leftrightarrow_{(1)} \sigma_1\eta' \in Sol_{\mathcal{H}}(Z_i \mathrel{/}= t_i) \Leftrightarrow_{(2)}$$
$$\eta' \in Sol_{\mathcal{H}}(Z_i \mathrel{/}= t_i) \Leftrightarrow \eta'(Z_i) \mathrel{/}= t_i\eta' \Leftrightarrow_{(3)} s_i \mathrel{/}= t_i\eta'$$

Observamos que (1) se cumple debido al Lema 3, (2) se cumple debido a que $\sigma_1\eta' = \eta'$, y (3) se cumple por construcción de η'. Esto finaliza la demostración para este subcaso.

Finalmente, asumimos ahora que el subcaso (b) se cumple. Entonces ($\star$) puede ser asegurada eligiendo transformar el almacén dado mediante la regla **H12** *y el constructor de datos particular $d \in DC^m$ para el cual sabemos que $\eta(X) = d\,\overline{s}_m$, tomando $\overline{Y'} = \overline{Z}_m$, definiendo η' como la valoración que satisface $\eta'(Z_j) = s_j$ para todo $1 \leq j \leq m$ y $\eta'(Y) = \eta(Y)$ para cualquier otra variable Y, y probando que $\eta' \in WTSol_{\mathcal{H}}(\Pi\sigma_1 \,\square\, \sigma\sigma_1)$. Obviamente, $\eta =_{\setminus \overline{Z}_m} \eta'$. Más aún, $\sigma_1\eta' = \eta'$ puede ser fácilmente comprobado, como en el subcaso (a). Por tanto, el Lema 18 puede ser aplicado para asegurar que $\eta' \in Sol_{\mathcal{H}}(\Pi\sigma_1) \cap Sol(\sigma\sigma_1)$. Finalmente, puesto que η era una solución bien tipada, η' es claramente también bien tipada bajo suposiciones de tipo apropiadas para las nuevas variables $\overline{Y'} = \overline{Z}_n$ introducidas por el paso de transformación.*

Usando los apartados anteriores y el Lema 4, podemos ahora afirmar que $solve^{\mathcal{H}}$ satisface todos los requisitos para resolutores que han sido enumerados en la Definición 5, excepto en el caso de la **Propiedad de Completitud***, la cual queda garantizada sólo para invocaciones seguras (es decir, $\mathcal{URS}$-libres) del resolutor, y la* **Propiedad de Discriminación** *que no ha sido aún considerada.*

La observación hecha en el apartado (1) del Lema 4 permite reformular la **Propiedad de Discriminación** *del modo siguiente: si un $\mathcal{H}$-almacén dado $\Pi \,\square\, \sigma$ no satisface ni (a) $\mathcal{X} \cap odvar(\Pi) \neq \emptyset$ ni (b) $\mathcal{X} \cap var(\Pi) = \emptyset$, entonces $\Pi \,\square\, \sigma$ puede ser reducido mediante alguna $\Vdash_{\mathcal{H},\mathcal{X}}$-transformación. Asumamos que $\Pi \,\square\, \sigma$ no satisface ni (a) ni (b). Debido a que no satisface (b), ha de existir algún $\pi \in \Pi$ tal que (c) $\mathcal{X} \cap var(\pi) \neq \emptyset$. Puesto que no satisface (a), este π ha de satisfacer (d) $\mathcal{X} \cap odvar(\pi) = \emptyset$, lo que junto con (c) implica (e) $\mathcal{X} \cap cvar(\pi) \neq \emptyset$. Usando (d), (e) y razonando por distinción de casos sobre la forma sintáctica de π, encontramos en todos los casos alguna $\Vdash_{\mathcal{H},\mathcal{X}}$-transformación que puede ser usada para transformar el almacén $\Pi \,\square\, \sigma$ tomando π como la restricción atómica seleccionada. Los casos*

son como sigue:

- π *es* $(t == s) \rightarrow! R$. *En este caso el almacén puede ser transformado mediante la regla* **H1** *o* **H2**.
- π *es* $h\,\bar{t}_m == h\,\bar{s}_m$. *En este caso el almacén puede ser transformado mediante la regla* **H3**.
- π *es* $t == X$ *con* $t \notin \mathcal{V}ar$. *En este caso el almacén puede ser transformado mediante la regla* **H4**.
- π *es* $X == t$ *con* $X \notin var(t)$ *y* $X \neq t$. *Debido a* (d) *sabemos que* $X \notin \mathcal{X}$, *y el almacén puede ser transformado por medio de la regla* **H5**.
- π *es* $X == t$ *con* $X \in var(t)$, $X \neq t$. *En este caso el almacén puede ser transformado por medio de la regla* **H6**.
- π *es* $h\,\bar{t}_m /= h\,\bar{s}_m$. *En este caso el almacén puede ser transformado por medio de la regla* **H7**.
- π *es* $h\,\bar{t}_n /= h'\,\bar{s}_m$, *con* $h \neq h'$ *o* $n \neq m$. *En este caso el almacén puede ser transformado por medio de la regla* **H8**.
- π *es* $t /= t$ *con* $t \in \mathcal{V}ar \cup DC \cup DF \cup SPF$. *En este caso el almacén puede ser transformado por medio de la regla* **H9**.
- π *es* $t /= X$ *con* $t \notin \mathcal{V}ar$. *En este caso el almacén puede ser transformado por medio de la regla* **H10**.
- π *es* $X /= c\,\bar{t}_n$, *con* $c \in DC^n$. *Debido a* $(d), (e)$ *sabemos que* $X \notin \mathcal{X}$ *y que* $\mathcal{X} \cap var(c\,\bar{t}_n) \neq \emptyset$. *Por tanto, el almacén puede ser transformado mediante la regla* **H11** *o* **H12**.
- π *es* $X /= h\,\bar{t}_m$ *con* $h \notin DC^m$. *Puesto que se da* $(d), (e)$ *sabemos que* $X \notin \mathcal{X}$ *y que* $\mathcal{X} \cap var(h\,\bar{t}_m) \neq \emptyset$. *Por tanto, el almacén puede transformarse mediante la regla* **H13**.

Esto completa la demostración de la **Propiedad de Discriminación** *y por tanto, del teorema.*

□

A.2. Resultados presentados en el Capítulo 3

En esta sección incluimos las demostraciones de los principales resultados enunciados en el Capítulo 3 que tienen que ver con la semántica declarativa del esquema $CFLP(\mathcal{D})$.

A.2.1. Propiedades del cálculo semántico

Comenzamos demostrando las propiedades básicas que han sido enunciadas en el Lema 5 sobre el cálculo semántico presentado en la Definición 10 del Capítulo 3.

Demostración 29 (Demostración del Lema 5)

(1) ***Propiedad de Compacidad****. Asumamos un árbol de prueba dado T para $\mathcal{I} \Vdash_{\mathcal{D}} \varphi$. Sea $\mathcal{I}_0$ el subconjunto finito de $\mathcal{I}$ formado por todos los c-hechos de $\mathcal{I}$ utilizados en los pasos de inferencia de T que usen la regla $DF_{\mathcal{I}}$. Entonces es obvio que $T : cl_{\mathcal{D}}(\mathcal{I}_0) \Vdash_{\mathcal{D}} \varphi$.*

De manera más detallada, razonamos por inducción sobre $\|T\|$ para probar la existencia de algún subconjunto finito $\mathcal{I}_0 \subseteq \mathcal{I}$ y de un árbol de prueba T' tal que $T' : cl_{\mathcal{D}}(\mathcal{I}_0) \Vdash_{\mathcal{D}} \varphi$. Distinguimos casos de acuerdo con la regla de inferencia aplicada en la raíz de T.

En primer lugar, si T es un árbol de prueba simple, la propiedad se satisface trivialmente debido a que $T : \mathcal{I} \Vdash_{\mathcal{D}} \varphi$ es una derivación a partir de la c-interpretación trivial $cl_{\mathcal{D}}(\emptyset)$. Por lo tanto, T y T' son el mismo árbol de prueba simple para probar $cl_{\mathcal{D}}(\emptyset) \Vdash_{\mathcal{D}} \varphi$, y por supuesto, para $cl_{\mathcal{D}}(\mathcal{I}_0) \Vdash_{\mathcal{D}} \varphi$ con $\mathcal{I}_0 \subseteq \mathcal{I}$ cualquier subconjunto finito.
En otro caso, usando la hipótesis de inducción *y el hecho de que sólo usamos a lo sumo un c-hecho de $\mathcal{I}$ en cada paso de la derivación, la propiedad es obvia para todo el resto de reglas de inferencia aplicadas a la raíz de T.*

Por ejemplo, si $\varphi = f\,\overline{e}_n\overline{a}_k \rightarrow t \Leftarrow \Pi$ y $T = \mathbf{DF}_{\mathcal{I}}(f\,\overline{e}_n\overline{a}_k \rightarrow t \Leftarrow \Pi, [T_1, \ldots, T_n, T_s])$ para algún c-hecho $(f\,\overline{t}_n \rightarrow s \Leftarrow \Pi) \in \mathcal{I}$ tal que $T_i : \mathcal{I} \Vdash_{\mathcal{D}} e_i \rightarrow t_i \Leftarrow \Pi$ con $\|T_i\| < \|T\|$ $(1 \leq i \leq n)$ y $T_s : \mathcal{I} \Vdash_{\mathcal{D}} s\,\overline{a}_k \rightarrow t \Leftarrow \Pi$ con $\|T_s\| < \|T\|$, por hipótesis de inducción *tenemos que $T_i' : cl_{\mathcal{D}}(\mathcal{I}_i) \Vdash_{\mathcal{D}} e_i \rightarrow t_i \Leftarrow \Pi$ para algún subconjunto finito $\mathcal{I}_i \subseteq \mathcal{I}$ $(1 \leq i \leq n)$, y $T_s' : cl_{\mathcal{D}}(\mathcal{I}_s) \Vdash_{\mathcal{D}} s\,\overline{a}_k \rightarrow t \Leftarrow \Pi$ para algún subconjunto finito $\mathcal{I}_s \subseteq \mathcal{I}$.*

Entonces, podemos definir el subconjunto finito $\mathcal{I}_0 =_{def} \bigcup_{i=1}^{m} \mathcal{I}_i \cup \mathcal{I}_s \cup \{f\,\overline{t}_n \rightarrow s \Leftarrow \Pi\}$. Observamos que $\mathcal{I}_0 \subseteq \mathcal{I}$ y que $cl_{\mathcal{D}}(\mathcal{I}_0) = \bigcup_{i=1}^{m} cl_{\mathcal{D}}(\mathcal{I}_i) \cup cl_{\mathcal{D}}(\mathcal{I}_s) \cup cl_{\mathcal{D}}(\{f\,\overline{t}_n \rightarrow s \Leftarrow \Pi\})$. Más aún, tenemos que $T_i' : cl_{\mathcal{D}}(\mathcal{I}_0) \Vdash_{\mathcal{D}} e_i \rightarrow t_i \Leftarrow \Pi$ $(1 \leq i \leq n)$, $T_s' : cl_{\mathcal{D}}(\mathcal{I}_0) \Vdash_{\mathcal{D}} s\,\overline{a}_k \rightarrow t \Leftarrow \Pi$ y $(f\,\overline{t}_n \rightarrow s \Leftarrow \Pi) \in cl_{\mathcal{D}}(\mathcal{I}_0)$. De aquí, $T' =_{def} \mathbf{DF}_{cl_{\mathcal{D}}(\mathcal{I}_0)}(f\,\overline{e}_n\overline{a}_k \rightarrow t \Leftarrow \Pi, [T_1', \ldots, T_n', T_s'])$.

(2) ***Propiedad de Extensión****. Asumamos un árbol de prueba dado T para $\mathcal{I} \Vdash_{\mathcal{D}} \varphi$. Entonces es obvio que también se cumple $T : \mathcal{I}' \Vdash_{\mathcal{D}} \varphi$, porque todos los c-hechos pertenecientes a $\mathcal{I}$ que se utilicen en los pasos de inferencia $DF_{\mathcal{I}}$ de T también pertenecen a $\mathcal{I}'$.*

De manera más detallada, razonamos por inducción sobre $\|T\|$ para probar la existencia de un árbol de prueba T' para $\mathcal{I}' \Vdash_{\mathcal{D}} \varphi$.

En primer lugar, si T es un árbol de prueba simple entonces la propiedad se

satisface trivialmente debido a que $T : \mathcal{I} \Vdash_{\mathcal{D}} \varphi$ *es una derivación a partir de la c-interpretación trivial* $cl_{\mathcal{D}}(\emptyset)$. *Por lo tanto,* T *y* T' *son el mismo árbol de prueba simple para* $cl_{\mathcal{D}}(\emptyset) \Vdash_{\mathcal{D}} \varphi$, *y por supuesto, para* $\mathcal{I}' \Vdash_{\mathcal{D}} \varphi$.

En otro caso, usando la hipótesis de inducción *y el hecho de que* $\mathcal{I} \subseteq \mathcal{I}'$ *si* $\mathcal{I}$ *es necesario en la derivación, la propiedad es obvia para todo el resto de reglas de inferencia aplicadas a la raíz de* T.

Por ejemplo, si $\varphi = f\,\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi$ *y* $T = \mathbf{DF}_{\mathcal{I}}(f\,\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi, [T_1, \ldots, T_n, T_s])$ *para algún c-hecho* $(f\,\overline{t}_n \to s \Leftarrow \Pi) \in \mathcal{I}$ *tal que* $T_i : \mathcal{I} \Vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$ *con* $\|T_i\| < \|T\|$ $(1 \leq i \leq n)$ *y* $T_s : \mathcal{I} \Vdash_{\mathcal{D}} s\,\overline{a}_k \to t \Leftarrow \Pi$ *con* $\|T_s\| < \|T\|$, *por* hipótesis de inducción *tenemos que* $T'_i : \mathcal{I}' \Vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$ $(1 \leq i \leq n)$ *y* $T'_s : \mathcal{I}' \Vdash_{\mathcal{D}} s\,\overline{a}_k \to t \Leftarrow \Pi$.

Más aún, puesto que $\mathcal{I} \subseteq \mathcal{I}'$, *también tenemos que* $(f\,\overline{t}_n \to s \Leftarrow \Pi) \in \mathcal{I}'$. *De aquí,* $T' =_{def} \mathbf{DF}_{\mathcal{I}'}(f\,\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi, [T'_1, \ldots, T'_n, T'_s])$, *el cuál verifica* $T' : \mathcal{I}' \Vdash_{\mathcal{D}} \varphi$.

(3) ***Propiedad de Aproximación****. En el caso en el que* $Sol_{\mathcal{D}}(\Pi) = \emptyset$, $\Pi \models_{\mathcal{D}} e \sqsupseteq t$ *es trivialmente cierto y* $T : \Vdash_{\mathcal{D}} e \to t \Leftarrow \Pi$ *con una regla de inferencia* **TI**. *En el resto de esta demostración podemos asumir que* $Sol_{\mathcal{D}}(\Pi) \neq \emptyset$ *y razonar por inducción sobre el tamaño sintáctico de* e. *Distinguimos casos para* t:

- $t = \bot$. *En este caso,* $\Pi \models_{\mathcal{D}} e \sqsupseteq \bot$ *es trivialmente cierto y* $T : \Vdash_{\mathcal{D}} e \to \bot \Leftarrow \Pi$ *con una inferencia* **TI**.

- $t = u \in \mathcal{B}^{\mathcal{D}}$. *Consideramos varios subcasos para* e. *Si* $e = u$ *entonces* $\Pi \models_{\mathcal{D}} u \sqsupseteq u$ *es cierto y* $T : \Vdash_{\mathcal{D}} u \to u \Leftarrow \Pi$ *con una inferencia* **RR**. *Si* $e = X \in \mathcal{V}ar$ *y* $\Pi \models_{\mathcal{D}} X \sqsupseteq u$ *entonces* $T : \Vdash_{\mathcal{D}} X \to u \Leftarrow \Pi$ *con una inferencia* **SP**, *y si* $\Pi \not\models_{\mathcal{D}} X \sqsupseteq u$ *entonces* $\not\Vdash_{\mathcal{D}} X \to u \Leftarrow \Pi$ *(puesto que ninguna regla de inferencia es aplicable).*

  Finalmente, si e *no es ni* u *ni una variable entonces* $\Pi \not\models_{\mathcal{D}} e \sqsupseteq u$. *Asumamos un árbol de prueba* $T : \Vdash_{\mathcal{D}} e \to u \Leftarrow \Pi$ *(si no hubiera árbol de prueba, entonces ya estaría demostrado). Puesto que la c-interpretación es* $cl_{\mathcal{D}}(\emptyset)$ *y* $e \neq u$, $e \notin \mathcal{V}ar$, *la regla de inferencia aplicada en la raíz de* T *ha de ser* **PF** *o bien* **AC**, *y por tanto,* T *no es simple.*

- $t = X \in \mathcal{V}ar$. *Consideramos varios subcasos para* e. *Si* $e = X$ *entonces* $\Pi \models_{\mathcal{D}} X \sqsupseteq X$ *y* $T : \Vdash_{\mathcal{D}} X \to X \Leftarrow \Pi$ *con una inferencia* **RR**. *Si* e *es un patrón* $s \neq X$ *y* $\Pi \models_{\mathcal{D}} s \sqsupseteq X$ *entonces* $T : \Vdash_{\mathcal{D}} s \to X \Leftarrow \Pi$ *con una inferencia* **SP**, *y si* $\Pi \not\models_{\mathcal{D}} s \sqsupseteq X$ *entonces* $\not\Vdash_{\mathcal{D}} s \to X \Leftarrow \Pi$ *(puesto que ninguna regla de inferencia es aplicable).*

  Finalmente, si e *no es un patrón, consideramos cualquier* $\mu \in Sol_{\mathcal{D}}(\Pi)$ *tal que* $X\mu$ *es un patrón total. Entonces,* $e\mu \sqsupseteq X\mu$ *no es cierto y por tanto* $\Pi \not\models_{\mathcal{D}} e \sqsupseteq X$. *Asumamos un árbol de prueba* $T : \Vdash_{\mathcal{D}} e \to X \Leftarrow \Pi$ *(si no*

hubiera un árbol de prueba, entonces ya estaría demostrado). Puesto que la c-interpretación es $cl_{\mathcal{D}}(\emptyset)$ *y* e *no es un patrón, la regla de inferencia aplicada a la raíz de* T *ha de ser* **IR**, **PF** *o bien* **AC**. *En los dos últimos casos,* T *no es simple.*

En el primer caso, podemos asumir que $e = h\,\overline{e_m}$ *es una expresión rígida y pasiva pero no un patrón. De aquí,* $T = \mathbf{IR}(h\,\overline{e}_m \to X \Leftarrow \Pi, [T_1, \ldots, T_m])$, *y para cada* $1 \leq i \leq m$, $T_i : \Vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$ *tal que* $\Pi \models_{\mathcal{D}} h\,\overline{t}_m \sqsupseteq X$. *Entonces, para algún* $1 \leq i \leq m$, $\Pi \not\models_{\mathcal{D}} e_i \sqsupseteq t_i$. *De otro modo podríamos tener* $\Pi \models_{\mathcal{D}} e_i \sqsupseteq t_i$ *para todo* $1 \leq i \leq m$, *y entonces* $\Pi \models_{\mathcal{D}} h\,\overline{e}_m \sqsupseteq h\,\overline{t}_m$ *y* $\Pi \models_{\mathcal{D}} h\,\overline{e}_m \sqsupseteq X$, *que no es el caso. Fijamos cualquier* $1 \leq i \leq m$ *tal que* $\Pi \not\models_{\mathcal{D}} e_i \sqsupseteq t_i$. *Por* hipótesis de inducción *(observamos que el tamaño de* e_i *es más pequeño que el tamaño de* $h\,\overline{e}_m$*),* T_i *no es un árbol de prueba simple. Por lo tanto,* T *no es simple tampoco.*

- $t = h\,\overline{t_m}$ *con* $t_1, \ldots, t_m$ *patrones. Consideramos de nuevo varios subcasos para* e. *Si* $e = X \in \mathcal{V}ar$ *y* $\Pi \models_{\mathcal{D}} X \sqsupseteq h\,\overline{t}_m$ *entonces* $T : \Vdash_{\mathcal{D}} X \to h\,\overline{t}_m \Leftarrow \Pi$ *con una inferencia* **SP**, *y si* $\Pi \not\models_{\mathcal{D}} X \sqsupseteq h\,\overline{t}_m$ *entonces* $\not\Vdash_{\mathcal{D}} X \to h\,\overline{t}_m \Leftarrow \Pi$ *(puesto que ninguna regla de inferencia es aplicable).*

  Si $e = h\,\overline{e_m}$ *y* $\Pi \models_{\mathcal{D}} h\,\overline{e}_m \sqsupseteq h\,\overline{t}_m$ *entonces* $\Pi \models_{\mathcal{D}} e_i \sqsupseteq t_i$ *para todo* $1 \leq i \leq m$. *De aquí, por* hipótesis de inducción *(el tamaño de* e_i *es más pequeño que el tamaño de* $h\,\overline{e}_m$*),* $\Vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$ *con un árbol de prueba simple* T_i *para todo* $1 \leq i \leq m$ *y* $T : \Vdash_{\mathcal{D}} h\,\overline{e}_m \to h\,\overline{t}_m \Leftarrow \Pi$ *con* $T = \mathbf{DC}(h\,\overline{e}_m \to h\,\overline{t}_m \Leftarrow \Pi, [T_1, \ldots, T_m])$ *un árbol de prueba simple. Más aún, si* $\Pi \not\models_{\mathcal{D}} h\,\overline{e}_m \sqsupseteq h\,\overline{t}_m$ *entonces* $\Pi \not\models_{\mathcal{D}} e_i \sqsupseteq t_i$ *para algún* $1 \leq i \leq m$. *De aquí, por* hipótesis de inducción, *hay algún* $1 \leq i \leq m$ *tal que no existe un árbol de prueba simple* T_i *para* $\Vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$. *Por lo tanto, ningún árbol de prueba simple* T *existe para* $\Vdash_{\mathcal{D}} h\,\overline{e}_m \to h\,\overline{t}_m \Leftarrow \Pi$ *(***DC** *no serviría y ninguna otra regla de inferencia podría aplicarse).*

  Finalmente, si e *no es ni una variable ni es de la forma* $h\,\overline{e}_m$, *entonces consideramos cualquier* $\mu \in Sol_{\mathcal{D}}(\Pi)$ *total. Se tiene entonces que* $e\mu \sqsupseteq (h\overline{t}_m)\mu$ *no es cierto. De aquí,* $\Pi \not\models_{\mathcal{D}} e \sqsupseteq h\overline{t}_m$. *Si* $\not\Vdash_{\mathcal{D}} e \to h\overline{t}_m \Leftarrow \Pi$ *ya lo tendríamos demostrado. Si hay algún árbol de prueba* $T : \Vdash_{\mathcal{D}} e \to h\overline{t}_m \Leftarrow \Pi$, *la regla de inferencia aplicada en la raíz ha de ser* **PF** *o bien* **AC** *(*$\mathbf{DF}_{\mathcal{I}}$ *no puede ser usada con la c-interpretación trivial* $cl_{\mathcal{D}}(\emptyset)$*). En cualquier caso,* T *no es simple.*

(4) ***c-Átomos Primitivos****. Comenzamos demostrando la parte correspondiente al "sólo si". Por hipótesis inicial, un árbol de prueba* T *para* $\mathcal{I} \Vdash_{\mathcal{D}} p\,\overline{t}_n \to !\,t \Leftarrow \Pi$ *ha de ser de la forma* $T = \mathbf{AC}(p\,\overline{t}_n \to !\,t \Leftarrow \Pi, [T_1, \ldots, T_n])$, *donde* $\Pi \models_{\mathcal{D}} p\,\overline{t'}_n \to !\,t$ *para algunos* $t'_1, \ldots, t'_n \in Pat_{\mathcal{D}}$ *y* $T_i : \mathcal{I} \Vdash_{\mathcal{D}} t_i \to t'_i \Leftarrow \Pi$ $(1 \leq i \leq n)$ *son árboles de*

prueba simples. Más aún, $\Pi \models_{\mathcal{D}} t_i \sqsupseteq t'_i$ $(1 \leq i \leq n)$ *se deduce usando la* Propiedad de Aproximación, *y entonces* $\Pi \models_{\mathcal{D}} p\,\bar{t}_n \rightarrow\!! \, t$.

Probamos ahora la parte del "si". Puesto que $\Pi \models_{\mathcal{D}} p\,\bar{t}_n \rightarrow\!! \, t$ *por hipótesis inicial y* $T_i : \Vdash_{\mathcal{D}} t_i \rightarrow t_i \Leftarrow \Pi$ *son árboles de prueba simples para todo* $1 \leq i \leq n$, *usando la* Propiedad de Aproximación *podemos construir un árbol de prueba* $T =_{def} \mathbf{AC}(p\,\bar{t}_n \rightarrow\!! \, t \Leftarrow \Pi, [T_1, \ldots, T_n])$ *para* $\mathcal{I} \Vdash_{\mathcal{D}} p\,\bar{t}_n \rightarrow\!! \, t \Leftarrow \Pi$.

(5) ***Propiedad de Implicación****. Asumamos* $\varphi \succcurlyeq_{\mathcal{D}} \varphi'$ *y una sustitución* σ *que relaciona* φ *y* φ' *como se indica en la* relación de implicación *(ver el apartado (2) de la Definición 8). Podemos también asumir que* $Sol_{\mathcal{D}}(\Pi') \neq \emptyset$, *de otro modo* $T' : \mathcal{I} \Vdash_{\mathcal{D}} \varphi'$ *con un árbol de prueba simple* $T' =_{def} \mathbf{TI}(\varphi', [\,])$ *y* $\mid T \mid \geq 0 = \mid T' \mid$. *Sea* T *un árbol de prueba dado para* $\mathcal{I} \Vdash_{\mathcal{D}} \varphi$. *Razonando por inducción sobre* $\|T\|$ *probamos la existencia de un árbol de prueba* T' *para* $\mathcal{I} \Vdash_{\mathcal{D}} \varphi'$ *tal que* $\mid T \mid \geq \mid T' \mid$. *Distinguimos varios casos posibles:*

- T *es un árbol de prueba simple y* $\varphi = e \rightarrow t \Leftarrow \Pi$. *Esto cubre los casos en los que* T *es de alguna de las formas* $\mathbf{TI}(\varphi, [\,])$, $\mathbf{RR}(\varphi, [\,])$, $\mathbf{SP}(\varphi, [\,])$ *o* $\mathbf{DC}(\varphi, [\,])$. *Puesto que* T *es simple,* $\mathbf{DF}_{\mathcal{I}}$ *no se utiliza. Por lo tanto,* $T : \Vdash_{\mathcal{D}} e \rightarrow t \Leftarrow \Pi$. *Por la* Propiedad de Aproximación, $\Pi \models_{\mathcal{D}} e \sqsupseteq t$. *Esto implica que* $\Pi\sigma \models_{\mathcal{D}} e\sigma \sqsupseteq t\sigma$. *Puesto que* $\varphi \succcurlyeq_{\mathcal{D}} \varphi'$, *sabemos que* $\varphi' = e' \rightarrow t' \Leftarrow \Pi'$ *con* $\Pi' \models_{\mathcal{D}} \Pi\sigma$, $\Pi' \models_{\mathcal{D}} e' \sqsupseteq e\sigma$ *y* $\Pi' \models_{\mathcal{D}} t\sigma \sqsupseteq t'$. *Podemos concluir que* $\Pi' \models_{\mathcal{D}} e' \sqsupseteq t'$. *Por la* Propiedad de Aproximación *de nuevo, existe un árbol de prueba simple* $T' : \Vdash_{\mathcal{D}} e' \rightarrow t' \Leftarrow \Pi'$, *y por supuesto* $T' : \mathcal{I} \Vdash_{\mathcal{D}} e' \rightarrow t' \Leftarrow \Pi'$. *Puesto que* T *y* T' *son ambos simples,* $\mid T \mid = 0 \geq 0 = \mid T' \mid$.

- $T = \mathbf{DC}(h\,\bar{e}_m \rightarrow h\,\bar{t}_m \Leftarrow \Pi, [T_1, \ldots, T_m])$. *En este caso, sabemos que* $\varphi = h\,\bar{e}_m \rightarrow h\,\bar{t}_m \Leftarrow \Pi$ *con* $T_i : \mathcal{I} \Vdash_{\mathcal{D}} e_i \rightarrow t_i \Leftarrow \Pi$, $\|T_i\| < \|T\|$ $(1 \leq i \leq m)$ *y* $\varphi' = e' \rightarrow t' \Leftarrow \Pi'$ *con* $\Pi' \models_{\mathcal{D}} \Pi\sigma$, $\Pi' \models_{\mathcal{D}} e' \sqsupseteq (h\,\bar{e}_m)\sigma$, $\Pi' \models_{\mathcal{D}} (h\,\bar{t}_m)\sigma \sqsupseteq t'$.

  Podemos asumir que T *no es simple; en caso contrario podríamos razonar como en el caso anterior. Puesto que* T *no es simple,* $h\,\bar{e}_m$ *no es un patrón. Entonces ha de ser el caso en el que* $e' = h\,\overline{e'_m}$ *con* $\Pi' \models_{\mathcal{D}} e'_i \sqsupseteq e_i\sigma$ *para todo* $1 \leq i \leq m$ *(de otro modo, cualquier* $\mu \in Sol_{\mathcal{D}}(\Pi')$ *total podría ser tal que* $e'\mu \sqsupseteq (h\,\bar{e}_m)\sigma\mu$ *no sea cierto). Más aún,* $\Pi' \models_{\mathcal{D}} (h\,\bar{t}_m)\sigma \sqsupseteq t'$ *y* $Sol_{\mathcal{D}}(\Pi') \neq \emptyset$ *deja solo dos casos posibles para* t'.

  En primer lugar, $t' = h\,\overline{t'_m}$ *con* $\Pi' \models_{\mathcal{D}} t_i\sigma \sqsupseteq t'_i$ *para todo* $1 \leq i \leq m$. *En este caso, para cada* $1 \leq i \leq m$ *tenemos que* $(e_i \rightarrow t_i \Leftarrow \Pi) \succcurlyeq_{\mathcal{D}} (e'_i \rightarrow t'_i \Leftarrow \Pi')$ *debido a que* $\Pi' \models_{\mathcal{D}} \Pi\sigma$, $\Pi' \models_{\mathcal{D}} e'_i \sqsupseteq e_i\sigma$ *y* $\Pi' \models_{\mathcal{D}} t_i\sigma \sqsupseteq t'_i$. *Por* hipótesis de inducción, *podemos asumir árboles de prueba* $T'_i : \mathcal{I} \Vdash_{\mathcal{D}} e'_i \rightarrow t'_i \Leftarrow \Pi'$ *con* $\mid T_i \mid \geq \mid T'_i \mid$ $(1 \leq i \leq m)$. *Por lo tanto,* $T' =_{def} \mathbf{DC}(h\,\overline{e'}_m \rightarrow h\,\overline{t'}_m \Leftarrow \Pi', [T'_1, \ldots, T'_m])$ *verifica que* $T' : \mathcal{I} \Vdash_{\mathcal{D}} h\,\overline{e'}_m \rightarrow h\,\overline{t'}_m \Leftarrow \Pi'$ *y que* $\mid T \mid = \sum_{i=1}^{m} \mid T_i \mid \geq \sum_{i=1}^{m} \mid T'_i \mid = \mid T' \mid$.

En segundo lugar, $t' = X \in \mathcal{Var}$ *con* $\Pi' \models_{\mathcal{D}} (h\,\overline{t}_m)\sigma \sqsupseteq X$. *En este caso, para cada* $1 \leq i \leq m$ *tenemos que* $(e_i \rightarrow t_i \Leftarrow \Pi) \succcurlyeq_{\mathcal{D}} (e'_i \rightarrow t_i\sigma \Leftarrow \Pi')$ *debido a que* $\Pi' \models_{\mathcal{D}} \Pi\sigma$, $\Pi' \models_{\mathcal{D}} e'_i \sqsupseteq e_i\sigma$ *y* $\Pi' \models_{\mathcal{D}} t_i\sigma \sqsupseteq t_i\sigma$. *Por* hipótesis de inducción, *podemos asumir árboles de prueba* $T'_i : \mathcal{I} \Vdash_{\mathcal{D}} e'_i \rightarrow t_i\sigma \Leftarrow \Pi'$ *con* $\mid T_i \mid \geq \mid T'_i \mid$ $(1 \leq i \leq m)$. *Puesto que* $\Pi' \models_{\mathcal{D}} (h\,\overline{t}_m)\sigma \sqsupseteq X$, *podemos construir el árbol de prueba* $T' =_{def} \mathbf{IR}(h\,\overline{e'}_m \rightarrow X \Leftarrow \Pi', [T'_1, \ldots, T'_m])$, *el cual verifica que* $T' : \mathcal{I} \Vdash_{\mathcal{D}} h\,\overline{e'}_m \rightarrow X \Leftarrow \Pi'$ *y que* $\mid T \mid = \sum_{i=1}^{m} \mid T_i \mid \geq \sum_{i=1}^{m} \mid T'_i \mid = \mid T' \mid$.

- $T = \mathbf{IR}(h\,\overline{e}_m \rightarrow X \Leftarrow \Pi, [T_1, \ldots, T_m])$. *En este caso, sabemos que* $\varphi = h\,\overline{e}_m \rightarrow X \Leftarrow \Pi$ *con* $h\,\overline{e}_m$ *una expresión rígida y pasiva pero no un patrón,* $\Pi \models_{\mathcal{D}} h\,\overline{t}_m \sqsupseteq X$ *(y de aquí,* $\Pi\sigma \models_{\mathcal{D}} (h\,\overline{t}_m)\sigma \sqsupseteq X\sigma$*),* $T_i : \mathcal{I} \Vdash_{\mathcal{D}} e_i \rightarrow t_i \Leftarrow \Pi$, $\|T_i\| < \|T\|$ $(1 \leq i \leq m)$ *y* $\varphi' = e' \rightarrow t' \Leftarrow \Pi'$ *con* $\Pi' \models_{\mathcal{D}} \Pi\sigma$ *(y de aquí,* $\Pi' \models_{\mathcal{D}} (h\,\overline{t}_m)\sigma \sqsupseteq X\sigma$*),* $\Pi' \models_{\mathcal{D}} e' \sqsupseteq (h\,\overline{e}_m)\sigma$ *y* $\Pi' \models_{\mathcal{D}} X\sigma \sqsupseteq t'$ *(y de aquí también,* $\Pi' \models_{\mathcal{D}} (h\,\overline{t}_m)\sigma \sqsupseteq t'$*). Ahora ya es posible razonar de forma completamente similar a como hemos hecho en el caso anterior.*

- $T = \mathbf{DF}_{\mathcal{I}}(f\,\overline{e}_n\overline{a}_k \rightarrow t \Leftarrow \Pi, [T_1, \ldots, T_n, T_s])$. *Asumamos que* $k > 0$ *(el caso* $k = 0$ *es análogo y más sencillo). En este caso, sabemos que* $\varphi = f\,\overline{e}_n\overline{a}_k \rightarrow t \Leftarrow \Pi$ *con* $t \neq \bot$*, y hay algún c-hecho* $(f\,\overline{t}_n \rightarrow s \Leftarrow \Pi) \in \mathcal{I}$ *y algún patrón parcial* $s \neq \bot$ *tal que* $T_i : \mathcal{I} \Vdash_{\mathcal{D}} e_i \rightarrow t_i \Leftarrow \Pi$, $\|T_i\| < \|T\|$ $(1 \leq i \leq n)$ *y* $T_s : \mathcal{I} \Vdash_{\mathcal{D}} s\,\overline{a}_k \rightarrow t \Leftarrow \Pi$, $\|T_s\| < \|T\|$.

  Puesto que $\varphi \succcurlyeq_{\mathcal{D}} \varphi'$*, sabemos que* $\varphi' = e' \rightarrow t' \Leftarrow \Pi'$ *con* $\Pi' \models_{\mathcal{D}} \Pi\sigma$, $\Pi' \models_{\mathcal{D}} e' \sqsupseteq (f\,\overline{e}_n\overline{a}_k)\sigma$, $\Pi' \models_{\mathcal{D}} t\sigma \sqsupseteq t'$ *y* $t' \neq \bot$ *(si* $t' = \bot$ *entonces* T' *está constituido por un paso* $\mathbf{TI}$ $y \mid T \mid > 0 = \mid T' \mid$*). A partir de* $\Pi' \models_{\mathcal{D}} e' \sqsupseteq (f\,\overline{e}_n\overline{a}_k)\sigma$*, se sigue que* $e' = f\,\overline{e'}_n\overline{a'}_k$ *con* $\Pi' \models_{\mathcal{D}} e'_i \sqsupseteq e_i\sigma$ *para todo* $1 \leq i \leq n$ *y* $\Pi' \models_{\mathcal{D}} a'_j \sqsupseteq a_j\sigma$ *para todo* $1 \leq j \leq k$ *(en caso contrario, para cualquier* $\mu \in Sol_{\mathcal{D}}(\Pi')$ *total podríamos tener que* $e'\mu \sqsupseteq (f\,\overline{e}_n\overline{a}_k)\sigma\mu$ *no es cierto).*

  Usando todas estas condiciones, es fácil comprobar que $(e_i \rightarrow t_i \Leftarrow \Pi) \succcurlyeq_{\mathcal{D}} (e'_i \rightarrow t_i\sigma \Leftarrow \Pi')$ *para todo* $1 \leq i \leq n$ *y* $(s\,\overline{a}_k \rightarrow t \Leftarrow \Pi) \succcurlyeq_{\mathcal{D}} (s\sigma\,\overline{a'}_k \rightarrow t' \Leftarrow \Pi')$*. Por* hipótesis de inducción *(aplicada a* T_i, T_s*), tenemos* $T'_i : \mathcal{I} \Vdash_{\mathcal{D}} e'_i \rightarrow t_i\sigma \Leftarrow \Pi'$, $\mid T_i \mid \geq \mid T'_i \mid$ $(1 \leq i \leq n)$ *y que* $T'_s : \mathcal{I} \Vdash_{\mathcal{D}} s\sigma\,\overline{a'}_k \rightarrow t' \Leftarrow \Pi'$, $\mid T_s \mid \geq \mid T'_s \mid$.

  Puesto que $(f\,\overline{t}_n \rightarrow s \Leftarrow \Pi) \in \mathcal{I}$ *y* $(f\,\overline{t}_n \rightarrow s \Leftarrow \Pi) \succcurlyeq_{\mathcal{D}} (f\,\overline{t_n\sigma} \rightarrow s\sigma \Leftarrow \Pi')$*, esto implica que* $(f\,\overline{t_n\sigma} \rightarrow s\sigma \Leftarrow \Pi') \in \mathcal{I}$ *por definición de c-interpretación, con* $s\sigma \neq \bot$ *un patrón parcial (si* $s\sigma = \bot$ *entonces el patrón* s *ha de ser una variable y la deducción no es posible en el cálculo semántico debido a que* $\mathcal{I} \Vdash_{\mathcal{D}} s\,\overline{a}_k \rightarrow t \Leftarrow \Pi$ *con* $k > 0$ *y* $t \neq \bot$*). Podemos construir el árbol de prueba* $T' =_{def} \mathbf{DF}_{\mathcal{I}}(f\,\overline{e'}_n\overline{a'}_k \rightarrow t' \Leftarrow \Pi', [T'_1, \ldots, T'_n, T'_s])$*, el cual verifica que* $T' : \mathcal{I} \Vdash_{\mathcal{D}} f\,\overline{e'}_n\overline{a'}_k \rightarrow t' \Leftarrow \Pi'$ *y que* $\mid T \mid = 1 + \sum_{i=1}^{m} \mid T_i \mid + \mid T_s \mid \geq 1 + \sum_{i=1}^{m} \mid T'_i \mid + \mid T'_s \mid = \mid T' \mid$.

- $T = \mathbf{PF}(p\,\overline{e}_n \to t \Leftarrow \Pi, [T_1, \ldots, T_n])$. *En este caso, sabemos que* $\varphi = p\,\overline{e}_n \to t \Leftarrow \Pi$ *y que* $T_i : \mathcal{I} \Vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$, $\|T_i\| < \|T\|$ $(1 \leq i \leq n)$ *con* $\Pi \models_{\mathcal{D}} p\,\overline{t}_n \to t$.

  Puesto que $\varphi \succcurlyeq_{\mathcal{D}} \varphi'$ *y* $Sol_{\mathcal{D}}(\Pi') \neq \emptyset$, *ha de darse el caso de que* $\varphi' = p\,\overline{e'}_n \to t' \Leftarrow \Pi'$ *con* $\Pi' \models_{\mathcal{D}} \Pi\sigma$, $\Pi' \models_{\mathcal{D}} e'_i \sqsupseteq e_i\sigma$ *para todo* $1 \leq i \leq n$, *y* $\Pi' \models_{\mathcal{D}} t\sigma \sqsupseteq t'$. *A partir de las condiciones anteriores, podemos deducir que* $\Pi\sigma \models_{\mathcal{D}} (p\,\overline{t}_n)\sigma \to t\sigma$, *y de aquí que* $\Pi' \models_{\mathcal{D}} (p\,\overline{t}_n)\sigma \to t'$. *También observamos que* $(e_i \to t_i \Leftarrow \Pi) \succcurlyeq_{\mathcal{D}} (e'_i \to t_i\sigma \Leftarrow \Pi')$ $(1 \leq i \leq n)$. *Por* hipótesis de inducción, *tenemos árboles de prueba* $T'_i : \mathcal{I} \Vdash_{\mathcal{D}} e'_i \to t_i\sigma \Leftarrow \Pi'$ *para los que* $|\,T_i\,| \geq |\,T'_i\,|$ $(1 \leq i \leq n)$.

  Puesto que $\Pi' \models_{\mathcal{D}} (p\,\overline{t}_n)\sigma \to t'$, *podemos construir el árbol de prueba* $T' =_{def} \mathbf{PF}(p\,\overline{e'}_n \to t' \Leftarrow \Pi', [T'_1, \ldots, T'_n])$, *el cual verifica que* $T' : \mathcal{I} \Vdash_{\mathcal{D}} p\,\overline{e'}_n \to t' \Leftarrow \Pi'$ *con* $|\,T\,| = 1 + \sum_{i=1}^{m} |\,T_i\,| \geq 1 + \sum_{i=1}^{m} |\,T'_i\,| = |\,T'\,|$.

- $T = \mathbf{AC}(p\,\overline{e}_n \to !\,t \Leftarrow \Pi, [T_1, \ldots, T_n])$. *Similar al caso de la regla* **PF**.

(6) ***Propiedad de Conservación****. Sea* φ *un c-hecho de la forma* $f\,\overline{t}_n \to t \Leftarrow \Pi$. *Probamos en primer lugar la parte del "si". Teniendo en cuenta que* $T_i : \mathcal{I} \Vdash_{\mathcal{D}} t_i \to t_i \Leftarrow \Pi$ *son árboles de prueba simples para todo* $1 \leq i \leq n$ *por la* Propiedad de Aproximación *(a partir de la definición del orden de aproximación,* $t_i \sqsupseteq t_i$ *siempre se cumple para todo* $t_i \in Pat_{\mathcal{D}}$, *y por tanto* $\Pi \models_{\mathcal{D}} t_i \sqsupseteq t_i$ *también se cumple para todo* $1 \leq i \leq n$*), podemos también suponer que* $t \neq \bot$ *(el caso* $\mathcal{I} \Vdash_{\mathcal{D}} f\,\overline{t}_n \to \bot \Leftarrow \Pi$ *es trivial por* **TI***) y construir directamente la deducción* $\mathbf{DF}_{\mathcal{I}}(f\,\overline{t}_n \to t \Leftarrow \Pi, [T_1, \ldots, T_n])$ *usando la hipótesis inicial* $(f\,\overline{t}_n \to t \Leftarrow \Pi) \in \mathcal{I}$.

Probamos ahora la parte correspondiente al "solo si". En primer lugar, si suponemos que $t = \bot$ *o que* $Sol_{\mathcal{D}}(\Pi) = \emptyset$, *directamente* $(f\,\overline{t}_n \to t \Leftarrow \Pi) \in \mathcal{I}$ *por la definición de c-interpretación. En otro caso, por hipótesis inicial* $T : \mathcal{I} \Vdash_{\mathcal{D}} f\,\overline{t}_n \to t \Leftarrow \Pi$ *ha de ser de la forma* $T = \mathbf{DF}_{\mathcal{I}}(f\,\overline{t}_n \to t \Leftarrow \Pi, [T_1, \ldots, T_n])$, *donde* $T_i : \mathcal{I} \Vdash_{\mathcal{D}} t_i \to t'_i \Leftarrow \Pi$ $(1 \leq i \leq n)$ *son árboles de prueba simples tales que* $(f\,\overline{t'}_n \to t \Leftarrow \Pi) \in \mathcal{I}$ *con* $t'_1, \ldots, t'_n \in Pat_{\mathcal{D}}$. *En este contexto, se tiene que* $\Pi \models_{\mathcal{D}} t'_i \sqsubseteq t_i$ *para todo* $1 \leq i \leq n$ *usando la* Propiedad de Aproximación. *Se sigue que* $(f\,\overline{t'}_n \to t \Leftarrow \Pi) \succcurlyeq_{\mathcal{D}} (f\,\overline{t}_n \to t \Leftarrow \Pi)$, *y consecuentemente* $(f\,\overline{t}_n \to t \Leftarrow \Pi) \in \mathcal{I}$ *debido a que* $\mathcal{I}$ *es cerrado bajo* $\mathcal{D}$*-implicación por definición de c-interpretación.*

(7) ***Propiedad de Cierre****. La parte del "si" se sigue de la* Propiedad de Extensión, *puesto que* $[\mathcal{I}]_{\mathcal{D}} \subseteq \mathcal{I}$. *Con el fin de probar la parte del "solo si", introducimos dos notaciones auxiliares:*

- *Una c-sentencia* $\varphi = e \to^{?} t \Leftarrow \Pi$ *se denomina* c-cerrada *si y sólo si* Π *es cerrado. En particular, cualquier c-sentencia cerrada es c-cerrada.*

- *El* menor cierre *de una c-sentencia c-cerrada* $\varphi = e \rightarrow^{?} t \Leftarrow \Pi$ *es la c-sentencia cerrada* $\varphi^{\perp} = e^{\perp} \rightarrow^{?} t^{\perp} \Leftarrow \Pi$ *que se obtiene a partir de* φ *mediante la sustitución por* $\perp$ *en el lugar de todas las apariciones de variables.*

Obviamente, la parte del "solo si" se sigue a partir de la siguiente propiedad más general:

GP *Para cualquier c-sentencia c-cerrada* φ, $T : \mathcal{I} \Vdash_{\mathcal{D}} \varphi$ *implica que* $T' : [\mathcal{I}]_{\mathcal{D}} \Vdash_{\mathcal{D}} \varphi^{\perp}$ *con árbol de prueba* T' *tal que* $\|T'\| \leq \|T\|$.

Vamos a probar **GP** *razonando por inducción sobre* $\|T\|$. *Distinguimos casos de acuerdo con la regla de inferencia aplicada en la raíz de* T. *En el resto de esta demostración, la notación asumida para* φ *en cada caso es la misma que ha sido usada en las reglas de inferencia dadas en la Definición 10.*

TI *En este caso,* $\varphi^{\perp}$ *es también una c-sentencia trivial y podemos tomar* $T' = \mathbf{TI}(\varphi^{\perp}, [\,])$.

RR *Si* $t \in \mathcal{V}ar$, *entonces* $\varphi^{\perp}$ *es una c-sentencia trivial y podemos tomar* $T' = \mathbf{TI}(\varphi^{\perp}, [\,])$. *En caso contrario* $t \in \mathcal{B}^{\mathcal{D}}$, $\varphi^{\perp} = \varphi$, *y podemos tomar* $T' = T$.

SP *Si* $t \in \mathcal{V}ar$, *entonces* $\varphi^{\perp}$ *es una c-sentencia trivial y podemos tomar* $T' = \mathbf{TI}(\varphi^{\perp}, [\,])$. *Si* $t \notin \mathcal{V}ar$, *entonces* $\varphi = X \rightarrow t \Leftarrow \Pi$ *para algún* $X \in \mathcal{V}ar$, *y* $\Pi \models_{\mathcal{D}} X \sqsupseteq t$. *Puesto que* X *no aparece en* Π, *la c-sentencia* φ *ha de ser trivial (de otro modo podríamos tener* $\Pi \models_{\mathcal{D}} \perp \sqsupseteq t$, $Sat_{\mathcal{D}}(\Pi)$, $t \notin \mathcal{V}ar$, $t \neq \perp$, *lo que es imposible). En este caso,* $\varphi^{\perp}$ *es también una c-sentencia trivial y podemos tomar* $T' = \mathbf{TI}(\varphi^{\perp}, [\,])$.

DC *En este caso tenemos que* $T = \mathbf{DC}(h\,\overline{e}_m \rightarrow h\,\overline{t}_m \Leftarrow \Pi, [T_1, \ldots, T_m])$, *donde* $T_i : \mathcal{I} \Vdash_{\mathcal{D}} e_i \rightarrow t_i \Leftarrow \Pi$ *para todo* $1 \leq i \leq m$. *Por* hipótesis de inducción, *existe* $\|T_i'\| \leq \|T_i\|$ *tal que* $T_i' : [\mathcal{I}]_{\mathcal{D}} \Vdash_{\mathcal{D}} e_i^{\perp} \rightarrow t_i^{\perp} \Leftarrow \Pi$ *para todo* $1 \leq i \leq m$. *Por tanto,* $T' : [\mathcal{I}]_{\mathcal{D}} \Vdash_{\mathcal{D}} \varphi^{\perp}$ *con* $T' = \mathbf{DC}(h\,\overline{e_m^{\perp}} \rightarrow h\,\overline{t_m^{\perp}} \Leftarrow \Pi, [T_1', \ldots, T_m'])$.

IR *En este caso* $\varphi^{\perp}$ *es obviamente una c-sentencia trivial y podemos tomar* $T' = \mathbf{TI}(\varphi^{\perp}, [\,])$.

DF$_{\mathcal{I}}$ *Asumamos que* $k > 0$ *(la demostración para el caso* $k = 0$ *es similar y más simple). En este caso* $T = \mathbf{DF}_{\mathcal{I}}(f\,\overline{e}_n\overline{a}_k \rightarrow t \Leftarrow \Pi, [T_1, \ldots, T_n, T_{n+1}])$, *donde* $T_i : \mathcal{I} \Vdash_{\mathcal{D}} e_i \rightarrow t_i \Leftarrow \Pi$ *para todo* $1 \leq i \leq n$, $T_{n+1} : \mathcal{I} \Vdash_{\mathcal{D}} s\,\overline{a}_k \rightarrow t \Leftarrow \Pi$ *y* $\overline{t}_n$, s *son patrones tales que* $(f\,\overline{t}_n \rightarrow s \Leftarrow \Pi) \in \mathcal{I}$. *Puesto que* $\mathcal{I}$ *es cerrado bajo* $\mathcal{D}$*-implicación, en particular bajo sustitución, podemos asumir que* $(f\,\overline{t_n^{\perp}} \rightarrow s^{\perp} \Leftarrow \Pi) \in \mathcal{I}$, *y de aquí también* $(f\,\overline{t_n^{\perp}} \rightarrow s^{\perp} \Leftarrow \Pi) \in [\mathcal{I}]_{\mathcal{D}}$, *puesto que este es un c-hecho cerrado. Por* hipótesis de inducción, *existe* $\|T_i'\| \leq \|T_i\|$ *tal que* $T_i' : [\mathcal{I}]_{\mathcal{D}} \Vdash_{\mathcal{D}} e_i^{\perp} \rightarrow t_i^{\perp} \Leftarrow \Pi$ *para todo* $1 \leq i \leq n$,

$T'_{n+1} : [\mathcal{I}]_{\mathcal{D}} \Vdash_{\mathcal{D}} s^{\perp}\,\overline{a_k^{\perp}} \rightarrow t^{\perp} \Leftarrow \Pi$. *Por lo tanto,* $T' : [\mathcal{I}]_{\mathcal{D}} \Vdash_{\mathcal{D}} \varphi^{\perp}$ *con* $T' = \mathbf{DF}_{\mathcal{I}}(f\,\overline{e_n^{\perp}}\,\overline{a_k^{\perp}} \rightarrow t^{\perp} \Leftarrow \Pi, [T'_1, \ldots, T'_n, T'_{n+1}])$.

PF *En este caso,* $T = \mathbf{PF}(p\,\bar{e}_n \rightarrow t \Leftarrow \Pi, [T_1, \ldots, T_n])$, *donde* $T_i : \mathcal{I} \Vdash_{\mathcal{D}} e_i \rightarrow t_i \Leftarrow \Pi$ *para todo* $1 \leq i \leq n$ *y* $\bar{t}_n$ *son patrones tales que* $\Pi \models_{\mathcal{D}} p\,\bar{t}_n \rightarrow t$. *Puesto que* Π *es cerrado,* $\Pi \models_{\mathcal{D}} p\,\bar{t}_n \rightarrow t$ *trivialmente implica que* $\Pi \models_{\mathcal{D}} p\,\overline{t_n^{\perp}} \rightarrow t^{\perp}$. *Más aún, por* hipótesis de inducción, *ha de existir* $\|T'_i\| \leq \|T_i\|$ *tal que* $T'_i : [\mathcal{I}]_{\mathcal{D}} \Vdash_{\mathcal{D}} e_i^{\perp} \rightarrow t_i^{\perp} \Leftarrow \Pi$ *para todo* $1 \leq i \leq n$. *Por lo tanto,* $T' : [\mathcal{I}]_{\mathcal{D}} \Vdash_{\mathcal{D}} \varphi^{\perp}$ *con* $T' = \mathbf{PF}(p\,\overline{e_n^{\perp}} \rightarrow t^{\perp} \Leftarrow \Pi, [T'_1, \ldots, T'_n])$.

AC *En este caso,* $T = \mathbf{AC}(p\,\bar{e}_n \rightarrow\!! t \Leftarrow \Pi, [T_1, \ldots, T_n])$, *donde* $T_i : \mathcal{I} \Vdash_{\mathcal{D}} e_i \rightarrow t_i \Leftarrow \Pi$ *para todo* $1 \leq i \leq n$ *y* $\bar{t}_n$ *son patrones tales que* $\Pi \models_{\mathcal{D}} p\,\bar{t}_n \rightarrow\!! t$. *Por* hipótesis de inducción *existe* $\|T'_i\| \leq \|T_i\|$ *tal que* $T'_i : [\mathcal{I}]_{\mathcal{D}} \Vdash_{\mathcal{D}} e_i^{\perp} \rightarrow t_i^{\perp} \Leftarrow \Pi$ *para todo* $1 \leq i \leq n$. *Puesto que* Π *es cerrado,* $\Pi \models_{\mathcal{D}} p\,\bar{t}_n \rightarrow\!! t$ *trivialmente implica que* $\Pi \models_{\mathcal{D}} p\,\overline{t_n^{\perp}} \rightarrow\!! t^{\perp}$. *Esto sólo puede ser si* $t^{\perp}$ *es total. Por lo tanto,* t *ha de ser cerrado,* $t^{\perp} = t$, *y* $T' : [\mathcal{I}]_{\mathcal{D}} \Vdash_{\mathcal{D}} \varphi^{\perp}$ *con* $T' = \mathbf{AC}(p\,\overline{e_n^{\perp}} \rightarrow\!! t \Leftarrow \Pi, [T'_1, \ldots, T'_n])$.

□

A.2.2. Propiedades de los operadores de transformación

Demostramos las propiedades del operador fuerte de transformación de interpretaciones $ST_{\mathcal{P}}$; las propiedades correspondientes a $WT_{\mathcal{P}}$ pueden ser demostradas de manera muy similar.

Demostración 30 (demostración de la Proposición 3) *En primer lugar, observamos que* $ST_{\mathcal{P}} : \mathbb{I}_{\mathcal{D}} \rightarrow \mathbb{I}_{\mathcal{D}}$ *es una función bien definida, debido a que para cada* $\mathcal{I} \in \mathbb{I}_{\mathcal{D}}$ *la imagen* $ST_{\mathcal{P}}(\mathcal{I})$ *ha sido definida como* $cl_{\mathcal{D}}(preST_{\mathcal{P}}(\mathcal{I}))$, *y es por tanto, una c-interpretación.*

Una observación minuciosa de la Definición 13 revela que $\mathcal{I} \models^{s}_{\mathcal{D}} \mathcal{P}$ *si y sólo si* $preST_{\mathcal{P}}(\mathcal{I}) \subseteq \mathcal{I}$. *Por otra parte,* $preST_{\mathcal{P}}(\mathcal{I}) \subseteq \mathcal{I}$ *si y sólo si* $ST_{\mathcal{P}}(\mathcal{I}) = cl_{\mathcal{D}}(preST_{\mathcal{P}}(\mathcal{I})) \subseteq \mathcal{I}$, *puesto que* $\mathcal{I}$ *es cerrada bajo* $cl_{\mathcal{D}}$. *Por tanto,* $\mathcal{I} \models^{s}_{\mathcal{D}} \mathcal{P}$ *si y sólo si* $ST_{\mathcal{P}}(\mathcal{I}) \subseteq \mathcal{I}$, *es decir, los modelos fuertes de* $\mathcal{P}$ *son los pre-puntos fijos de* $ST_{\mathcal{P}}$.

Finalmente, el hecho de que $ST_{\mathcal{P}}$ *sea continua se sigue de los dos siguientes apartados que se muestran a continuación:*

(1) $ST_{\mathcal{P}}$ *es monótona:*

Asumamos $\mathcal{I}, \mathcal{J} \in \mathbb{I}_{\mathcal{D}}$ *tales que* $\mathcal{I} \subseteq \mathcal{J}$. *Entonces,* $preST_{\mathcal{P}}(\mathcal{I}) \subseteq preST_{\mathcal{P}}(\mathcal{J})$ *es una consecuencia fácil de la* Propiedad de Extensión *dada en el Lema 5, por lo que podemos concluir que* $ST_{\mathcal{P}}(\mathcal{I}) \subseteq ST_{\mathcal{P}}(\mathcal{J})$.

(2) $ST_{\mathcal{P}}$ *preserva las menores cotas superiores de conjuntos dirigidos no vacíos:*

Asumiendo un conjunto dirigido no vacío $\mathfrak{I} \subseteq \mathbb{I}_{\mathcal{D}}$*, tenemos que probar que* $ST_{\mathcal{P}}(\sqcup \mathfrak{I}) = \sqcup ST_{\mathcal{P}}(\mathfrak{I})$*. La inclusión* $ST_{\mathcal{P}}(\sqcup \mathfrak{I}) \supseteq \sqcup ST_{\mathcal{P}}(\mathfrak{I})$ *se cumple debido a que* $ST_{\mathcal{P}}$ *es monótona. Puesto que* $\mathfrak{I}$ *no es vacía, la inclusión contraria puede ser reescrita como* $ST_{\mathcal{P}}(\bigcup \mathfrak{I}) \subseteq \bigcup ST_{\mathcal{P}}(\mathfrak{I})$*, la cual es una consecuencia de* $preST_{\mathcal{P}}(\bigcup \mathfrak{I}) \subseteq \bigcup ST_{\mathcal{P}}(\mathfrak{I})$*. Con el fin de probar esta última inclusión, asumamos un c-hecho arbitrariamente fijado* φ *perteneciente al conjunto* $preST_{\mathcal{P}}(\bigcup \mathfrak{I})$*. Debido al modo en el que este conjunto ha sido definido,* φ *llega a ser miembro suyo gracias a la existencia de una cantidad finita de otras c-sentencias* φ_i *tales que* $\bigcup \mathfrak{I} \Vdash_{\mathcal{D}} \varphi_i$*. Por tanto, por la* Propiedad de Compacidad *dada en el Lema 5, ha de existir algún conjunto finito de c-hechos* $\mathcal{I}_0 \subseteq \bigcup \mathfrak{I}$ *tal que* $\varphi \in preST_{\mathcal{P}}(cl_{\mathcal{D}}(\mathcal{I}_0))$*. Puesto que* $\mathfrak{I}$ *es dirigido, ha de existir también algún* $\mathcal{I} \in \mathfrak{I}$ *tal que* $\mathcal{I}_0 \subseteq \mathcal{I}$*. Puesto que además* $\mathcal{I}$ *es cerrada bajo* $cl_{\mathcal{D}}$*, obtenemos que* $cl_{\mathcal{D}}(\mathcal{I}_0) \subseteq \mathcal{I}$*, y por tanto (usando la* Propiedad de Extensión *dada en el Lema 5)* $\varphi \in preST_{\mathcal{P}}(\mathcal{I}) \subseteq \bigcup ST_{\mathcal{P}}(\mathfrak{I})$*.*

□

A.2.3. Relación entre los operadores de transformación

Probamos en esta subsección la relación existente entre los dos operadores de transformación sobre c-interpretaciones definidos en la Subsección 3.2.2.

Demostración 31 (demostración de la Proposición 4) *Probamos las dos inclusiones (i) y (ii) que se muestran a continuación:*

(i) $WT_{\mathcal{P}}([\mathcal{I}]_{\mathcal{D}}) \subseteq [ST_{\mathcal{P}}(\mathcal{I})]_{\mathcal{D}}$

Puesto que $[ST_{\mathcal{P}}(\mathcal{I})]_{\mathcal{D}}$ *es cerrado bajo* $cl_{\mathcal{D}}$*, es suficiente con demostrar que se verifica* $preWT_{\mathcal{P}}([\mathcal{I}]_{\mathcal{D}}) \subseteq [ST_{\mathcal{P}}(\mathcal{I})]_{\mathcal{D}}$*. Asumamos cualquier* $\varphi' \in preWT_{\mathcal{P}}([\mathcal{I}]_{\mathcal{D}})$*. Debido a la Definición 15 (2), sabemos que* $\varphi' = f\,\overline{t'}_n \rightarrow t'$*, donde* $\overline{t'}_n$*,* t' *son cerrados y existe*

(1) $(f\,\overline{t}_n \rightarrow r \Leftarrow P \,\square\, \Delta) \in \mathcal{P}$*,* $\eta \in GSub_{\mathcal{D}}$

tal que

(2) $(f\,\overline{t}_n \rightarrow r \Leftarrow P \,\square\, \Delta)\eta$ *es cerrada,*
(3) $\overline{t}_n\eta = \overline{t'}_n$*,*
(4) $[\mathcal{I}]_{\mathcal{D}} \Vdash_{\mathcal{D}} (P \,\square\, \Delta)\eta$*,* $[\mathcal{I}]_{\mathcal{D}} \Vdash_{\mathcal{D}} r\eta \rightarrow t'$*.*

Por la Propiedad de Extensión *dada en el Lema 5, la condición (4) implica*

(5) $\mathcal{I} \Vdash_{\mathcal{D}} (P \,\square\, \Delta)\eta$*,* $\mathcal{I} \Vdash_{\mathcal{D}} r\eta \rightarrow t'$

De las condiciones (1), (3) y (5) y la Definición 15 (1), claramente obtenemos que $\varphi' \in preST_{\mathcal{P}}(\mathcal{I}) \subseteq ST_{\mathcal{P}}(\mathcal{I})$*. Puesto que* φ' *es cerrado, podemos inferir que* $\varphi' \in gk_{\mathcal{D}}(ST_{\mathcal{P}}(\mathcal{I})) \subseteq [ST_{\mathcal{P}}(\mathcal{I})]_{\mathcal{D}}$ *como pretendíamos.*

(ii) $[ST_{\mathcal{P}}(\mathcal{I})]_{\mathcal{D}} \subseteq WT_{\mathcal{P}}([\mathcal{I}]_{\mathcal{D}})$

Puesto que $WT_{\mathcal{P}}([\mathcal{I}]_{\mathcal{D}})$ *es cerrada bajo* $cl_{\mathcal{D}}$*, es suficiente con probar que todo c-hecho no trivial perteneciente a* $gk_{\mathcal{D}}(ST_{\mathcal{P}}(\mathcal{I}))$ *también pertenece a* $WT_{\mathcal{P}}([\mathcal{I}]_{\mathcal{D}})$*. Asumamos cualquier c-hecho cerrado no trivial* $\varphi' = f\,\overline{t'}_n \rightarrow t' \Leftarrow \Pi' \in ST_{\mathcal{P}}(\mathcal{I})$*. Por la Definición 15 (1), han de existir*

(6) $(f\,\bar{t}_n \rightarrow r \Leftarrow P \,\square\, \Delta) \in \mathcal{P}, \theta \in Sub_{\mathcal{D}}$,
(7) $\Pi \subseteq PCon_{\mathcal{D}}, t \in Pat_{\mathcal{D}}$,

tales que

(8) $\mathcal{I} \Vdash_{\mathcal{D}} (P \,\square\, \Delta)\theta \Leftarrow \Pi, \mathcal{I} \Vdash_{\mathcal{D}} r\theta \rightarrow t \Leftarrow \Pi$,
(9) $f\,\bar{t}_n\theta \rightarrow t \Leftarrow \Pi \succcurlyeq_{\mathcal{D}} f\,\overline{t'}_n \rightarrow t' \Leftarrow \Pi'$.

Utilizando ahora la condición (9) y la definición de $\mathcal{D}$-implicación *(ver la Definición 8), hay algún* $\sigma \in Sub_{\mathcal{D}}$ *tal que*

(10) $\Pi' \models_{\mathcal{D}} \Pi\sigma, \Pi' \models_{\mathcal{D}} \overline{t'}_n \sqsupseteq \bar{t}_n\theta\sigma, \Pi' \models_{\mathcal{D}} t' \sqsubseteq t\sigma$

Puesto que φ' *es no trivial y cerrado, sabemos que* Π' *es cerrado y* $\mathcal{D}$*-satisfactible, y de este modo,* $\mathcal{D}$*-válido. Por esta razón, la condición (10) puede ser reescrita como sigue:*

(11) $\models_{\mathcal{D}} \Pi\sigma, \models_{\mathcal{D}} \overline{t'}_n \sqsupseteq \bar{t}_n\theta\sigma, \models_{\mathcal{D}} t' \sqsubseteq t\sigma$

A partir de la primera condición en (11) y de la definición de $\mathcal{D}$-implicación*, sabemos que:*

(12) $(P \,\square\, \Delta)\theta \Leftarrow \Pi \succcurlyeq_{\mathcal{D}} (P \,\square\, \Delta)\theta\sigma, r\theta \rightarrow t \Leftarrow \Pi \succcurlyeq_{\mathcal{D}} r\theta\sigma \rightarrow t\sigma$

A partir de las condiciones (8) y (12) y de la Propiedad de Implicación *dada en el Lema 5, podemos inferir*

(13) $\mathcal{I} \Vdash_{\mathcal{D}} (P \,\square\, \Delta)\theta\sigma, \mathcal{I} \Vdash_{\mathcal{D}} r\theta\sigma \rightarrow t\sigma$

Consideramos ahora $\eta \in GSub_{\mathcal{D}}$ *elegida de tal modo que*

(14) $(P \,\square\, \Delta)\eta = ((P \,\square\, \Delta)\theta\sigma)^{\perp}, r\eta \rightarrow t\eta = (r\theta\sigma)^{\perp} \rightarrow (t\sigma)^{\perp}$

donde la notación $(\ldots)^{\perp}$ *indica la sustitución por* $\perp$ *en todas las apariciones de variables. A partir de (13), (14) y de la* Propiedad de Cierre *generalizada* **GP** *usada en la demostración del Lema 5 (7), se tiene:*

(15) $[\mathcal{I}]_{\mathcal{D}} \Vdash_{\mathcal{D}} (P \,\square\, \Delta)\eta, [\mathcal{I}]_{\mathcal{D}} \Vdash_{\mathcal{D}} r\eta \rightarrow t\eta$

A partir de las condiciones (6) y (15) y de la Definición 15 (2), obtenemos

(16) $(f\,\bar{t}_n\eta \rightarrow t\eta) \in preWT_{\mathcal{P}}([\mathcal{I}]_{\mathcal{D}})$

Por otra parte, a partir de la segunda y tercera condición en (11), la elección de η *y el hecho de que* $\overline{t'}_n$ *y* t' *sean básicos, se tiene*

(17) $\models_{\mathcal{D}} \overline{t'}_n \sqsupseteq \bar{t}_n\eta, \models_{\mathcal{D}} t' \sqsubseteq t\eta$

Puesto que Π' *es básico y* $\mathcal{D}$*-válido, la condición (17) asegura que*

(18) $(f\,\overline{t}_n\eta \to t\eta) \succcurlyeq_{\mathcal{D}} (f\,\overline{t'}_n \to t' \Leftarrow \Pi') = \varphi'$

A partir de las condiciones (16) y (18) y de la Propiedad de Implicación *dada en el Lema 5, concluimos finalmente que*

$\varphi' \in cl_{\mathcal{D}}(preWT_{\mathcal{P}}([\mathcal{I}]_{\mathcal{D}})) = WT_{\mathcal{P}}([\mathcal{I}]_{\mathcal{D}})$

como se pretendía demostrar.

□

A.2.4. Propiedades del cálculo *CRWL*($\mathcal{D}$)

Probamos ahora las propiedades básicas que satisface el cálculo para la reescritura con restricciones *CRWL*($\mathcal{D}$) presentado en la Definición 16 del Capítulo 3. La demostración es similar a la del Lema 5.

Demostración 32 (Demostración del Lema 7)
En (1) la propiedad se satisface trivialmente debido a que sólo usamos a lo sumo una instancia de una regla de programa de $\mathcal{P}$ en cada paso de la derivación.

(2) Es obvia usando el hecho de que $\mathcal{P} \subseteq \mathcal{P}'$ si alguna instancia de regla de programa de $\mathcal{P}$ es necesaria en la derivación.

(3) y (4) se demuestran del mismo modo que en las propiedades análogas del cálculo semántico.
Finalmente, en (5) podemos usar de nuevo inducción sobre $\|T\|$ para demostrar la existencia del árbol de prueba T' para $\mathcal{P} \vdash_{\mathcal{D}} \varphi'$ tal que $\mid T \mid \geq \mid T' \mid$. Ahora, el único caso diferente está en la aplicación de la regla $\mathbf{DF}_{\mathcal{P}}$.

Si $T = \mathbf{DF}_{\mathcal{P}}(f\,\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi, [T_1, \ldots, T_n, \overline{T}, T_r, T_s])$ y $k > 0$ (el caso $k = 0$ es análogo e incluso más sencillo), sabemos que $\varphi = f\,\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi$ con $t \neq \bot$, y hay alguna instancia de regla de programa $(f\,\overline{t}_n \to r \Leftarrow P \,\square\, \Delta) \in [\mathcal{P}]_\bot$ y algún patrón parcial $s \neq \bot$ tales que $T_i : \mathcal{P} \vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$, $\|T_i\| < \|T\|$ $(1 \leq i \leq n)$, $\overline{T} : \mathcal{P} \vdash_{\mathcal{D}} P \,\square\, \Delta \Leftarrow \Pi$, $\|\overline{T}\| < \|T\|$, $T_r : \mathcal{P} \vdash_{\mathcal{D}} r \to s \Leftarrow \Pi$, $\|T_r\| < \|T\|$ y $T_s : \mathcal{P} \vdash_{\mathcal{D}} s\,\overline{a}_k \to t \Leftarrow \Pi$, $\|T_s\| < \|T\|$.

Puesto que $\varphi \succcurlyeq_{\mathcal{D}} \varphi'$, sabemos que $\varphi' = e' \to t' \Leftarrow \Pi'$ con $\Pi' \models_{\mathcal{D}} \Pi\sigma$, $\Pi' \models_{\mathcal{D}} e' \sqsupseteq (f\,\overline{e}_n\overline{a}_k)\sigma$, $\Pi' \models_{\mathcal{D}} t\sigma \sqsupseteq t'$ y $t' \neq \bot$ (si $t' = \bot$ entonces T' está formado por un paso **TI** *y $\mid T \mid > 0 = \mid T' \mid$).*

A partir de $\Pi' \models_{\mathcal{D}} e' \sqsupseteq (f\,\overline{e}_n\overline{a}_k)\sigma$, se sigue que $e' = f\,\overline{e'}_n\overline{a'}_k$ con $\Pi' \models_{\mathcal{D}} e'_i \sqsupseteq e_i\sigma$ para todo $1 \leq i \leq n$ y $\Pi' \models_{\mathcal{D}} a'_j \sqsupseteq a_j\sigma$ para todo $1 \leq j \leq k$ (en caso contrario, para cualquier $\mu \in Sol_{\mathcal{D}}(\Pi')$ total, podríamos tener que $e'\mu \sqsupseteq (f\,\overline{e}_n\overline{a}_k)\sigma\mu$ no es cierto).

Usando las anteriores condiciones, es fácil comprobar que $(e_i \to t_i \Leftarrow \Pi) \succcurlyeq_{\mathcal{D}} (e'_i \to t_i\sigma \Leftarrow \Pi')$ para todo $1 \leq i \leq n$, $(P \,\square\, \Delta \Leftarrow \Pi) \succcurlyeq_{\mathcal{D}} ((P \,\square\, \Delta)\sigma \Leftarrow \Pi')$, $(r \to s \Leftarrow \Pi) \succcurlyeq_{\mathcal{D}} (r\sigma \to s\sigma \Leftarrow \Pi')$ y que $(s\,\overline{a}_k \to t \Leftarrow \Pi) \succcurlyeq_{\mathcal{D}} (s\sigma\,\overline{a'}_k \to t' \Leftarrow \Pi')$.

Por hipótesis de inducción *(aplicada a T_i, $\overline{T}$, T_r, T_s), se tiene $T'_i : \mathcal{P} \vdash_{\mathcal{D}} e'_i \to$*

$t_i\sigma \Leftarrow \Pi'$, $\mid T_i \mid \geq \mid T'_i \mid$ $(1 \leq i \leq n)$, $\overline{T'} : \mathcal{P} \vdash_{\mathcal{D}} (P \square \Delta)\sigma \Leftarrow \Pi'$, $\mid \overline{T} \mid \geq \mid \overline{T'} \mid$, $T'_r : \mathcal{P} \vdash_{\mathcal{D}} r\sigma \rightarrow s\sigma \Leftarrow \Pi'$, $\mid T_r \mid \geq \mid T'_r \mid$, y $T'_s : \mathcal{P} \vdash_{\mathcal{D}} s\sigma\, \overline{a'}_k \rightarrow t' \Leftarrow \Pi'$, $\mid T_s \mid \geq \mid T'_s \mid$.

Puesto que $(f\, \bar{t}_n \rightarrow r \Leftarrow P \square \Delta)\sigma \in [\mathcal{P}]_\perp$ *y* $s\sigma \neq \perp$ *es un patrón parcial (si* $s\sigma = \perp$ *entonces el patrón* s *ha de ser una variable y la deducción no es posible en el cálculo de reescritura con restricciones porque* $\mathcal{P} \vdash_{\mathcal{D}} s\, \bar{a}_k \rightarrow t \Leftarrow \Pi$ *con* $k > 0$ *y* $t \neq \perp$*), podemos construir el árbol de prueba* $T' =_{def} \mathbf{DF}_{\mathcal{P}}(f\, \overline{e'}_n \overline{a'}_k \rightarrow t' \Leftarrow \Pi', [T'_1, \ldots, T'_n, \overline{T'}, T'_r, T'_s])$*, el cual verifica* $T' : \mathcal{P} \vdash_{\mathcal{D}} f\, \overline{e'}_n \overline{a'}_k \rightarrow t' \Leftarrow \Pi'$ *y* $\mid T \mid = 1 + \sum_{i=1}^{m} \mid T_i \mid + \mid \overline{T} \mid + \mid T_r \mid + \mid T_s \mid \geq 1 + \sum_{i=1}^{m} \mid T'_i \mid + \mid \overline{T'} \mid + \mid T'_r \mid + \mid T'_s \mid = \mid T' \mid$.

□

A.2.5. Resultados de adecuación para semánticas fuertes y débiles

Demostramos los principales resultados presentados en las Subsecciones 3.4.1 y 3.4.2 sobre la adecuación de la semántica fuerte y de la semántica débil.

Demostración 33 (Demostración del Teorema 4) *El resultado enunciado en el teorema se sigue a partir de las siguientes tres implicaciones:*

(1) $\mathcal{P} \vdash_{\mathcal{D}} \varphi \Rightarrow \mathcal{P} \models^s_{\mathcal{D}} \varphi$*:*

Asumamos $T : \mathcal{P} \vdash_{\mathcal{D}} \varphi$*. Consideremos cualquier modelo fuerte* $\mathcal{I} \models^s \mathcal{P}$*. Probamos que* $\mathcal{I} \Vdash_{\mathcal{D}} \varphi$ *razonando por inducción sobre* $\|T\|$*. Los casos base corresponden a* $T = \mathbf{RL}(\varphi, [\,])$*, donde* $\mathbf{RL} \in \{\mathbf{TI}, \mathbf{RR}, \mathbf{SP}\}$*. Todos ellos son triviales puesto que el mismo árbol* T *verifica* $T : \mathcal{I} \Vdash_{\mathcal{D}} \varphi$*.*

Los casos inductivos correspondientes a $T = \mathbf{RL}(\varphi, [T_1, \ldots, T_n])$*, donde* $\mathbf{RL} \in \{\mathbf{DC}, \mathbf{IR}, \mathbf{PF}, \mathbf{AC}\}$*, son también inmediatos, simplemente observando que la misma regla* $\mathbf{RL}$ *en CRWL*$(\mathcal{D})$ *es aplicable, y usando la* hipótesis de inducción *para* $T_1, \ldots, T_n$*.*

El caso interesante es aquel en el que la regla $\mathbf{DF}_{\mathcal{P}}$ *es aplicada en el paso raíz. Consideramos la primera variante de la regla* $\mathbf{DF}_{\mathcal{P}}$ *(el razonamiento para la segunda es muy similar).*

El árbol T *tendría la forma*

$$T = \mathbf{DF}_{\mathcal{P}}(f\, \bar{e}_n \rightarrow t \Leftarrow \Pi, [T_1, \ldots, T_n, \overline{T}, T_r])$$

con $T_i : \mathcal{P} \vdash_{\mathcal{D}} e_i \rightarrow t_i \Leftarrow \Pi$, $\overline{T} : \mathcal{P} \vdash_{\mathcal{D}} P \square \Delta \Leftarrow \Pi$, $T_r : \mathcal{P} \vdash_{\mathcal{D}} r \rightarrow t \Leftarrow \Pi$*, donde* $f \in DF^n$*,* $(f\, \bar{t}_n \rightarrow r \Leftarrow P \square \Delta) \in [\mathcal{P}]_\perp$*.*

Por la hipótesis de inducción *aplicada a* $\overline{T}, T_r$*, ha de existir* $\overline{T'}, T'_r$ *tal que* $\overline{T'} : \mathcal{I} \Vdash_{\mathcal{D}} P \,\square\, \Delta \Leftarrow \Pi$ *y* $T'_r : \mathcal{I} \Vdash_{\mathcal{D}} r \to t \Leftarrow \Pi$*. Esto, junto con el hecho de que* $\mathcal{I}$ *es un modelo fuerte de* $\mathcal{P}$*, asegura que* $(f\,\overline{t}_n \to t \Leftarrow \Pi) \in \mathcal{I}$*. Combinando esto con el resto de las* hipótesis de inducción*, las cuales aseguran la existencia de árboles* $T_i : \mathcal{I} \Vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$ *para* $i = 1 \ldots n$*, podemos construir el árbol* $T' = \mathbf{DF}_I(f\,\overline{e}_n \to t \Leftarrow \Pi, [T'_1, \ldots, T'_n, \overline{T'}, T'_r])$*, el cual prueba que* $\mathcal{I} \Vdash_{\mathcal{D}} f\,\overline{e}_n \to t$*.*

(2) $\mathcal{P} \models^s_{\mathcal{D}} \varphi \Rightarrow \mathcal{S}_{\mathcal{P}} \Vdash_{\mathcal{D}} \varphi$*:*

Esta implicación se satisface debido simplemente a que $\mathcal{S}_{\mathcal{P}} \models^s \mathcal{P}$*, como se demostró en el Teorema 2.*

(3) $\mathcal{S}_{\mathcal{P}} \Vdash_{\mathcal{D}} \varphi \Rightarrow \mathcal{P} \vdash_{\mathcal{D}} \varphi$*:*

Asumamos $\mathcal{S}_{\mathcal{P}} \Vdash_{\mathcal{D}} \varphi$*, donde* $\mathcal{S}_{\mathcal{P}} = \bigcup_{k \in \mathbb{N}} ST_{\mathcal{P}} \uparrow^k (\perp\!\!\!\perp)$*. Debido al hecho de que* $\bigcup_{k \in \mathbb{N}} ST_{\mathcal{P}} \uparrow^k (\perp\!\!\!\perp) \Vdash_{\mathcal{D}} \varphi$ *ha de ser probado mediante un árbol de prueba finito, y teniendo en cuenta que* $ST_{\mathcal{P}} \uparrow^k (\perp\!\!\!\perp)$ *crece con* k*, resulta fácil ver que ha de existir* $k \in \mathbb{N}$ *tal que* $ST_{\mathcal{P}} \uparrow^k (\perp\!\!\!\perp) \Vdash_{\mathcal{D}} \varphi$*. Por tanto, es suficiente con probar lo siguiente:*

$$\textit{Para todo } k \in \mathbb{N},\ ST_{\mathcal{P}} \uparrow^k (\perp\!\!\!\perp) \Vdash_{\mathcal{D}} \varphi \Rightarrow \mathcal{P} \vdash_{\mathcal{D}} \varphi$$

Probamos esto por inducción sobre k*.*

$\underline{k = 0}$*:*

Asumamos $ST_{\mathcal{P}} \uparrow^0 (\perp\!\!\!\perp) \Vdash_{\mathcal{D}} \varphi$*. Probamos que* $\mathcal{P} \vdash_{\mathcal{D}} \varphi$ *por inducción sobre la estructura de un árbol* T *con* $T : ST_{\mathcal{P}} \uparrow^0 (\perp\!\!\!\perp) \Vdash_{\mathcal{D}} \varphi$*. Distinguimos casos de acuerdo con la regla aplicada a la raíz de* T*:*

TI, RR *o* **SP***: trivial, puesto que la misma regla en CRWL*$(\mathcal{D})$ *demuestra que* $\mathcal{P} \vdash_{\mathcal{D}} \varphi$*.*

DC, IR, FV, PF *o* **AC***: inmediato usando la* hipótesis de inducción*, puesto que la misma regla también se aplica en CRWL*$(\mathcal{D})$*.*

$\mathbf{DF}_{\mathcal{I}}$*: Esta regla, de hecho, no es aplicable. En caso contrario, el árbol de prueba* T *tendría la forma* $T = \mathbf{DF}_{\mathcal{I}}(f\,\overline{e}_n \to t \Leftarrow \Pi, [T_1, \ldots, T_n])$*, con* $T_i : \mathcal{P} \vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$*, y donde* $f \in DF^n$*,* $(f\,\overline{t}_n \to t \Leftarrow \Pi) \in ST_{\mathcal{P}} \uparrow^0 (\perp\!\!\!\perp)$*. Pero* $ST_{\mathcal{P}} \uparrow^0 (\perp\!\!\!\perp) = \perp\!\!\!\perp$*, y por tanto* $(f\,\overline{t}_n \to t \Leftarrow \Pi)$ *es un c-hecho trivial, el cual implica que* $(f\,\overline{e}_n \to t \Leftarrow \Pi)$ *es también trivial, pero entonces la regla* **TI** *podría haber sido aplicada.*

Un razonamiento similar se cumple también para el segundo caso de la regla $\mathbf{DF}_{\mathcal{I}}$*.*

$\underline{k \mapsto k+1}$*:*

Asumamos $ST_{\mathcal{P}} \uparrow^{(k+1)} (\perp\!\!\!\perp) \Vdash_{\mathcal{D}} \varphi$. *La* hipótesis de inducción *indica que* $ST_{\mathcal{P}} \uparrow^{k} (\perp\!\!\!\perp) \Vdash_{\mathcal{D}} \psi \Rightarrow \mathcal{P} \vdash_{\mathcal{D}} \psi, \forall \psi$. *Como antes, probamos* $\mathcal{P} \vdash_{\mathcal{D}} \varphi$ *por inducción sobre la estructura del árbol de prueba para* $ST_{\mathcal{P}} \uparrow^{(k+1)} (\perp\!\!\!\perp) \Vdash_{\mathcal{D}} \varphi$. *También como antes, el caso interesante se produce cuando el paso raíz está constituido por una aplicación de* $\mathbf{DF}_{\mathcal{I}}$, *esto es, cuando* $T = \mathbf{DF}_{\mathcal{I}}(f\,\bar{e}_n \rightarrow t \Leftarrow \Pi, [T_1, \ldots, T_n])$, *con* $T_i : \mathcal{P} \vdash_{\mathcal{D}} e_i \rightarrow t_i \Leftarrow \Pi$ *y* $f \in DF^n$, $(f\,\bar{t}_n \rightarrow t \Leftarrow \Pi) \in ST_{\mathcal{P}} \uparrow^{k+1} (\perp\!\!\!\perp)$.

Ahora, puesto que $(f\,\bar{t}_n \rightarrow t \Leftarrow \Pi) \in ST_{\mathcal{P}} \uparrow^{(k+1)} (\perp\!\!\!\perp)$, *se sigue que, a partir de la definición de* $ST_{\mathcal{P}} \uparrow^{(k+1)} (\perp\!\!\!\perp)$, *ha de existir una instancia de regla* $(f\,\bar{t}_n \rightarrow r \Leftarrow P \,\square\, \Delta) \in [\mathcal{P}]_{\perp}$ *tal que* $r \rightarrow t \Leftarrow \Pi \in ST_{\mathcal{P}} \uparrow^{k} (\perp\!\!\!\perp)$ *y* $P \,\square\, \Delta \Leftarrow \Pi \in ST_{\mathcal{P}} \uparrow^{k} (\perp\!\!\!\perp)$. *Entonces, por la* hipótesis de inducción *(sobre* k*), tenemos que* $\mathcal{P} \vdash_{\mathcal{D}} r \rightarrow t \Leftarrow \Pi$ *y que* $\mathcal{P} \vdash_{\mathcal{D}} P \,\square\, \Delta \Leftarrow \Pi$. *Esto, junto con el árbol de prueba (*hipótesis de inducción*)* $\mathcal{P} \vdash_{\mathcal{D}} e_i \rightarrow t_i \Leftarrow \Pi$ *para* $i = 1, \ldots, n$, *nos permite construir usando* $\mathbf{DF}_{\mathcal{P}}$, *una derivación en CRWL*$(\mathcal{D})$ *para* $\mathcal{P} \vdash_{\mathcal{D}} f\bar{e}_n \rightarrow t \Leftarrow \Pi$.

Un razonamiento similar se satisface para el segundo caso de la regla $\mathbf{DF}_{\mathcal{I}}$.

□

Demostración 34 (Demostración del Teorema 5)

"$(a) \Rightarrow (b)$*" se verifica para cualquier c-sentencia* φ, *cerrada o no, puesto que:*

$\mathcal{P} \vdash_{\mathcal{D}} \varphi$
$\Rightarrow$ *(*Propiedad de Implicación *del Lema 7)*
$\mathcal{P} \vdash_{\mathcal{D}} \varphi\eta$ *para todo* $\eta \in GSub_{\mathcal{D}}$ *tal que* $\varphi\eta$ *es cerrado*
$\Rightarrow$ *(*Propiedad de Canonicidad *del Teorema 4)*
$\mathcal{S}_{\mathcal{P}} \Vdash_{\mathcal{D}} \varphi\eta$ *para todo* $\eta \in GSub_{\mathcal{D}}$ *tal que* $\varphi\eta$ *es cerrado*
$\Rightarrow$ *(*Propiedad de Cierre *del Lema 5)*
$[\mathcal{S}_{\mathcal{P}}]_{\mathcal{D}} \Vdash_{\mathcal{D}} \varphi\eta$ *para todo* $\eta \in GSub_{\mathcal{D}}$ *tal que* $\varphi\eta$ *es cerrado*
$\Rightarrow$ *(Teorema 3)*
$\mathcal{W}_{\mathcal{P}} \Vdash_{\mathcal{D}} \varphi\eta$ *para todo* $\eta \in GSub_{\mathcal{D}}$ *tal que* $\varphi\eta$ *es cerrado*
$\Rightarrow$ *(Teorema 2 (2),* Propiedad de Extensión *del Lema 5)*
$\mathcal{I} \Vdash_{\mathcal{D}} \varphi\eta$ *para todo* $\mathcal{I} \models^{w}_{\mathcal{D}} \mathcal{P}$ *y todo* $\eta \in GSub_{\mathcal{D}}$ *tal que* $\varphi\eta$ *es cerrado*
$\Rightarrow$ *(Definición 14 (2))*
$\mathcal{P} \models^{w}_{\mathcal{D}} \varphi$

"$(b) \Rightarrow (c)$" se cumple para cualquier c-sentencia cerrada φ, debido a que $\mathcal{W}_\mathcal{P}$ es un modelo débil de $\mathcal{P}$, como se muestra en el Teorema 2.

"$(c) \Rightarrow (a)$" también se cumple para cualquier c-sentencia cerrada φ:

$\mathcal{W}_\mathcal{P} \Vdash_\mathcal{D} \varphi$
$\Rightarrow$ *(Teorema 3)*
$[\mathcal{S}_\mathcal{P}]_\mathcal{D} \Vdash_\mathcal{D} \varphi$
$\Rightarrow$ *(*Propiedad de Extensión *del Lema 5)*
$\mathcal{S}_\mathcal{P} \Vdash_\mathcal{D} \varphi$
$\Rightarrow$ *(*Propiedad de Canonicidad *del Teorema 4)*
$\mathcal{P} \vdash_\mathcal{D} \varphi$

La Corrección *se satisface debido a que hemos probado la implicación "$(a) \Rightarrow (b)$" para un φ arbitrario. La* Completitud Básica *es una consecuencia inmediata de las dos implicaciones "$(b) \Rightarrow (c)$" y "$(c) \Rightarrow (a)$". La restricción a c-sentencias cerradas es necesaria, como muestra el siguiente CFLP($\mathcal{R}$)-programa:*

$$\mathcal{P} =_{def} \{notZero\, X \to true \Leftarrow X > 0,\\ notZero\, X \to true \Leftarrow X < 0\}$$

y el c-hecho $\varphi =_{def} notZero\, X \to true \Leftarrow X * X \not= 0$. Para esta elección particular de $\mathcal{P}$ y φ tenemos que:

- *Para cada modelo débil $\mathcal{I} \models^w_\mathcal{R} \mathcal{P}$, es fácil ver que $(notZero\ x \to true) \in \mathcal{I}$ para todo $x \in \mathbb{R} \setminus \{0\}$, lo que implica que $\mathcal{I} \models^w_\mathcal{R} \varphi$. Por tanto, $\mathcal{P} \models^w_\mathcal{R} \varphi$.*

- *Por otra parte, $\mathcal{P} \not\vdash_\mathcal{R} \varphi$ debido a que la prueba (si existe) debería usar la CRWL($\mathcal{R}$)-regla $\mathbf{DF}_\mathcal{P}$ junto con alguna regla de programa, y ninguna de las dos reglas en $\mathcal{P}$ soporta tal inferencia.*

Finalmente, la Canonicidad Básica *se sigue a partir de la equivalencia entre (c) y (a) y la* Propiedad de Conservación *del Lema 5, razonando como en la demostración del Teorema 4. Observamos que $\mathcal{W}_\mathcal{P}$ incluye también algunos c-hechos no cerrados, debido a que se requiere que todas las c-interpretaciones sobre $\mathcal{D}$ sean cerradas bajo $cl_\mathcal{D}$. No obstante, para los propósitos de la semántica débil, los c-hechos cerrados proporcionan toda la información relevante.*

□

A.3. Resultados presentados en el Capítulo 4

En esta sección del apéndice incluimos las demostraciones que han sido omitidas en el Capítulo 4 de esta tesis referentes al *Lema de Corrección* y al *Lema de Progreso* del cálculo $CLNC(\mathcal{D})$ y del cálculo $CDNC(\mathcal{D})$ de estrechamiento perezoso con restricciones. Ambos resultados resultan esenciales para probar la corrección y la completitud de la semántica operacional del esquema $CFLP(\mathcal{D})$. También damos la demostración de la *Propiedad de Particionado* y de las *propiedades del algoritmo de transformación COIS* para la obtención de árboles definicionales mediante la aplicación de técnicas de transformación de $CFLP(\mathcal{D})$-programas.

A.3.1. Lema de corrección del cálculo $CLNC(\mathcal{D})$

Comenzamos demostrando las principales propiedades del cálculo de resolución de objetivos $CLNC(\mathcal{D})$ con respecto a su corrección.

Demostración 35 (Demostración del Lema 10) *Dentro de esta demostración asumimos que, para cada $CLNC(\mathcal{D})$-regla, G, G' y G_i son exactamente como se han mostrado en la presentación del cálculo $CLNC(\mathcal{D})$.*

(1) Para cada $CLNC(\mathcal{D})$-regla damos a continuación breves explicaciones justificando la preservación de las condiciones de admisibilidad de los objetivos.

DC
LN: *Tanto $h\bar{t}_m$ como la tupla $\bar{t}_m$ comparten las mismas propiedades de linealidad.*
EX, SL: *Trivial.*
NC: *Debido a la descomposición, la nueva relación de producción es un subconjunto de la anterior.*
En este caso, trivialmente $fvar(G') \subseteq fvar(G)$.
SP_1
LN: *Puesto que X no es producida ($X \notin \overline{U}$ y entonces $X \notin pvar(P)$ por la condición de admisibilidad* **EX***), σ_0 no modifica los lados derechos de producciones en P.*
EX: *Por la misma razón, las variables en t no son ya producidas (debido a* **LN***), mientras que $\overline{U}$ no se modifica.*
NC: *Puesto que X no es producida, sólo producciones $e \to s \in P$ con $X \in var(e)$ se ven afectadas por σ_0. En estos casos, para cada $Y \in var(t)$ y $Z \in var(s)$, se crea la relación $Y \gg_P Z$. Pero $Y \gg_P Z$ no puede formar parte de un ciclo de variables, debido a que la linealidad asegura que Y no aparece en el lado derecho de una producción en el nuevo objetivo.*
SL: *X no es producida, las variables en t no son ya producidas, y σ_0 sólo puede introducir las variables en t como nuevas variables en la sustitución respuesta.*
Puesto que $X \notin \overline{U}$, $var(t) \subseteq \overline{U}$ (debido a **EX***) y $\overline{U}$ no se ve modificado, concluimos que $fvar(G') \subseteq fvar(G)$.*

SP_2
LN: *Si* $X \in pvar(P)$*, sea* $e \to s$ *la única producción en* P *tal que* $X \in var(s)$ *(*G *verifica* **LN***). Observamos que* σ_0 *reemplaza* X *por* t *en* s *(y ningún otro lado derecho es modificado), pero* $X \to t$ *es eliminado, de modo que* **LN** *se preserva. Si* $X \notin pvar(P)$*,* σ_0 *no modifica los lados derechos de las producciones en* P*.*
EX: *Si* $X \in pvar(P)$*,* $pvar(P') = pvar(P)\backslash\{X\} \subseteq evar(G') = evar(G)\backslash\{X\}$*. Si* $X \notin pvar(P)$*, las variables en* t *dejan de ser producidas (debido a* **LN***), mientras que* $\overline{U}$ *no se ve modificado.*
NC: *Si* $X \in pvar(P)$*, la relación de producción se modifica de las siguientes formas: (i) Si* $e \to s$ *es la única producción en* P *tal que* $X \in var(s)$*, entonces en* G' *tenemos que* $Y \gg_P Z$ *para cada* $Y \in var(e)$ *y* $Z \in var(t)$*. Pero en* G *teníamos que* $Y \gg_P X$ *y* $X \gg_P Z$*, y por tanto* $\gg_P^*$ *no se ve alargada.*
(ii) Si $e' \to s'$ *es otra producción en* P *tal que* $X \in var(e')$*, entonces en* G' *tenemos que* $Z \gg_P V$ *para cada* $Z \in var(t)$ *y* $V \in var(s')$*. Cualquier ciclo en* G' *que tenga que ver con* $Z \gg_P V$ *ha de ser de la forma* $\ldots, Y \gg_P Z, Z \gg_P V, \ldots$*, donde* $Y \gg_P Z$ *viene de (i). Pero en* G *podríamos tener* $\ldots, Y \gg_P X, X \gg_P V, \ldots$*, contradiciendo* **NC** *de* G*.*
Si $X \notin pvar(P)$*, solamente las producciones* $e \to s \in P$ *con* $X \in var(e)$ *se ven afectadas por* σ_0*. En estos casos, para cada* $Y \in var(t)$ *y* $Z \in var(s)$*, se crea la relación* $Y \gg_P Z$*. Sin embargo,* $Y \gg_P Z$ *no puede tomar parte de un ciclo de variables, debido a que la linealidad asegura que* Y *no aparece en ningún lado derecho de una producción en el nuevo objetivo.*
SL: σ_0 *no modifica* σ *y* $pvar(P') \subseteq pvar(P)$*.*
Puesto que $X \in evar(G)$*,* $var(t) \subseteq \overline{U}$ *y* $\overline{U}$ *no se modifica, concluimos que* $fvar(G') \subseteq fvar(G)$*.*

SP_3
LN: *Debido a que* $X \in pvar(P)$ *y a la linealidad de* G*,* σ_0 *no modifica los lados derechos de* P*.*
EX: $pvar(P') = pvar(P)\backslash\{X\} \subseteq evar(G') = evar(G)\backslash\{X\}$*.*
NC: *Puesto que* X *es producida, solamente las producciones* $e \to s$ *con* $X \in var(e)$ *se ven afectadas por* σ_0*. En estos casos, para cada* $Y \in var(t)$ *y* $Z \in var(s)$*, se crea la relación* $Y \gg_P Z$*, donde previamente teníamos* $Y \gg_P X$*,* $X \gg_P Z$*. Por tanto,* $\gg_P^*$ *no se ve alargada.*
SL: σ_0 *no modifica* σ *y* $pvar(P') \subseteq pvar(P)$*.*
Puesto que $X \in evar(G)$*,* $X \notin var(t)$ *y* $\overline{U}$ *no se modifica, concluimos que* $fvar(G') \subseteq fvar(G)$*.*

IM *Si* G *es admisible, entonces* $G'' \equiv \exists X, \overline{X}_m, \overline{U}.\ h\overline{e}_m \to X, X \to h\overline{X}_m, P \,\square\, C \,\square\, S \,\square\, \sigma$*. Aplicar* **IM** *a* G *es equivalente a aplicar* **SP_2** *y* **DC** *a* G''*.*

EL
LN, NC, SL: *Trivial.*
EX: $pvar(P') = pvar(P)\backslash\{X\} \subseteq evar(G') = evar(G)\backslash\{X\}$.
Como $X \in evar(G)$, $X \notin var(P \,\square\, C \,\square\, S \,\square\, \sigma)$ *y* $\overline{U}$ *no se modifica, concluimos que* $fvar(G') \subseteq fvar(G)$.

PF
LN: *Trivial, ya que* $\overline{X}_q$ *son variables nuevas y distintas.*
EX: *Todas las variables nuevas* $\overline{X}_q$ *están cuantificadas existencialmente.*
NC: *Las variables* $\overline{X}_q$ *son nuevas, y por tanto no aparecen en ningún lado izquierdo de producciones en* G'. *Por tanto, no se crea ningún ciclo de variables producidas.*
SL: σ *no se ve alterada y las variables* $\overline{X}_q$ *son nuevas.*
Puesto que todas las variables nuevas introducidas en G' *están cuantificadas existencialmente y* $\overline{U}$ *no se ve modificado, concluimos que trivialmente* $fvar(G') \subseteq fvar(G)$.

DF_2
LN: X *es una variable nueva, y puesto que* $f\overline{t}_n \to r \Leftarrow P' \,\square\, C'$ *es una variante fresca de una regla de programa,* $\overline{t}_n$ *es una tupla lineal de patrones con nuevas variables y cada variable producida en* P' *es también nueva y es producida una sola vez, por las condiciones de admisibilidad de las reglas de programa.*
EX: *Todas las variables nuevas* $X, \overline{Y}$ *están cuantificadas existencialmente.*
NC: *Las variables en* P' *son nuevas y la clausura transitiva de* $\gg_{P'}$ *es un orden parcial estricto debido a las condiciones de admisibilidad de las reglas de programa. Las variables en cada* t_i *son nuevas, por lo que no aparecen en ningún lado izquierdo de producciones de* $X\overline{a}_k \to t$ *y* P *(*X *es una nueva variable), con la excepción de* $r \to X$. *Si aparece un ciclo para la variable* $V \in var(t_i)$, *este debe ser de la forma* $\ldots, Y \gg_P V, V \gg_P X, X \gg_P Z, \ldots$, *donde* $Y \in var(e_i)$ *y* $Z \in var(t)$. *Pero en* G *podríamos tener* $\ldots, Y \gg_P Z, \ldots$ *contradiciendo* **NC** *de* G.
SL: σ *no se modifica y tanto la variable* X *como las variables en* t_i *y* P' *son nuevas. Puesto que todas las variables nuevas introducidas en* G' *son nuevas y cuantificadas existencialmente y* $\overline{U}$ *no se ve modificado, concluimos directamente que* $fvar(G') \subseteq fvar(G)$.

DF_1 *Análogo al caso* **DF_2** *e incluso más sencillo.*

FV_1
LN: *Puesto que* F *no es producida (*$F \notin var(t)$ *debido a* **NC** *y a que* $F \notin pvar(P)$*),* σ_0 *no modifica los lados derechos de las producciones en* G'.
EX: *Por la misma razón,* $pvar(P') = var(t) \cup pvar(P) \subseteq evar(G) \subseteq evar(G') = evar(G) \cup \{\overline{X}_p\}$.

NC: *Puesto que F no es producida, solamente las producciones $e \to s \in P$ con $F \in var(e)$ se ven afectadas por σ_0. En estos casos, para cada $X \in \{\overline{X}_p\}$ y $Z \in var(s)$, se crea la relación $X \gg_P Z$. Pero $X \gg_P Z$ no puede tomar parte de un ciclo de variables, debido a que cada variable $X \in \{\overline{X}_p\}$ es nueva y no aparece en ningún lado derecho de una producción en el nuevo objetivo.*
SL: *Como F no es producida y $\overline{X}_p$ son variables nuevas, σ_0 solamente puede introducir variables nuevas y no producidas en la sustitución respuesta.*
Como las nuevas variables $\overline{X}_p$ introducidas en G' están cuantificadas existencialmente, F no es producida y $\overline{U}$ no se ve modificada, concluimos que $fvar(G') \subseteq fvar(G)$.

FV_2
LN: *Como $F \in pvar(P)$, sea $e \to s$ la única producción en P tal que $F \in var(s)$ (G verifica* **LN***). Observamos que σ_0 reemplaza F por $h\overline{X}_p$ en s (y ningún otro lado derecho se ve modificado), pero $\overline{X}_p$ son variables nuevas y distintas, de modo que* **LN** *se preserva.*
EX: $pvar(P') = ((var(t) \cup pvar(P)) \setminus \{F\}) \cup \{\overline{X}_p\} \subseteq evar(G') = (evar(G) \setminus \{F\}) \cup \{\overline{X}_p\}$.
NC: *Como $F \in pvar(P)$, la relación de producción se ve modificada en los dos siguientes modos:*
(i) Si $e \to s$ es la única producción en P tal que $F \in var(s)$, entonces $F \notin var(e)$, $F \notin var(t)$ (debido a **NC***) y en G' tenemos $Y \gg_P X$, $X \gg_P Z$ para cada $Y \in var(e)$, $X \in \{\overline{X}_p\}$ y $Z \in var(t)$ (y de aquí $Y \gg_P Z$). Pero en G teníamos $Y \gg_P F$, $F \gg_P Z$ (y de aquí también $Y \gg_P Z$), y por tanto $\gg_P^*$ no se ve alargada.*
(ii) Si $e' \to s'$ es otra producción en P tal que $F \in var(e')$, entonces en G' tenemos $X \gg_P V$ para cada $X \in \{\overline{X}_p\}$ y $V \in var(s')$. Cualquier ciclo en G' que contenga a $X \gg_P V$ ha de ser de la forma $\ldots, Y \gg_P X, X \gg_P V, \ldots$, donde $Y \gg_P X$ procede de (i). Pero en G podríamos tener $\ldots, Y \gg_P F, F \gg_P V, \ldots$, contradiciendo **NC**.
SL: *σ_0 no modifica σ y $pvar(P') = ((var(t) \cup pvar(P)) \setminus \{F\}) \cup \{\overline{X}_p\}$, donde $F \in evar(G)$ y $\overline{X}_p$ son variables nuevas.*
Por las mismas razones, y puesto que todas las nuevas variables $\overline{X}_p$ están cuantificadas existencialmente, concluimos que $fvar(G') \subseteq fvar(G)$.

CS
LN: *Como $\chi = pvar(P)$ y el resolutor de restricciones satisface el requerimiento de que $\chi \cap (dom(\sigma_i) \cup ran(\sigma_i)) = \emptyset$, tenemos que $pvar(P) \cap dom(\sigma_i) = \emptyset$ y entonces σ_i no modifica los lados derechos de las producciones en P.*
EX: *Por la misma razón, $pvar(P_i) = pvar(P) \subseteq evar(G) \subseteq evar(G_i) = evar(G) \cup \{\overline{Y}_i\}$.*
NC: *Puesto que también $pvar(P) \cap ran(\sigma_i) = \emptyset$ e $\overline{Y}_i$ son variables nuevas, ninguna variable producida es introducida por σ_i en los nuevos lados izquierdos de las producciones en P. Por tanto, ningún ciclo de variables producidas es creado en G_i.*

SL: *Como* $pvar(P_i) = pvar(P)$ *y* $pvar(P) \cap var(\sigma_i) = \emptyset$*, ninguna variable producida entra en la nueva sustitución respuesta.*
Puesto que las nuevas variables $\overline{Y}_i$ *introducidas en* G_i *están cuantificadas existencialmente y* $\overline{U}$ *no se ve modificado, concluimos que* $fvar(G') \subseteq fvar(G)$*.*

AC *Análogo al caso de* **PF***.*

(2) Procedemos considerando las reglas de fallo de $CLNC(\mathcal{D})$ *una a una.*

CF *Asumimos que* $\Pi \square \theta \in Ans_{\mathcal{P}}(G)$*. Por definición, existe una sustitución* $\hat{\theta}$ *tal que* $\hat{\theta} =_{\backslash evar(G)} \theta$ *y* $\mathcal{P} \vdash_{\mathcal{D}} h\overline{e_p\hat{\theta}} \rightarrow h'\overline{t_q\hat{\theta}} \Leftarrow \Pi$*. A partir del hecho de que* $h \neq h'$ *o bien* $p \neq q$*, no puede ser cierto que* $h\overline{e_p\hat{\theta}} \rightarrow h'\overline{t_q\hat{\theta}} \Leftarrow \Pi$ *sea demostrable en* $CRWL(\mathcal{D})$ *si* $Sat_{\mathcal{D}}(\Pi)$*. Por tanto,* $Ans_{\mathcal{P}}(G) = \emptyset$*, y análogamente,* $Sol_{\mathcal{P}}(G) = \emptyset$*.*

SF *Asumamos que* $\Pi \square \theta \in Ans_{\mathcal{P}}(G)$*. Por definición, existe una sustitución* $\hat{\theta}$ *tal que* $\hat{\theta} =_{\backslash evar(G)} \theta$ *y* $\Pi \models_{\mathcal{D}} S\hat{\theta}$*. Sea* $\mu \in Sol_{\mathcal{D}}(\Pi)$*. Entonces,* $\hat{\theta}\mu \in Sol_{\mathcal{D}}(S)$*. Debido al hecho de que* $solve^{\mathcal{D}}(S, \chi) = \blacksquare$*, tenemos que* $Sol_{\mathcal{D}}(S) = \emptyset$ *(por los requisitos exigidos a los resolutores de restricciones). De aquí también* $Sol_{\mathcal{D}}(S\hat{\theta}) = \emptyset$*, y entonces* $Sol_{\mathcal{D}}(\Pi) = \emptyset$ *(lo que implica* $Insat_{\mathcal{D}}(\Pi)$*). En consecuencia,* $Ans_{\mathcal{P}}(G) = \emptyset$*, y analogamente,* $Sol_{\mathcal{P}}(G) = \emptyset$*.*

(3) Procedemos de nuevo regla por regla.

DC *Asumamos que* $\Pi \square \theta \in Ans_{\mathcal{P}}(G')$*. Existe* $\hat{\theta} =_{\backslash evar(G)} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$*,* $X\hat{\theta} \equiv t\hat{\theta}$ *para cada* $\{X \mapsto t\} \in \sigma$*,* $\mathcal{P} \vdash_{\mathcal{D}} (P \square C)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} (e_i \rightarrow t_i)\hat{\theta} \Leftarrow \Pi$ *para cada* $1 \leq i \leq m$*. Entonces,* $\mathcal{T} \equiv \mathbf{DC}((h\overline{e}_m \rightarrow h\overline{t}_m)\hat{\theta} \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_m])$ *es un árbol de prueba para* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (h\overline{e}_m \rightarrow h\overline{t}_m)\hat{\theta} \Leftarrow \Pi$*. Por tanto,* $\Pi \square \theta \in Ans_{\mathcal{P}}(G)$*.*
SP$_1$ *Asumamos que* $\Pi \square \theta \in Ans_{\mathcal{P}}(G')$*. Ha de existir* $\hat{\theta} =_{\backslash evar(G)} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\sigma_0\hat{\theta}$*,* $X\hat{\theta} \equiv t\hat{\theta}$ *para* $\{X \mapsto t\} \in \sigma_0$*,* $Y\hat{\theta} \equiv s\hat{\theta}$ *para cada* $\{Y \mapsto s\} \in \sigma$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (P \square C)\sigma_0\hat{\theta} \Leftarrow \Pi$*. Puesto que* $\sigma_0\hat{\theta} = \hat{\theta}$*, se tiene que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$*,* $Y\hat{\theta} \equiv s\hat{\theta}$ *para cada* $\{Y \mapsto s\} \in \sigma$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (P \square C)\hat{\theta} \Leftarrow \Pi$*. Más aún,* $\Pi \models_{\mathcal{D}} X\hat{\theta} \sqsupseteq t\hat{\theta}$*, y usando la* Propiedad de Aproximación *tenemos también que* $\mathcal{P} \vdash_{\mathcal{D}} (X \rightarrow t)\hat{\theta} \Leftarrow \Pi$*. En consecuencia,* $\Pi \square \theta \in Ans_{\mathcal{P}}(G)$*.*

SP$_2$ *Asumamos que* $\Pi \square \theta \in Ans_{\mathcal{P}}(G')$*. Entonces, ha de existir* $\hat{\theta} =_{\backslash evar(G)} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\sigma_0\hat{\theta}$*,* $Y\hat{\theta} \equiv s\hat{\theta}$ *para cada* $\{Y \mapsto s\} \in \sigma[\sigma_0]$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (P \square C)\sigma_0\hat{\theta} \Leftarrow \Pi$*. Definimos* $\hat{\theta}'(X) = t\hat{\theta}$ *y* $\hat{\theta}'(Y) = \hat{\theta}(Y)$ *para* $Y \not\equiv X$*. Se cumple que* $\sigma_0\hat{\theta} = \hat{\theta}'$*. Entonces* $\Pi \models_{\mathcal{D}} S\hat{\theta}'$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (P \square C)\hat{\theta}' \Leftarrow \Pi$*. Como* $X \in \overline{U}$*, si* $X \in pvar(P)$ *entonces* $X \notin var(\sigma)$ *por la condición de admisibilidad* **SL** *y tenemos que* $Y\hat{\theta}' \equiv Y\hat{\theta} \equiv s\hat{\theta} \equiv s\hat{\theta}'$ *para cada* $\{Y \mapsto s\} \in \sigma$*. Si* $X \notin pvar(P)$*, entonces* $Y\hat{\theta}' \equiv Y\hat{\theta} \equiv s\sigma_0\hat{\theta} \equiv s\hat{\theta}'$

para cada $\{Y \mapsto s\} \in \sigma$ *(puesto que* $\{Y \mapsto s\sigma_0\} \in \sigma\sigma_0$*). Es más, como* $X \notin var(t)$, $X\hat{\theta'} \equiv t\hat{\theta} \equiv t\hat{\theta'}$. *Ahora* $\Pi \models_{\mathcal{D}} X\hat{\theta'} \sqsupseteq t\hat{\theta'}$, *y usando la* Propiedad de Aproximación *tenemos también que* $\mathcal{P} \vdash_{\mathcal{D}} (X \to t)\hat{\theta'} \Leftarrow \Pi$. *Finalmente, como* $\hat{\theta} =_{\setminus\{X\}} \hat{\theta'}$ *concluimos que* $\Pi \square \theta \in Ans_{\mathcal{P}}(G)$.

SP$_3$ *Similar al caso* **SP$_2$**.

IM *Asumamos que* $\Pi \square \theta \in Ans_{\mathcal{P}}(G')$. *Existe* $\hat{\theta} =_{\setminus evar(G)} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\sigma_0\hat{\theta}$, $Y\hat{\theta} \equiv s\hat{\theta}$ *para cada* $\{Y \mapsto s\} \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \square C)\sigma_0\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} (e_i \to X_i)\sigma_0\hat{\theta} \Leftarrow \Pi$ *para cada* $1 \leq i \leq m$. *Como* $X \notin var(e_i)$ *por la condición de admisibilidad* **NC** *y* $\overline{X}_m$ *son variables nuevas en* G', *también tenemos que* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} (e_i \to X_i)\hat{\theta} \Leftarrow \Pi$. *Definimos ahora* $\hat{\theta'}(X) = h\overline{X_m\hat{\theta}}$, $\hat{\theta'}(X_i) = X_i$ *para todo* $1 \leq i \leq m$ *y* $\hat{\theta'}(Y) = \hat{\theta}(Y)$ *para todo* $Y \notin \{X, X_1, \ldots, X_m\}$. *Se cumple que* $\sigma_0\hat{\theta} =_{\setminus\{X_1,\ldots,X_m\}} \hat{\theta'}$. *Puesto que* $\overline{X}_m$ *son variables nuevas en* G', $\Pi \models_{\mathcal{D}} S\hat{\theta'}$, $\mathcal{P} \vdash_{\mathcal{D}} (P \square C)\hat{\theta'} \Leftarrow \Pi$ *y* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i\hat{\theta'} \to X_i\hat{\theta} \Leftarrow \Pi$. *Más aún, como* X *es una variable producida en* G, $X \notin var(\sigma)$ *por la condición de admisibilidad* **SL**, *y entonces* $Y\hat{\theta'} \equiv Y\hat{\theta} \equiv s\hat{\theta} \equiv s\hat{\theta'}$ *para cada* $\{Y \mapsto s\} \in \sigma$. *Finalmente,* $\mathcal{T} \equiv \mathbf{DC}(h\overline{e_m\hat{\theta'}} \to h\overline{X_m\hat{\theta}} \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_m])$ *es un árbol de prueba para* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (h\overline{e}_m \to X)\hat{\theta'} \Leftarrow \Pi$. *Por tanto, como* $\hat{\theta} =_{\setminus\{X,X_1,\ldots,X_m\}} \hat{\theta'}$ *concluimos que* $\Pi \square \theta \in Ans_{\mathcal{P}}(G)$.

EL *Asumamos que* $\Pi \square \theta \in Ans_{\mathcal{P}}(G')$. *Ha de existir* $\hat{\theta} =_{\setminus evar(G)} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $Y\hat{\theta} \equiv s\hat{\theta}$ *para cada* $\{Y \mapsto s\} \in \sigma$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (P \square C)\hat{\theta} \Leftarrow \Pi$. *Definimos* $\hat{\theta'}(X) = \bot$ *y* $\hat{\theta'}(Y) = \hat{\theta}(Y)$ *para* $Y \not\equiv X$ *(observamos que* $\hat{\theta'}$ *está bien definida, porque* X *es una variable producida en* G*). Se cumple que* $\hat{\theta} =_{\setminus\{X\}} \hat{\theta'}$. *Como* $X \notin var(P \square C \square S \square \sigma)$, *directamente tenemos que* $\Pi \models_{\mathcal{D}} S\hat{\theta'}$, $Y\hat{\theta'} \equiv Y\hat{\theta} \equiv s\hat{\theta} \equiv s\hat{\theta'}$ *para cada* $\{Y \mapsto s\} \in \sigma$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (P \square C)\hat{\theta'} \Leftarrow \Pi$. *Más aún,* $\mathcal{T} \equiv \mathbf{TI}(e\hat{\theta'} \to \bot \Leftarrow \Pi, [\,])$ *es un árbol de prueba para* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (e \to X)\hat{\theta'} \Leftarrow \Pi$. *Por tanto, puesto que* $\hat{\theta} =_{\setminus\{X\}} \hat{\theta'}$, *concluimos que* $\Pi \square \theta \in Ans_{\mathcal{P}}(G)$.

PF *Asumamos que* $\Pi \square \theta \in Ans_{\mathcal{P}}(G')$. *Entonces existe* $\hat{\theta} =_{\setminus evar(G)} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $\Pi \models_{\mathcal{D}} (p\bar{t}_n \to! \; t)\hat{\theta}$, $X\hat{\theta} \equiv s\hat{\theta}$ *para cada* $\{X \mapsto s\} \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \square C)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (e_j \to X_j)\hat{\theta} \Leftarrow \Pi$ *para cada* $1 \leq j \leq q$ *(si* $q = 0$ *se omiten estas pruebas). Definimos* $\hat{\theta'}(X_j) = X_j$ *para cada* $1 \leq j \leq q$ *y* $\hat{\theta'}(Y) = \hat{\theta}(Y)$ *para* $Y \notin \{X_1, \ldots, X_q\}$. *Se cumple que* $\hat{\theta'} =_{\setminus\{\overline{X}_q\}} \hat{\theta}$. *Como* $\overline{X}_q$ *son variables nuevas,* $\Pi \models_{\mathcal{D}} S\hat{\theta'}$, $X\hat{\theta'} \equiv X\hat{\theta} \equiv s\hat{\theta} \equiv s\hat{\theta'}$ *para cada* $\{X \mapsto s\} \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \square C)\hat{\theta'} \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}} e_j\hat{\theta'} \to X_j\hat{\theta} \Leftarrow \Pi$ *para cada* $1 \leq j \leq q$. *Como* $e_j \notin Pat_{\mathcal{D}}$ *para todo* $1 \leq j \leq q$, *se tiene que* $\mathcal{P} \vdash_{\mathcal{D}} e_j\hat{\theta'} \to t_j\hat{\theta} \Leftarrow \Pi$ *para cada* $1 \leq j \leq q$. *Es más, para cada* $e_i \in Pat_{\mathcal{D}}$ *sabemos que* $e_i \equiv t_i$, *y trivialmente* $\Pi \models_{\mathcal{D}} e_i\hat{\theta'} \sqsupseteq t_i\hat{\theta}$. *Por la* Propiedad de Aproximación *podemos obtener árboles de prueba* $\mathcal{P} \vdash_{\mathcal{D}} e_i\hat{\theta'} \to t_i\hat{\theta} \Leftarrow \Pi$. *Por tanto, tenemos árboles de prueba* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i\hat{\theta'} \to t_i\hat{\theta} \Leftarrow \Pi$ *para cada* $1 \leq i \leq n$. *Como*

$\Pi \models_{\mathcal{D}} (p\overline{t}_n \to! \ t)\hat{\theta}$, *podemos construir un árbol de prueba* $\mathcal{T} \equiv \mathbf{PF}(p\overline{e}_n\hat{\theta}' \to t\hat{\theta} \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_n])$ *(*$t\hat{\theta} \neq \bot$ *puesto que* $t \notin \mathcal{V}ar$ *o* $t \in dvar_{\mathcal{D}}(P \,\square\, S)$*). Finalmente, como* $\overline{X}_q$ *son variables nuevas,* $t\hat{\theta} \equiv t\hat{\theta}'$ *y tenemos que* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (p\overline{e}_n \to t)\hat{\theta}' \Leftarrow \Pi$*. Por tanto, como* $\hat{\theta} =_{\setminus\{\overline{X}_q\}} \hat{\theta}'$*, concluimos que* $\Pi \,\square\, \theta \in Ans_{\mathcal{P}}(G)$*.*

DF₂ *Asumamos que* $\Pi \,\square\, \theta \in Ans_{\mathcal{P}}(G')$*. Ha de existir* $\hat{\theta} =_{\setminus evar(G)} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$*,* $Y\hat{\theta} \equiv s\hat{\theta}$ *para cada* $\{Y \mapsto s\} \in \sigma$*,* $\mathcal{T}'_C : \mathcal{P} \vdash_{\mathcal{D}} (P' \,\square\, C')\hat{\theta} \Leftarrow \Pi$*,* $\mathcal{T}_C : \mathcal{P} \vdash_{\mathcal{D}} (P \,\square\, C)\hat{\theta} \Leftarrow \Pi$*,* $\mathcal{T}_r : \mathcal{P} \vdash_{\mathcal{D}} (r \to X)\hat{\theta} \Leftarrow \Pi$*,* $\mathcal{T}_s : \mathcal{P} \vdash_{\mathcal{D}} (Xa_k \to t)\hat{\theta} \Leftarrow \Pi$ $(k > 0)$ *y* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} (e_i \to t_i)\hat{\theta} \Leftarrow \Pi$ *para cada* $1 \leq i \leq n$*. Definimos* $\hat{\theta}'(X) = X$*,* $\hat{\theta}'(Y) = Y$ *para todo* $Y \in \overline{Y}$ *y* $\hat{\theta}'(Z) = \hat{\theta}(Z)$ *para todo* $Z \notin \overline{Y} \cup \{X\}$*. Se cumple entonces que* $\hat{\theta} =_{\setminus \overline{Y} \cup \{X\}} \hat{\theta}'$*. Como* $\overline{Y} \cup \{X\}$ *son variables nuevas en* G'*, tenemos que* $\Pi \models_{\mathcal{D}} S\hat{\theta}'$*,* $\mathcal{T}_C : \mathcal{P} \vdash_{\mathcal{D}} (P \,\square\, C)\hat{\theta}' \Leftarrow \Pi$ *y* $Y\hat{\theta}' \equiv Y\hat{\theta} \equiv s\hat{\theta} \equiv s\hat{\theta}'$ *para cada* $\{Y \mapsto s\} \in \sigma$*. Tomamos ahora* $\mathcal{T} \equiv \mathbf{DF}_{\mathcal{P}}((f\overline{e}_n\overline{a}_k \to t)\hat{\theta} \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_n, \mathcal{T}'_C, \mathcal{T}_r, \mathcal{T}_s])$ $(k > 0)$ *usando* $(f\overline{t}_n \to r \Leftarrow P' \,\square\, C')\hat{\theta} \in [\mathcal{P}]_{\bot}$ *a partir de* $(f\overline{t}_n \to r \Leftarrow P' \,\square\, C') \in_{var} \mathcal{P}$ *(puesto que* $t \notin \mathcal{V}ar$ *o* $t \in dvar_{\mathcal{D}}(P \,\square\, S)$*,* $t\hat{\theta} \neq \bot$*. Como* $\mathcal{P} \vdash_{\mathcal{D}} (Xa_k \to t)\hat{\theta} \Leftarrow \Pi$ *con* $k > 0$ *y* $t\hat{\theta} \neq \bot$*,* $\hat{\theta}(X) \in Pat_{\mathcal{D}}$ *tal que* $\hat{\theta}(X) \neq \bot$*). Se sigue que* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (f\overline{e}_n\overline{a}_k \to t)\hat{\theta} \Leftarrow \Pi$*. Finalmente, debido a que* $\overline{Y} \cup \{X\}$ *son variables nuevas, también tenemos que* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (f\overline{e}_n\overline{a}_k \to t)\hat{\theta}' \Leftarrow \Pi$*. Por tanto, concluimos que* $\Pi \,\square\, \theta \in Ans_{\mathcal{P}}(G)$*.*

DF₁ *Análogo al caso* **DF₂** *e incluso más sencillo.*

FV₁ *Sea* $\mathcal{M} : \Pi \,\square\, \theta \in Ans_{\mathcal{P}}(G)$*. Entonces, ha de existir* $\hat{\theta} =_{\setminus evar(G)} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\sigma_0\hat{\theta}$*,* $X\hat{\theta} \equiv s\hat{\theta}$ *para cada* $\{X \mapsto s\} \in \sigma$*,* $\mathcal{P} \vdash_{\mathcal{D}} (P \,\square\, C)\sigma_0\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (h\overline{X}_m\overline{e}_n \to t)\sigma_0\hat{\theta} \Leftarrow \Pi$*. Definimos* $\hat{\theta}'(F) = h\overline{X_m\hat{\theta}}$*,* $\hat{\theta}'(X_i) = X_i$ *para cada* $1 \leq i \leq m$ *y* $\hat{\theta}'(Y) = \hat{\theta}(Y)$ *para todo* $Y \notin \{F, X_1, \ldots, X_m\}$*. Se cumple que* $\sigma_0\hat{\theta} =_{\setminus\{X_1,\ldots,X_m\}} \hat{\theta}'$*. Como además* $\overline{X}_m$ *son variables nuevas y* $P \notin pvar(P)$*,* $\Pi \models_{\mathcal{D}} S\hat{\theta}'$*,* $X\hat{\theta}' \equiv X\hat{\theta} \equiv s\hat{\theta} \equiv s\hat{\theta}'$ *para cada* $\{X \mapsto s\} \in \sigma$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (P \,\square\, C)\hat{\theta}' \Leftarrow \Pi$*. Más aún, puesto que* $F \notin \{X_1, \ldots, X_m\}$ *y* $F\hat{\theta} \equiv h\overline{X_m\hat{\theta}}$*, se tiene que* $\mathcal{P} \vdash_{\mathcal{D}} (F\overline{e}_n \to t)\hat{\theta}' \Leftarrow \Pi$*. Por tanto, como* $\hat{\theta} =_{\setminus\{X_1,\ldots,X_m\}} \hat{\theta}'$*, se concluye que* $\Pi \,\square\, \theta \in Ans_{\mathcal{P}}(G)$*.*

FV₂ *Sea* $\mathcal{M} : \Pi \,\square\, \theta \in Ans_{\mathcal{P}}(G)$*. Entonces, ha de existir* $\hat{\theta} =_{\setminus evar(G)} \theta$ *para el que* $\Pi \models_{\mathcal{D}} S\sigma_0\hat{\theta}$*,* $F\hat{\theta} \equiv h\overline{X_m\hat{\theta}}$ *para* $\{F \mapsto h\overline{X}_m\} \in \sigma_0$*,* $X\hat{\theta} \equiv s\hat{\theta}$ *para cada* $\{X \mapsto s\} \in \sigma$*,* $\mathcal{P} \vdash_{\mathcal{D}} (P \,\square\, C)\sigma_0\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (h\overline{X}_m\overline{e}_n \to t)\sigma_0\hat{\theta} \Leftarrow \Pi$*. Definimos* $\hat{\theta}'(X_i) = X_i$ *para cada* $1 \leq i \leq m$ *y* $\hat{\theta}'(Y) = \hat{\theta}(Y)$ *para todo* $Y \notin \{X_1, \ldots, X_m\}$*. Se cumple que* $\sigma_0\hat{\theta} =_{\setminus\{X_1,\ldots,X_m\}} \hat{\theta}'$*. Como* $\overline{X}_m$ *son variables nuevas,* $\Pi \models_{\mathcal{D}} S\hat{\theta}'$*,* $X\hat{\theta}' \equiv X\hat{\theta} \equiv s\hat{\theta} \equiv s\hat{\theta}'$ *para cada* $\{X \mapsto s\} \in \sigma$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (P \,\square\, C)\hat{\theta}' \Leftarrow \Pi$*. Más aún, debido a que* $F \notin \{X_1, \ldots, X_m\}$ *y a que* $\hat{\theta}'(F) = h\overline{X_m\hat{\theta}}$*, también tenemos que* $\mathcal{P} \vdash_{\mathcal{D}} (F\overline{e}_n \to t)\hat{\theta}' \Leftarrow \Pi$*. Por tanto, puesto que* $\hat{\theta} =_{\setminus\{F,X_1,\ldots,X_m\}} \hat{\theta}'$*, concluimos también que* $\Pi \,\square\, \theta \in Ans_{\mathcal{P}}(G)$*.*

CS *Asumamos que* $\Pi_i \square \theta_i \in Ans_{\mathcal{P}}(G_i)$ $(1 \leq i \leq k)$. *Existe* $\hat{\theta}_i =_{\backslash evar(G_i)} \theta_i$ *para la que* $\Pi_i \models_{\mathcal{D}} S_i\hat{\theta}_i$, $X\hat{\theta}_i \equiv t\hat{\theta}_i$ *para cada* $\{X \mapsto t\} \in \sigma_i$, $Y\hat{\theta}_i \equiv s\hat{\theta}_i$ *para cada* $\{Y \mapsto s\} \in \sigma$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (P \square C)\sigma_i\hat{\theta}_i \Leftarrow \Pi_i$. *Definimos* $\hat{\theta}'_i(Y) = Y$ *para cualquier* $Y \in \overline{Y}_i \setminus dom(\sigma_i)$ *y* $\hat{\theta}'_i(Z) = \hat{\theta}_i(Z)$ *para* $Z \in (\backslash\overline{Y}_i) \cup dom(\sigma_i)$. *Como* $X\hat{\theta}_i \equiv t\hat{\theta}_i$ *para cada* $\{X \mapsto t\} \in \sigma_i$, *se verifica que* $\sigma_i\hat{\theta}_i =_{(\backslash\overline{Y}_i) \cup dom(\sigma_i)} \hat{\theta}'_i$. *Debido a que* $\overline{Y}_i$ *son variables nuevas,* $Y\hat{\theta}'_i \equiv Y\hat{\theta}_i \equiv s\hat{\theta}_i \equiv s\hat{\theta}'_i$ *para cada* $\{Y \mapsto s\} \in \sigma$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (P \square C)\hat{\theta}'_i \Leftarrow \Pi_i$. *Probamos ahora que* $\Pi_i \models_{\mathcal{D}} S\hat{\theta}'_i$. *Sea* $\mu \in Sol_{\mathcal{D}}(\Pi_i)$. *Puesto que* $\Pi_i \models_{\mathcal{D}} S_i\hat{\theta}_i$, *se tiene que* $\mu \in Sol_{\mathcal{D}}(S_i\hat{\theta}_i)$. *De aquí,* $\hat{\theta}_i\mu \in Sol_{\mathcal{D}}(S_i)$. *Es más, ya que* $X\hat{\theta}_i \equiv t\hat{\theta}_i$ *para cada* $\{X \mapsto t\} \in \sigma_i$ *también tenemos que* $X\hat{\theta}_i\mu \equiv t\hat{\theta}_i\mu$ *para cada* $\{X \mapsto t\} \in \sigma_i$. *En consecuencia, ha de existir* $\hat{\theta}'_i\mu =_{\backslash S} \hat{\theta}_i\mu$ *para la que* $\hat{\theta}_i\mu \in Sol_{\mathcal{D}}(S_i)$ *y* $X\hat{\theta}_i\mu \equiv t\hat{\theta}_i\mu$ *para cada* $\{X \mapsto t\} \in \sigma_i$. *Por definición,* $\hat{\theta}'_i\mu \in Sol_{\mathcal{D}}(\exists_{\backslash S}.\ S_i \square \sigma_i)$ $(1 \leq i \leq k)$ *y entonces* $\hat{\theta}'_i\mu \in \bigcup_{i=1}^{k} Sol_{\mathcal{D}}(\exists_{\backslash S}.\ S_i \square \sigma_i)$. *Usando los requisitos de un resolutor de restricciones, se sigue también que* $\hat{\theta}'_i\mu \in Sol_{\mathcal{D}}(S)$. *Entonces,* $\mu \in Sol_{\mathcal{D}}(S\hat{\theta}'_i)$. *En consecuencia, se concluye que* $\Pi_i \square \theta_i \in Ans_{\mathcal{P}}(G)$ *para cada* $1 \leq i \leq n$.

AC *Análogo al caso* **PF**.

□

A.3.2. Lema de progreso del cálculo $CLNC(\mathcal{D})$

Demostramos ahora las principales propiedades del cálculo de resolución de objetivos $CLNC(\mathcal{D})$ con respecto a su completitud bajo el supuesto de que el resolutor utilizado sea idealmente completo.

Demostración 36 (Demostración del apartado (2) **del Lema 11)**

(2) En cada uno de los casos siguientes, G' *y* G_i *son los objetivos que se obtienen mediante la aplicación de la correspondiente transformación de* $CLNC(\mathcal{D})$.

DC *Sea* $\mathcal{M} : \Pi \square \theta \in Ans_{\mathcal{P}}(G)$. *Entonces existe* $\hat{\theta} =_{\backslash evar(G)} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $X\hat{\theta} \equiv t\hat{\theta}$ *para cada* $\{X \mapsto t\} \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \square C)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (h\overline{e}_m \rightarrow h\overline{t}_m)\hat{\theta} \Leftarrow \Pi$, *donde* $\mathcal{T} \equiv \mathbf{DC}((h\overline{e}_m \rightarrow h\overline{t}_m)\hat{\theta} \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_m])$ *con* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} (e_i \rightarrow t_i)\hat{\theta} \Leftarrow \Pi$ *para cada* $1 \leq i \leq m$. *De aquí,* $\mathcal{M}' : \Pi \square \theta \in Ans_{\mathcal{P}}(G')$, *donde* $\mathcal{M}'$ *es el resultado de reemplazar el árbol de prueba* $\mathcal{T}$ *en* $\mathcal{M}$ *por los árboles de prueba* $\mathcal{T}_i$ *para todo* $1 \leq i \leq n$.

SP$_1$ *Sea* $\mathcal{M} : \Pi \square \theta \in Ans_{\mathcal{P}}(G)$. *Entonces existe* $\hat{\theta} =_{\backslash evar(G)} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$,

$Z\hat{\theta} \equiv s\hat{\theta}$ *para cada* $\{Z \mapsto s\} \in \sigma$, $\overline{\mathcal{T}} : \mathcal{P} \vdash_{\mathcal{D}} (P \,\square\, C)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (X \to t)\hat{\theta} \Leftarrow \Pi$. *Debido a la* Propiedad de Aproximación, $\Pi \models_{\mathcal{D}} X\hat{\theta} \sqsupseteq t\hat{\theta}$, *y es posible elevar* $\hat{\theta}$ *hasta obtener* $\hat{\theta}' \sqsupseteq \hat{\theta}$ *de modo que* $\hat{\theta}' =_{\setminus var(t)} \hat{\theta}$ *y* $X\hat{\theta} \equiv X\hat{\theta}' \equiv t\hat{\theta}'$. *Como una consecuencia adicional, se tiene que* $\sigma_0\hat{\theta}' \sqsupseteq \hat{\theta}$. *Se sigue entonces que* $\Pi \models_{\mathcal{D}} S\sigma_0\hat{\theta}'$, $Z\hat{\theta}' \equiv s\hat{\theta}'$ *para cada* $\{Z \mapsto s\} \in \sigma\sigma_0$ *y que* $((P \,\square\, C)\hat{\theta} \Leftarrow \Pi) \succcurlyeq_{\mathcal{D}} ((P \,\square\, C)\sigma_0\hat{\theta}' \Leftarrow \Pi)$. *Usando la* Propiedad de Implicación, *también tenemos que* $\overline{\mathcal{T}'} : \mathcal{P} \vdash_{\mathcal{D}} (P \,\square\, C)\sigma_0\hat{\theta}' \Leftarrow \Pi$. *De aquí, si tomamos* $\theta' =_{\setminus evar(G')} \hat{\theta}'$ *entonces* $\mathcal{M}' : \Pi \,\square\, \theta' \in Ans_{\mathcal{P}}(G')$, *donde* $\mathcal{M}'$ *es el resultado de eliminar en* $\mathcal{M}$ *el árbol de prueba* $\mathcal{T}$ *y de reemplazar los árboles de prueba* $\overline{\mathcal{T}}$ *en* $\mathcal{M}$ *por los árboles de prueba* $\overline{\mathcal{T}'}$. *Claramente,* $Sol_{\mathcal{D}}(\Pi \,\square\, \theta) \subseteq Sol_{\mathcal{D}}(\exists_{\setminus G}.\ \Pi \,\square\, \theta')$.

SP₂ *Similar al caso de* **SP₁**.

SP₃ *Sea* $\mathcal{M} : \Pi \,\square\, \theta \in Ans_{\mathcal{P}}(G)$. *Ha de existir* $\hat{\theta} =_{\setminus evar(G)} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $Z\hat{\theta} \equiv s\hat{\theta}$ *para cada* $\{Z \mapsto s\} \in \sigma$, $\overline{\mathcal{T}} : \mathcal{P} \vdash_{\mathcal{D}} (P \,\square\, C)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (t \to X)\hat{\theta} \Leftarrow \Pi$. *Por la* Propiedad de Aproximación, $\Pi \models_{\mathcal{D}} t\hat{\theta} \sqsupseteq X\hat{\theta}$. *Si consideramos* $\hat{\theta}' = \sigma_0\hat{\theta}$, *se cumple que* $\hat{\theta}' =_{\setminus\{X\}} \hat{\theta}$, $\hat{\theta}' \sqsupseteq \hat{\theta}$ *y que* $\sigma_0\hat{\theta}' = \hat{\theta}'$. *Ahora es posible razonar de un modo similar al caso de* **SP₁**.

IM *Consideremos* $\mathcal{M} : \Pi \,\square\, \theta \in Ans_{\mathcal{P}}(G)$. *Ha de existir* $\hat{\theta} =_{\setminus evar(G)} \theta$ *para la que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $Z\hat{\theta} \equiv t\hat{\theta}$ *para cada* $\{Z \mapsto t\} \in \sigma$, $\overline{\mathcal{T}} : \mathcal{P} \vdash_{\mathcal{D}} (P \,\square\, C)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (h\bar{e}_m \to X)\hat{\theta} \Leftarrow \Pi$ *(puesto que* $X \in dvar_{\mathcal{D}}(P \,\square\, S)$, $X\hat{\theta} \neq \bot$*) donde* $\mathcal{T} \equiv \mathbf{R}((h\bar{e}_m \to X)\hat{\theta} \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_m])$ *con* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i\hat{\theta} \to t_i \Leftarrow \Pi$ *para cada* $1 \leq i \leq m$. *Si* $X\hat{\theta}$ *es de la forma* $h\bar{t}_m$ *entonces* **R** *es* **DC**, *y si* $X\hat{\theta}$ *fuera una variable, entonces* $\Pi \models_{\mathcal{D}} h\,\bar{t}_m \sqsupseteq X\hat{\theta}$ *y* **R** *es* **IR**. *Definimos* $\hat{\theta}'(X_i) = t_i$ *para todo* $1 \leq i \leq m$ *y* $\hat{\theta}'(Y) = \hat{\theta}(Y)$ *para todo* $Y \notin \{X_1, \ldots, X_m\}$. *Se cumple que* $\sigma_0\hat{\theta}' =_{\setminus\{X_1,\ldots,X_m\}} \hat{\theta}$ *si* **R** *es* **DC** *y que* $\Pi \models_{\mathcal{D}} \sigma_0\hat{\theta}' \sqsupseteq_{\setminus\{X_1,\ldots,X_m\}} \hat{\theta}$ *si* **R** *es* **IR**. *En ambos casos, puesto que* $\overline{X}_m$ *son variables nuevas y* X *es una variable producida, esto implica que* $\Pi \models_{\mathcal{D}} S\sigma_0\hat{\theta}'$, $Z\hat{\theta}' \equiv t\hat{\theta}'$ *para cada* $\{Z \mapsto t\} \in \sigma$ *y* $(e_i\hat{\theta} \to t_i \Leftarrow \Pi) \succcurlyeq_{\mathcal{D}} (e_i\sigma_0\hat{\theta}' \to t_i \Leftarrow \Pi)$ *para cada* $1 \leq i \leq m$. *Usando ahora la* Propiedad de Implicación, $\mathcal{T}'_i : \mathcal{P} \vdash_{\mathcal{D}} e_i\sigma_0\hat{\theta}' \to t_i \Leftarrow \Pi$ $(1 \leq i \leq m)$. *Puesto que* $\hat{\theta}'(\sigma_0(X_i)) = t_i$, *también tenemos que* $\mathcal{T}'_i : \mathcal{P} \vdash_{\mathcal{D}} (e_i \to X_i)\sigma_0\hat{\theta}' \Leftarrow \Pi$ *para cada* $1 \leq i \leq m$. *Es más, tenemos también que* $\overline{\mathcal{T}'} : \mathcal{P} \vdash_{\mathcal{D}} (P \,\square\, C)\sigma_0\hat{\theta}' \Leftarrow \Pi$ *mediante la aplicación de nuevo de la* Propiedad de Implicación. *De aquí, si tomamos* $\theta' =_{\setminus evar(G')} \hat{\theta}'$ *entonces* $\mathcal{M}' : \Pi \,\square\, \theta' \in Ans_{\mathcal{P}}(G')$, *donde* $\mathcal{M}'$ *es el resultado de reemplazar los árboles de prueba* $\mathcal{T}$,$\overline{\mathcal{T}}$ *en* $\mathcal{M}$ *por los árboles de prueba* $\mathcal{T}'_i$ $(1 \leq i \leq n)$ *y* $\overline{\mathcal{T}'}$, *respectivamente. Claramente,* $Sol_{\mathcal{D}}(\Pi \,\square\, \theta) \subseteq Sol_{\mathcal{D}}(\exists_{\setminus G}.\ \Pi \,\square\, \theta')$.

EL *Sea ahora* $\mathcal{M} : \Pi \,\square\, \theta \in Ans_{\mathcal{P}}(G)$. *Existe entonces* $\hat{\theta} =_{\setminus evar(G)} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $Z\hat{\theta} \equiv t\hat{\theta}$ *para cada* $\{Z \mapsto t\} \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \,\square\, C)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (e \to X)\hat{\theta} \Leftarrow \Pi$. *Por tanto, podemos tomar* $\mathcal{M}' : \Pi \,\square\, \theta \in Ans_{\mathcal{P}}(G')$, *donde* $\mathcal{M}'$ *es el resultado de eliminar en* $\mathcal{M}$ *el árbol de prueba* $\mathcal{T}$.

PF *Consideremos* $\mathcal{M} : \Pi \square \theta \in Ans_{\mathcal{P}}(G)$. *Sabemos que ha de existir* $\hat{\theta} =_{\backslash evar(G)} \theta$ *para la que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $X\hat{\theta} \equiv s\hat{\theta}$ *para cualquier* $\{X \mapsto s\} \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \square C)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (p\overline{e}_n \to t)\hat{\theta} \Leftarrow \Pi$ *(puesto que* $t \notin \mathcal{V}ar$ *o bien* $t \in dvar_{\mathcal{D}}(P \square S)$, $t\hat{\theta} \neq \bot$*) donde* $\mathcal{T} \equiv \mathbf{PF}((p\overline{e}_n \to t)\hat{\theta} \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_n])$ *siendo* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i\hat{\theta} \to t'_i \Leftarrow \Pi$ *para cada* $1 \leq i \leq n$ *y* $\Pi \models_{\mathcal{D}} p\,\overline{t'}_n \to\!! \, t\hat{\theta}$. *Para cualquier* $1 \leq i \leq n$, *si* $e_i \in Pat_{\mathcal{D}}$ *entonces* $\Pi \models_{\mathcal{D}} t'_i \sqsubseteq e_i\hat{\theta}$ *por la* Propiedad de Aproximación. *Definimos* $\hat{\theta}'(X_j) = t'_j$ *para todo* $1 \leq j \leq q$ *y* $\hat{\theta}'(Y) = \hat{\theta}(Y)$ *para todo* $Y \notin \{X_1, \ldots, X_q\}$ *(si* $q = 0$ *entonces tomamos* $\hat{\theta}' = \hat{\theta}$*). Disponemos de árboles de prueba* $\mathcal{T}_j : \mathcal{P} \vdash_{\mathcal{D}} e_j\hat{\theta} \to t'_j \Leftarrow \Pi$ *para cada* $1 \leq j \leq q$. *Como además* $\overline{X}_q$ *son variables nuevas, también tenemos que* $\Pi \models_{\mathcal{D}} S\hat{\theta}'$, $X\hat{\theta}' \equiv s\hat{\theta}'$ *para cualquier* $\{X \mapsto s\} \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \square C)\hat{\theta}' \Leftarrow \Pi$ *y* $\mathcal{T}_j : \mathcal{P} \vdash_{\mathcal{D}} (e_j \to X_j)\hat{\theta}' \Leftarrow \Pi$ *para cada* $1 \leq j \leq q$. *Por otra parte, para cualquier* $1 \leq i \leq n$, *si* $e_i \in Pat_{\mathcal{D}}$ *entonces sabemos que* $t_i \equiv e_i$ *y que* $\Pi \models_{\mathcal{D}} t'_i \sqsubseteq e_i\hat{\theta}$. *De aquí,* $\Pi \models_{\mathcal{D}} t'_i \sqsubseteq t_i\hat{\theta}'$. *En otro caso,* $t'_j = X_j\hat{\theta}'$, $t_j \equiv X_j$ *y se tiene que* $t'_j = t_j\hat{\theta}'$. *Por tanto, como* $t\hat{\theta} = t\hat{\theta}'$ *y puesto que* $\Pi \models_{\mathcal{D}} p\,\overline{t'}_n \to\!! \, t\hat{\theta}$, *también se tiene que* $\Pi \models_{\mathcal{D}} (p\,\overline{t}_n \to\!! \, t)\hat{\theta}'$. *Finalmente, si consideramos* $\theta' =_{\backslash evar(G')} \hat{\theta}'$ *entonces* $\mathcal{M}' : \Pi \square \theta' \in Ans_{\mathcal{P}}(G')$, *donde* $\mathcal{M}'$ *es el resultado de reemplazar el árbol de prueba* $\mathcal{T}$ *in* $\mathcal{M}$ *por los árboles de prueba* $\mathcal{T}_j$ $(1 \leq j \leq q)$ *(si* $q = 0$ *entonces es el resultado de eliminar el árbol de prueba* $\mathcal{T}$ *en* $\mathcal{M}$*). Claramente,* $Sol_{\mathcal{D}}(\Pi \square \theta) \subseteq Sol_{\mathcal{D}}(\exists_{\backslash G}.\ \Pi \square \theta')$.

$\mathbf{DF_2}$ *Asumamos* $\mathcal{M} : \Pi \square \theta \in Ans_{\mathcal{P}}(G)$. *Existe* $\hat{\theta} =_{\backslash evar(G)} \theta$ *para la que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $Z\hat{\theta} \equiv s\hat{\theta}$ *para cada* $\{Z \mapsto s\} \in \sigma$, $\overline{\mathcal{T}} : \mathcal{P} \vdash_{\mathcal{D}} (P \square C)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (f\overline{e}_n\overline{a}_k \to t)\hat{\theta} \Leftarrow \Pi$ *(como* $t \notin \mathcal{V}ar$ *o bien* $t \in dvar_{\mathcal{D}}(P \square S)$, $t\hat{\theta} \neq \bot$*) donde* $\mathcal{T} \equiv \mathbf{DF}_{\mathcal{P}}((f\overline{e}_n\overline{a}_k \to t)\hat{\theta} \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_n, \mathcal{T}_C, \mathcal{T}_r, \mathcal{T}_s])$ $(k > 0)$ *con* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i\hat{\theta} \to \overline{t}_i \Leftarrow \Pi$ *para cada* $1 \leq i \leq n$, $\mathcal{T}_C : \mathcal{P} \vdash_{\mathcal{D}} \overline{P} \square \overline{C} \Leftarrow \Pi$, $\mathcal{T}_r : \mathcal{P} \vdash_{\mathcal{D}} \overline{r} \to s \Leftarrow \Pi$ *y* $\mathcal{T}_s : \mathcal{P} \vdash_{\mathcal{D}} s\ \overline{a_k\hat{\theta}} \to t\hat{\theta} \Leftarrow \Pi$ *usando* $(f\overline{\overline{t}_n} \to \overline{r} \Leftarrow \overline{P} \square \overline{C}) \in [\mathcal{P}]_{\bot}$ *(es decir,* $(f\ \overline{t}_n \to r \Leftarrow P' \square C')\rho$ *con* $(f\overline{t}_n \to r \Leftarrow P' \square C') \in_{var} \mathcal{P}$ *y* $\rho \in Sub_{\mathcal{D}}$*). Definimos* $\hat{\theta}'(X_i) = \rho(X_i)$ *para cualquier* $X_i \in \overline{Y} \equiv var(f\overline{t}_n \to r \Leftarrow P' \square C')$, $\hat{\theta}'(X) = s$ *y* $\hat{\theta}'(Y) = \hat{\theta}(Y)$ *para cualquier otra variable* Y. *Gracias a que* X,$\overline{Y}$ *son nuevas variables, se obtiene directamente que* $\Pi \models_{\mathcal{D}} S\hat{\theta}'$, $Z\hat{\theta} \equiv s\hat{\theta}'$ *para cada* $\{Z \mapsto s\} \in \sigma$ *y* $\overline{\mathcal{T}} : \mathcal{P} \vdash_{\mathcal{D}} (P \square C)\hat{\theta}' \Leftarrow \Pi$. *Es más, debido a que* $\overline{t}_i \equiv t_i\rho \equiv t_i\hat{\theta}'$, *a partir de* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i\hat{\theta} \to \overline{t}_i \Leftarrow \Pi$ *también tenemos que* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} (e_i \to t_i)\hat{\theta}' \Leftarrow \Pi$ *para todo* $1 \leq i \leq n$. *Como* $\overline{P} \equiv P'\rho \equiv P'\hat{\theta}'$ *y* $\overline{C} \equiv C'\rho \equiv C'\hat{\theta}'$, *de* $\mathcal{T}_C : \mathcal{P} \vdash_{\mathcal{D}} \overline{P} \square \overline{C} \Leftarrow \Pi$ *tenemos* $\mathcal{T}_C : \mathcal{P} \vdash_{\mathcal{D}} (P' \square C')\hat{\theta}' \Leftarrow \Pi$. *Además,* $\overline{r} \equiv r\rho \equiv r\hat{\theta}'$ *y* $s = \hat{\theta}'(X)$, *luego de* $\mathcal{T}_r : \mathcal{P} \vdash_{\mathcal{D}} \overline{r} \to s \Leftarrow \Pi$ *se tiene que* $\mathcal{T}_r : \mathcal{P} \vdash_{\mathcal{D}} (r \to X)\hat{\theta}' \Leftarrow \Pi$. *Análogamente, a partir de* $\mathcal{T}_s : \mathcal{P} \vdash_{\mathcal{D}} s\ \overline{a_k\hat{\theta}} \to t\hat{\theta} \Leftarrow \Pi$ *también se tiene que* $\mathcal{T}_s : \mathcal{P} \vdash_{\mathcal{D}} (X\ a_k \to t)\hat{\theta}' \Leftarrow \Pi$. *Finalmente, si consideramos* $\theta' =_{\backslash evar(G')} \hat{\theta}'$ *entonces* $\mathcal{M}' : \Pi \square \theta' \in Ans_{\mathcal{P}}(G')$, *donde* $\mathcal{M}'$ *es el resultado de reemplazar el árbol de prueba* $\mathcal{T}$ *en* $\mathcal{M}$ *por los árboles de prueba* $\mathcal{T}_i$ $(1 \leq i \leq n)$, $\mathcal{T}_C$, $\mathcal{T}_r$ *y* $\mathcal{T}_s$. *Claramente,* $Sol_{\mathcal{D}}(\Pi \square \theta) \subseteq Sol_{\mathcal{D}}(\exists_{\backslash G}.\ \Pi \square \theta')$.

DF$_1$ *Completamente análoga al caso de* **DF$_2$** *e incluso más sencilla.*

FV$_1$ *Sea* $\mathcal{M} : \Pi \,\square\, \theta \in Ans_{\mathcal{P}}(G)$. *Ha de existir* $\hat{\theta} =_{\backslash evar(G)} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $X\hat{\theta} \equiv s\hat{\theta}$ *para cada* $\{X \mapsto s\} \in \sigma$, $\overline{\mathcal{T}} : \mathcal{P} \vdash_{\mathcal{D}} (P \,\square\, C)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (F\overline{e}_n \to t)\hat{\theta} \Leftarrow \Pi$. *Como* $t \notin \mathcal{V}ar$ *o bien* $t \in dvar_{\mathcal{D}}(P \,\square\, S)$, $t\hat{\theta} \neq \bot$. *Más aún,* F *es una variable demandada y también* $F\hat{\theta} \neq \bot$. *Puesto que* $k > 0$, *también sabemos que* $\hat{\theta}(F) \notin \mathcal{B}^{\mathcal{D}} \cup \mathcal{V}ar$ *(en otro caso, el árbol de prueba* $\mathcal{T}$ *no sería posible en el cálculo CRWL(*$\mathcal{D}$*)). Por tanto,* $\hat{\theta}(F)$ *ha de ser de la forma* $h\overline{t}_m \in Pat_{\mathcal{D}}$. *Definimos* $\hat{\theta}'(X_i) = t_i$ *para cada* $1 \leq i \leq m$, $\hat{\theta}'(F) = h\overline{t}_m$ *y* $\hat{\theta}'(Y) = \hat{\theta}(Y)$ *para todo* $Y \notin \{X_1, \ldots, X_m\, F\}$. *Se cumple que* $\sigma_0\hat{\theta}' = \hat{\theta}'$ *y que* $\sigma_0\hat{\theta}' =_{\backslash\{X_1,\ldots,X_m\}} \hat{\theta}$. *Puesto que* $\overline{X}_m$ *son variables nuevas, esto implica que* $\Pi \models_{\mathcal{D}} S\sigma_0\hat{\theta}'$, $X\hat{\theta}' \equiv s\hat{\theta}'$ *para cualquier* $\{X \mapsto s\} \in \sigma$, $\hat{\theta}'(F) \equiv \hat{\theta}'(h\overline{X}_m)$ *y* $\overline{\mathcal{T}} : \mathcal{P} \vdash_{\mathcal{D}} (P \,\square\, C)\sigma_0\hat{\theta}' \Leftarrow \Pi$. *Más aún, a partir de* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (F\overline{e}_n \to t)\hat{\theta} \Leftarrow \Pi$ *también tenemos que* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (h\overline{X}_m\overline{e}_n \to t)\sigma_0\hat{\theta}' \Leftarrow \Pi$. *Finalmente, si consideramos* $\theta' =_{\backslash evar(G')} \hat{\theta}'$ *entonces* $\mathcal{M} : \Pi \,\square\, \theta' \in Ans_{\mathcal{P}}(G')$. *Claramente,* $Sol_{\mathcal{D}}(\Pi \,\square\, \theta) \subseteq Sol_{\mathcal{D}}(\exists_{\backslash G}.\ \Pi \,\square\, \theta')$.

FV$_2$ *Similar al caso de* **FV$_1$**.

CS *Asumamos que* $\mathcal{M} : \Pi \,\square\, \theta \in Ans_{\mathcal{P}}(G)$. *Sabemos entonces que ha de existir* $\hat{\theta} =_{\backslash evar(G)} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $X\hat{\theta} \equiv t\hat{\theta}$ *para cada* $\{X \mapsto t\} \in \sigma$ *y* $\overline{\mathcal{T}} : \mathcal{P} \vdash_{\mathcal{D}} (P \,\square\, C)\hat{\theta} \Leftarrow \Pi$. *Para cualquier* $1 \leq i \leq k$, *definimos* $\hat{\theta}_i = umg(\hat{\theta}, \sigma_i)$ *como el* unificador de máxima generalidad [BN98] *de las sustituciones* $\hat{\theta}$ *y* σ_i *(existe* $\rho_i, \delta_i \in Sub_{\mathcal{D}}$ *para el que* $\hat{\theta}_i = \hat{\theta}\rho_i$ *y* $\hat{\theta}_i = \sigma_i\delta_i$*) y* $\Pi_i = \Pi\rho_i \wedge S_i\delta_i$. *Probamos que:*

- $\Pi_i \models_{\mathcal{D}} S_i\hat{\theta}_i$. *Sea* $\mu \in Sol_{\mathcal{D}}(\Pi_i)$. *Por definición de* Π_i, $\mu \in Sol_{\mathcal{D}}(\Pi\rho_i)$ *y* $\mu \in Sol_{\mathcal{D}}(S_i\delta_i)$. *De aquí,* $\rho_i\mu \in Sol_{\mathcal{D}}(\Pi)$ *y* $\delta_i\mu \in Sol_{\mathcal{D}}(S_i)$. *Puesto que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, *se tiene que* $\rho_i\mu \in Sol_{\mathcal{D}}(S\hat{\theta})$ *y que* $\delta_i\mu \in Sol_{\mathcal{D}}(S_i)$. *Por tanto, de aquí se sigue que* $\hat{\theta}\rho_i\mu \in Sol_{\mathcal{D}}(S)$ *y que* $\delta_i\mu \in Sol_{\mathcal{D}}(S_i)$. *Es más, como* $\hat{\theta}_i = \hat{\theta}\rho_i$, *se tiene que* $\hat{\theta}_i\mu \in Sol_{\mathcal{D}}(S)$ *y que* $\delta_i\mu \in Sol_{\mathcal{D}}(S_i)$ $(1 \leq i \leq k)$. *Usando ahora el requerimiento de un resolutor ideal de restricciones de que* $Sol_{\mathcal{D}}(S) = \bigcup_{i=1}^{k} Sol_{\mathcal{D}}(\exists_{\backslash S}.\ S_i \,\square\, \sigma_i)$, *se deduce que también* $\hat{\theta}_i\mu \in Sol_{\mathcal{D}}(S_i)$ *y entonces* $\mu \in Sol_{\mathcal{D}}(S_i\hat{\theta}_i)$.

- $X\hat{\theta}_i = t\hat{\theta}_i$ *para cada* $\{X \mapsto t\} \in \sigma\sigma_i$. *Si* $\{X \mapsto t\} \in \sigma$ *entonces* $X\hat{\theta}_i \equiv X\hat{\theta}\rho_i \equiv t\hat{\theta}\rho_i \equiv t\hat{\theta}_i$. *Si* $\{X \mapsto t\} \in \sigma_i$ *entonces* $X\hat{\theta}_i \equiv X\sigma_i\delta_i \equiv t\delta_i \equiv t\sigma_i\delta_i \equiv t\hat{\theta}_i$.

- $\overline{\mathcal{T}}_i : \mathcal{P} \vdash_{\mathcal{D}} (P \,\square\, C)\sigma_i\hat{\theta}_i \Leftarrow \Pi_i$. *Tenemos que* $((P \,\square\, C)\hat{\theta} \Leftarrow \Pi) \succcurlyeq_{\mathcal{D}} ((P \,\square\, C)\hat{\theta}_i \Leftarrow \Pi_i)$ *(si* $(e \to t) \in P$ *entonces existe* ρ_i *para el que* $\Pi_i \models_{\mathcal{D}} e\hat{\theta}\rho_i = e\hat{\theta}_i$, $\Pi_i \models_{\mathcal{D}} t\hat{\theta}\rho_i = t\hat{\theta}_i$ *y* $\Pi_i \models_{\mathcal{D}} \Pi\rho_i$. *Análogamente para cualquier* $(p\overline{e}_n \to! t) \in C$*) y* $\hat{\theta}_i = \sigma_i\delta_i = (\sigma_i\sigma_i)\delta_i = \sigma_i(\sigma_i\delta_i) = \sigma_i\hat{\theta}_i$. *Entonces,* $((P \,\square\, C)\hat{\theta} \Leftarrow \Pi) \succcurlyeq_{\mathcal{D}} ((P \,\square\, C)\sigma_i\hat{\theta}_i \Leftarrow \Pi_i)$. *Como además* $\overline{\mathcal{T}} : \mathcal{P} \vdash_{\mathcal{D}} (P \,\square\, C)\hat{\theta} \Leftarrow \Pi$, *usando la* Propiedad de Implicación, *también tenemos que* $\overline{\mathcal{T}}_i : \mathcal{P} \vdash_{\mathcal{D}} (P \,\square\, C)\sigma_i\hat{\theta}_i \Leftarrow \Pi_i$.

Si consideramos ahora $\theta_i =_{\setminus evar(G_i)} \hat{\theta}_i$ *para cualquier* $1 \leq i \leq k$, *entonces* $\mathcal{M}_i : \Pi_i \square \theta_i \in Ans_{\mathcal{P}}(G_i)$, *donde cada* $\mathcal{M}_i$ *es el resultado de reemplazar los árboles de prueba* $\overline{\mathcal{T}}$ *en* $\mathcal{M}$ *por los árboles de prueba* $\overline{\mathcal{T}}_i$. *Finalmente, probamos que* $Sol_{\mathcal{D}}(\Pi \square \theta) \subseteq \bigcup_{i=1}^{k} Sol_{\mathcal{D}}(\exists_{\setminus G}.\ \Pi_i \square \theta_i)$. *Sea* $\mu \in Sol_{\mathcal{D}}(\Pi \square \theta)$. *Por definición,* $\mu \in Sol_{\mathcal{D}}(\Pi)$ *y* $X\mu \equiv t\mu$ *para cada* $\{X \mapsto t\} \in \theta$. *Puesto que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, *se tiene que* $\mu \in Sol_{\mathcal{D}}(S\hat{\theta})$, *y de aquí que* $\hat{\theta}\mu \in Sol_{\mathcal{D}}(S)$. *Usando ahora de nuevo el requerimiento de que* $Sol_{\mathcal{D}}(S) = \bigcup_{i=1}^{k} Sol_{\mathcal{D}}(\exists_{\setminus S}.\ S_i \square \sigma_i)$, *se sigue que* $\mu \in Sol_{\mathcal{D}}(\exists_{\setminus G}.\ S_i\hat{\theta} \square \sigma_i\hat{\theta})$ $(1 \leq i \leq k)$. *Por definición, ha de existir* $\mu' =_{\setminus G} \mu$ *para el que* $\mu' \in Sol_{\mathcal{D}}(S_i\hat{\theta})$ *y* $X\mu' \equiv t\mu'$ *para cualquier* $\{X \mapsto t\} \in \sigma_i\hat{\theta}$. *Como además* $\hat{\theta}_i = umg(\hat{\theta}, \sigma_i)$, *obtenemos que* $X\mu' \equiv t\mu'$ *para cada* $\{X \mapsto t\} \in \hat{\theta}_i$. *Es más, debido a que* $\hat{\theta}_i = \sigma_i\delta_i$, $S_i\sigma_i = S_i$ *y* $\Pi_i = \Pi\rho_i \wedge S_i\delta_i$, *se obtiene que* $\mu' \in Sol_{\mathcal{D}}(\Pi_i)$. *A partir de las definiciones de* μ' *y de* $\hat{\theta}_i$, *se sigue que* $\mu \in Sol_{\mathcal{D}}(\exists_{\setminus G}.\ \Pi_i \square \theta_i)$ $(1 \leq i \leq k)$. *En consecuencia,* $\mu \in \bigcup_{i=1}^{k} Sol_{\mathcal{D}}(\exists_{\setminus G}.\ \Pi_i \square \theta_i)$.

AC *Análogo al caso de* **PF**.

□

A.3.3. Propiedad de particionado de deducciones en *CRWL*($\mathcal{D}$)

Demostramos a continuación la propiedad de particionado de deducciones en la lógica para la reescritura con restricciones *CRWL*($\mathcal{D}$). Este resultado será utilizado en la demostración de las principales propiedades del cálculo de estrechamiento *CDNC*($\mathcal{D}$).

Demostración 37 (Demostración del Lema 12) *Distinguimos varios casos:*

- $p = \epsilon$:
  (1) $\Rightarrow$ (2). *Asumamos* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} e \to t \Leftarrow \Pi$. *Tomando* $s \equiv t \in Pat_{\mathcal{D}}$ *se sigue que* $\mathcal{T}_1 \equiv \mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} e|_\epsilon = e \to t \Leftarrow \Pi$ *y que* $\mathcal{T}_2 : \mathcal{P} \vdash_{\mathcal{D}} e[t]_\epsilon = t \to t \Leftarrow \Pi$ *por la* Propiedad de Aproximación, *puesto que trivialmente* $\Pi \models_{\mathcal{D}} t \sqsupseteq t$. *Observamos que* $|\ \mathcal{T}_1\ | = |\ \mathcal{T}\ |$ *y que* $|\ \mathcal{T}_2\ | = 0 \leq |\ \mathcal{T}\ |$. *Por tanto,* $|\ \mathcal{T}_1\ |, |\ \mathcal{T}_2\ | \leq |\ \mathcal{T}\ |$.
  (2) $\Rightarrow$ (1). *Asumamos ahora* $\mathcal{T}_1 : \mathcal{P} \vdash_{\mathcal{D}} e|_\epsilon = e \to s \Leftarrow \Pi$ *y* $\mathcal{T}_2 : \mathcal{P} \vdash_{\mathcal{D}} e[s]_\epsilon = s \to t \Leftarrow \Pi$. *Como* $s \in Pat_{\mathcal{D}}$, *por la* Propiedad de Aproximación, *también tenemos que* $\Pi \models_{\mathcal{D}} s \sqsupseteq t$. *Finalmente, por la* Propiedad de Implicación, *obtenemos* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} e \to t \Leftarrow \Pi$.

- $p \neq \epsilon$: *razonamos por inducción sobre el tamaño de* $e \in Exp_{\mathcal{D}}$ *y distinguimos casos de acuerdo con la forma de* $t \in Pat_{\mathcal{D}}$ *y* $\Pi \subseteq PCon_{\mathcal{D}}$:

- $t = \bot$:
  (1) $\Rightarrow$ (2). *Asumamos* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} e \to \bot \Leftarrow \Pi$. *Tomando* $s \equiv \bot \in Pat_{\mathcal{D}}$ *se*

sigue que $\mathcal{T}_1 : \mathcal{P} \vdash_{\mathcal{D}} e|_p \to \bot \Leftarrow \Pi$ *y que* $\mathcal{T}_2 : \mathcal{P} \vdash_{\mathcal{D}} e[\bot]_p \to \bot \Leftarrow \Pi$ *usando la regla* **TI**. *En este caso,* $| \mathcal{T} | = | \mathcal{T}_1 | = | \mathcal{T}_2 | = 0$.
(2) $\Rightarrow$ (1). *Trivialmente,* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} e \to \bot \Leftarrow \Pi$ *usando* **TI**.

- $t \neq \bot$, $Insat_{\mathcal{D}}(\Pi)$*:*
(1) $\Leftrightarrow$ (2). *En este caso* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} e \to t \Leftarrow \Pi$, $\mathcal{T}_1 : \mathcal{P} \vdash_{\mathcal{D}} e|_p \to s \Leftarrow \Pi$ *y* $\mathcal{T}_2 : \mathcal{P} \vdash_{\mathcal{D}} e[s]_p \to t \Leftarrow \Pi$ *son deducciones triviales usando* **TI**. *Más aún,* $| \mathcal{T} | = | \mathcal{T}_1 | = | \mathcal{T}_2 | = 0$.

- $t \neq \bot$, $Sat_{\mathcal{D}}(\Pi)$. *Distinguimos nuevos casos:*

- $e \in Pat_{\mathcal{D}}$. *Probamos* (1) $\Rightarrow$ (2)*:*
En este caso, $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} e \to t \Leftarrow \Pi$ *es equivalente a* $\Pi \models_{\mathcal{D}} e \sqsupseteq t$ *por la* Propiedad de Aproximación. *Puesto que* $p \neq \epsilon$, $e = h\bar{e}_m \in Pat_{\mathcal{D}}$ *es pasivo. Entonces,* $p = i \cdot q$ *con* $1 \leq i \leq m$ *y* $q \in Pos(e_i)$. *Como además* $t \neq \bot$, *tenemos dos posibilidades para* t*:*

- $t = h\bar{t}_m$*:*
En este caso $\mathcal{T} \equiv$ **DC** $(h\bar{e}_m \to h\bar{t}_m \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_m])$, *donde* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$ *para todo* $1 \leq i \leq m$. *Como* $q \in Pos(e_i)$, *por* hipótesis de inducción *ha de existir* $s_i \in Pat_{\mathcal{D}}$ *para el que* $\mathcal{T}_{i1} : \mathcal{P} \vdash_{\mathcal{D}} e_i|_q \to s_i \Leftarrow \Pi$ *y* $\mathcal{T}_{i2} : \mathcal{P} \vdash_{\mathcal{D}} e_i[s_i]_q \to t_i \Leftarrow \Pi$ *tal que* $| \mathcal{T}_{i1} |, | \mathcal{T}_{i2} | \leq | \mathcal{T}_i |$. *En esta situación,* $e|_p = h\bar{e}_m|_{i \cdot q} = e_i|_q$ *y* $e[s]_p = h\bar{e}_m[s]_{i \cdot q} = h\ e_1 \ldots e_{i-1}\ e_i[s]_q\ e_{i+1} \ldots e_m$. *De este modo, tomando* $s \equiv s_i \in Pat_{\mathcal{D}}$ *se tiene que* $\mathcal{T}_1 \equiv \mathcal{T}_{i1} : \mathcal{P} \vdash_{\mathcal{D}} e|_p = e_i|_q \to s_i = s \Leftarrow \Pi$ *y que* $\mathcal{T}_2 : \mathcal{P} \vdash_{\mathcal{D}} e[s]_p = h\ e_1 \ldots e_{i-1}\ e_i[s_i]_q\ e_{i+1} \ldots e_m \to t \Leftarrow \Pi$ *donde* $\mathcal{T}_2 \equiv$ **DC** $(e[s_i]_p \to h\bar{t}_m \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_{i-1}, \mathcal{T}_{i2}, \mathcal{T}_{i+1}, \ldots, \mathcal{T}_m])$. *Más aún,* $| \mathcal{T}_1 | = | \mathcal{T}_{i1} | \leq | \mathcal{T}_i | \leq \sum_i | \mathcal{T}_i | = | \mathcal{T} |$ *y* $| \mathcal{T}_2 | = \sum_{\setminus i} | \mathcal{T}_i | + | \mathcal{T}_{i2} | \leq \sum_{\setminus i} | \mathcal{T}_i | + | \mathcal{T}_i | = | \mathcal{T} |$. *Por tanto,* $| \mathcal{T}_1 |, | \mathcal{T}_2 | \leq | \mathcal{T} |$.

- $t = X \in \mathcal{V}ar$*:*
En este caso $\mathcal{T} \equiv$ **IR** $(h\bar{e}_m \to X \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_m])$, *donde* $\Pi \models_{\mathcal{D}} h\bar{t}_m \sqsupseteq X$ *y* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$ *para todo* $1 \leq i \leq m$. *Como* $q \in Pos(e_i)$, *por* hipótesis de inducción, *ha de existir* $s_i \in Pat_{\mathcal{D}}$ *para el que* $\mathcal{T}_{i1} : \mathcal{P} \vdash_{\mathcal{D}} e_i|_q \to s_i \Leftarrow \Pi$ *y* $\mathcal{T}_{i2} : \mathcal{P} \vdash_{\mathcal{D}} e_i[s_i]_q \to t_i \Leftarrow \Pi$ *tal que* $| \mathcal{T}_{i1} |, | \mathcal{T}_{i2} | \leq | \mathcal{T}_i |$. *En esta situación,* $e|_p = h\bar{e}_m|_{i \cdot q} = e_i|_q$ *y* $e[s]_p = h\bar{e}_m[s]_{i \cdot q} = h\ e_1 \ldots e_{i-1}\ e_i[s]_q\ e_{i+1} \ldots e_m$. *De este modo, tomando* $s \equiv s_i \in Pat_{\mathcal{D}}$ *tenemos que* $\mathcal{T}_1 \equiv \mathcal{T}_{i1} : \mathcal{P} \vdash_{\mathcal{D}} e|_p = e_i|_q \to s_i = s \Leftarrow \Pi$ *y que* $\mathcal{T}_2 : \mathcal{P} \vdash_{\mathcal{D}} e[s]_p = h\ e_1 \ldots e_{i-1}\ e_i[s_i]_q\ e_{i+1} \ldots e_m \to t \Leftarrow \Pi$, *donde* $\mathcal{T}_2 \equiv$ **IR** $(e[s_i]_p \to X \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_{i-1}, \mathcal{T}_{i2}, \mathcal{T}_{i+1}, \ldots, \mathcal{T}_m])$ *debido a que* $\Pi \models_{\mathcal{D}} h\bar{t}_m \sqsupseteq X$. *Es más,* $| \mathcal{T}_1 | = | \mathcal{T}_{i1} | \leq | \mathcal{T}_i | \leq \sum_i | \mathcal{T}_i | = | \mathcal{T} |$ *y* $| \mathcal{T}_2 | = \sum_{\setminus i} | \mathcal{T}_i | + | \mathcal{T}_{i2} | \leq \sum_{\setminus i} | \mathcal{T}_i | + | \mathcal{T}_i | = | \mathcal{T} |$. *Por tanto,* $| \mathcal{T}_1 |, | \mathcal{T}_2 | \leq | \mathcal{T} |$.

- $e \in Pat_{\mathcal{D}}$. *Probamos* (2) $\Rightarrow$ (1)*:*
En este caso $\mathcal{T}_1 \equiv \mathcal{T}_{i1} : \mathcal{P} \vdash_{\mathcal{D}} e|_p = h\bar{e}_m|_{i \cdot q} = e_i|_q \to s \Leftarrow \Pi$ *y* $\mathcal{T}_2 : \mathcal{P} \vdash_{\mathcal{D}} e[s]_p$

$= h\bar{e}_m[s]_{i\cdot q} = h\ e_1\ \ldots\ e_{i-1}\ e_i[s]_q\ e_{i+1}\ \ldots\ e_m \to t \Leftarrow \Pi$. *Como* $t \neq \bot$ *tenemos de nuevo dos posibilidades para* t*:*

- $t = h\bar{t}_m$*:*
  En este caso $\mathcal{T}_2 \equiv \mathbf{DC}\ (h\ e_1\ \ldots\ e_{i-1}\ e_i[s]_q\ e_{i+1}\ \ldots\ e_m \to h\bar{t}_m \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_{i-1}, \mathcal{T}_{i2}, \mathcal{T}_{i+1}, \ldots, \mathcal{T}_m])$*, donde* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$ *para todo* $i \in \{1, \ldots, i-1, i+1, \ldots, m\}$ *y* $\mathcal{T}_{i2} : \mathcal{P} \vdash_{\mathcal{D}} e_i[s]_q \to t_i \Leftarrow \Pi$*. Por* hipótesis de inducción *también tenemos que* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$*. Entonces,* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} h\bar{e}_m \to h\bar{t}_m \Leftarrow \Pi$*, donde* $\mathcal{T} \equiv \mathbf{DC}\ (h\bar{e}_m \to h\bar{t}_m \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_{i-1}, \mathcal{T}_i, \mathcal{T}_{i+1}, \ldots, \mathcal{T}_m])$*.*

- $t = X \in \mathcal{V}ar$*:*
  En este caso $\mathcal{T}_2 \equiv \mathbf{IR}\ (h\ e_1\ \ldots\ e_{i-1}\ e_i[s]_q\ e_{i+1}\ \ldots\ e_m \to X \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_{i-1}, \mathcal{T}_{i2}, \mathcal{T}_{i+1}, \ldots, \mathcal{T}_m])$*, donde* $\Pi \models_{\mathcal{D}} h\bar{t}_m \sqsupseteq X$*,* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$ *para todo* $i \in \{1, \ldots, i-1, i+1, \ldots, m\}$ *y* $\mathcal{T}_{i2} : \mathcal{P} \vdash_{\mathcal{D}} e_i[s]_q \to t_i \Leftarrow \Pi$*. Por* hipótesis de inducción *también tenemos que* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$*. Entonces,* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} h\bar{e}_m \to X \Leftarrow \Pi$*, donde* $\Pi \models_{\mathcal{D}} h\bar{t}_m \sqsupseteq X$ *y* $\mathcal{T} \equiv \mathbf{IR}\ (h\bar{e}_m \to X \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_{i-1}, \mathcal{T}_i, \mathcal{T}_{i+1}, \ldots, \mathcal{T}_m])$*.*

- $e \notin Pat_{\mathcal{D}}$*. Distinguimos nuevos casos:*

- $e = h\bar{e}_m \notin Pat_{\mathcal{D}}$ *es pasivo. Este caso es análogo al caso anterior (ya que* $t \neq \bot$*,* $t = h\bar{t}_m$ *o* $t = X \in \mathcal{V}ar$*).*

- $e = f\bar{e}_n\bar{a}_k$ *con* $f \in DF^n$ $(k \geq 0)$*. Probamos* $(1) \Rightarrow (2)$*:*
  Asumimos $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} f\bar{e}_n\bar{a}_k \to t \Leftarrow \Pi$*. En este caso* $\mathcal{T} \equiv \mathbf{DF}_{\mathcal{P}}\ (f\bar{e}_n\bar{a}_k \to t \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_n, \mathcal{T}_c, \mathcal{T}_r, \mathcal{T}_s\ ])$*, donde* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$ *para todo* $1 \leq i \leq n$*,* $\mathcal{T}_p : \mathcal{P} \vdash_{\mathcal{D}} P\ \square\ C \Leftarrow \Pi$*,* $\mathcal{T}_r : \mathcal{P} \vdash_{\mathcal{D}} r \to s \Leftarrow \Pi$ *y* $\mathcal{T}_s : \mathcal{P} \vdash_{\mathcal{D}} s\bar{a}_k \to t \Leftarrow \Pi$*, usando* $(f\bar{t}_n \to r \Leftarrow P\ \square\ C) \in [\mathcal{P}]_\bot$ *y* $s \in Pat_{\mathcal{D}}$*. Como* $p = i \cdot q$ *con* $i \in \{1, \ldots, n+k\ \}$*, distinguimos dos casos:*

- $1 \leq i \leq n$*:*
  Como $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$ *y* $q \in Pos(e_i)$*, por* hipótesis de inducción *ha de existir* $s_i \in Pat_{\mathcal{D}}$ *tal que* $\mathcal{T}_{i1} : \mathcal{P} \vdash_{\mathcal{D}} e_i|_q \to s_i \Leftarrow \Pi$*,* $\mathcal{T}_{i2} : \mathcal{P} \vdash_{\mathcal{D}} e_i[s_i]_q \to t_i \Leftarrow \Pi$ *y* $\mid \mathcal{T}_{i1} \mid, \mid \mathcal{T}_{i2} \mid \leq \mid \mathcal{T}_i \mid$*. En esta situación,* $\mathcal{T}_1 \equiv \mathcal{T}_{i1} : \mathcal{P} \vdash_{\mathcal{D}} f\bar{e}_n\bar{a}_k|_{i\cdot q} = e_i|_q \to s_i \Leftarrow \Pi$ *y* $\mathcal{T}_2 : \mathcal{P} \vdash_{\mathcal{D}} f\bar{e}_n\bar{a}_k[s_i]_p \to t \Leftarrow \Pi$*, donde* $\mathcal{T}_2 \equiv \mathbf{DF}_{\mathcal{P}}\ (fe_1\ \ldots\ e_{i-1}\ e_i[s_i]_q\ e_{i+1}\ \ldots\ e_n\ \bar{a}_k \to t \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_{i-1}, \mathcal{T}_{i2}, \mathcal{T}_{i+1}, \ldots, \mathcal{T}_n, \mathcal{T}_c, \mathcal{T}_r, \mathcal{T}_s\ ])$*, usando* $(f\bar{t}_n \to r \Leftarrow P\ \square\ C) \in [\mathcal{P}]_\bot$ *y* $s \in Pat_{\mathcal{D}}$*. Más aún,* $\mid \mathcal{T}_1 \mid = \mid \mathcal{T}_{i1} \mid \leq \mid \mathcal{T}_i \mid \leq \mid \mathcal{T}_i \mid + \sum_{\setminus i} \mid \mathcal{T}_i \mid + \mid \mathcal{T}_c \mid + \mid \mathcal{T}_r \mid + \mid \mathcal{T}_s \mid + 1 = \mid \mathcal{T} \mid$ *y* $\mid \mathcal{T}_2 \mid = 1 + \mid \mathcal{T}_{i2} \mid + \sum_{\setminus i} \mid \mathcal{T}_i \mid + \mid \mathcal{T}_c \mid + \mid \mathcal{T}_r \mid + \mid \mathcal{T}_s \mid \leq 1 + \mid \mathcal{T}_i \mid + \sum_{\setminus i} \mid \mathcal{T}_i \mid + \mid \mathcal{T}_c \mid + \mid \mathcal{T}_r \mid + \mid \mathcal{T}_s \mid = \mid \mathcal{T} \mid$*. Por tanto,* $\mid \mathcal{T}_1 \mid, \mid \mathcal{T}_2 \mid \leq \mid \mathcal{T} \mid$*.*

- $n+1 \leq i \leq k+1$*:*
  Sabemos que $\mathcal{T}_s : \mathcal{P} \vdash_{\mathcal{D}} s\bar{a}_k \to t \Leftarrow \Pi$*. Como además* $q \in Pos(a_i)$*, ha de existir*

$q' \in Pos(s\overline{a}_k)$. *Por* hipótesis de inducción, *existe* $s_i \in Pat_{\mathcal{D}}$ *tal que* $\mathcal{T}_{s1} : \mathcal{P} \vdash_{\mathcal{D}} s\overline{a}_k|_{q'} = a_i|_q \to s_i \Leftarrow \Pi$, $\mathcal{T}_{s2} : \mathcal{P} \vdash_{\mathcal{D}} s\overline{a}_k[s_i]_{q'} = s\ a_1\ \ldots\ a_{i-1}\ a_i[s_i]_q\ a_{i+1}\ \ldots\ a_k \to t \Leftarrow \Pi\ y\ |\ \mathcal{T}_{s1}\ |, |\ \mathcal{T}_{s2}\ | \leq |\ \mathcal{T}_s\ |$. *En esta situación,* $\mathcal{T}_1 \equiv \mathcal{T}_{s1} : \mathcal{P} \vdash_{\mathcal{D}} f\overline{e}_n\overline{a}_k|_{i\cdot q} = a_i|_q \to s_i \Leftarrow \Pi\ y\ \mathcal{T}_2 : \mathcal{P} \vdash_{\mathcal{D}} f\overline{e}_n\overline{a}_k[s_i]_{i\cdot q} \to t \Leftarrow \Pi$, *donde* $\mathcal{T}_2 \equiv \mathbf{DF}_{\mathcal{P}}\ (f\overline{e}_n\ a_1\ \ldots\ a_{i-1}\ a_i[s_i]_q\ a_{i+1}\ \ldots\ a_k \to t \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_n, \mathcal{T}_c, \mathcal{T}_r, \mathcal{T}_{s2}])$. *Más aún,* $|\ \mathcal{T}_1\ | = |\ \mathcal{T}_{s1}\ | \leq |\ \mathcal{T}_s\ | \leq 1 + \sum_i |\ \mathcal{T}_i\ | + |\ \mathcal{T}_c\ | + |\ \mathcal{T}_r\ | + |\ \mathcal{T}_s\ | = |\ \mathcal{T}\ |\ y\ |\ \mathcal{T}_2\ | = 1 + \sum_i |\ \mathcal{T}_i\ | + |\ \mathcal{T}_c\ | + |\ \mathcal{T}_r\ | + |\ \mathcal{T}_{s2}\ | \leq 1 + \sum_i |\ \mathcal{T}_i\ | + |\ \mathcal{T}_c\ | + |\ \mathcal{T}_r\ | + |\ \mathcal{T}_s\ | = |\ \mathcal{T}\ |$. *Por tanto,* $|\ \mathcal{T}_1\ |, |\ \mathcal{T}_2\ | \leq |\ \mathcal{T}\ |$.

- $e = f\overline{e}_n\overline{a}_k$ *con* $f \in DF^n$ $(k \geq 0)$. *Probamos* $(2) \Rightarrow (1)$*:*
  Asumimos $\mathcal{T}_1 : \mathcal{P} \vdash_{\mathcal{D}} f\overline{e}_n\overline{a}_k|_p \to s \Leftarrow \Pi\ y\ \mathcal{T}_2 : \mathcal{P} \vdash_{\mathcal{D}} f\overline{e}_n\overline{a}_k[s]_p \to t \Leftarrow \Pi$. *Puesto que* $p = i \cdot q$ *con* $i \in \{1, \ldots, n+k\}$, *distinguimos de nuevo dos casos:*

- $1 \leq i \leq n$*:*
  En este caso $\mathcal{T}_1 : \mathcal{P} \vdash_{\mathcal{D}} e_i|_q \to s \Leftarrow \Pi\ y\ \mathcal{T}_2 : \mathcal{P} \vdash_{\mathcal{D}} f\ e_1\ \ldots\ e_{i-1}\ e_i[s]_q\ e_{i+1}\ \ldots\ e_n\ \overline{a}_k \to t \Leftarrow \Pi$. *Observamos que* $\mathcal{T}_2 \equiv \mathbf{DF}_{\mathcal{P}}\ (f\ e_1\ \ldots\ e_{i-1}\ e_i[s]_q\ e_{i+1}\ \ldots\ e_n\ \overline{a}_k \to t \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_{i-1}, \mathcal{T}_{i2}, \mathcal{T}_{i+1}, \ldots, \mathcal{T}_n, \mathcal{T}_c, \mathcal{T}_r, \mathcal{T}_{s'}])$, *donde* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$ *para todo* $i \in \{1, \ldots, i-1, i+1, \ldots, n\}$, $\mathcal{T}_{i2} : \mathcal{P} \vdash_{\mathcal{D}} e_i[s]_q \to t_i \Leftarrow \Pi$, $\mathcal{T}_c : \mathcal{P} \vdash_{\mathcal{D}} P\ \square\ C \Leftarrow \Pi$, $\mathcal{T}_r : \mathcal{P} \vdash_{\mathcal{D}} r \to s' \Leftarrow \Pi\ y\ \mathcal{T}_{s'} : \mathcal{P} \vdash_{\mathcal{D}} s'\overline{a}_k \to t \Leftarrow \Pi$, *usando* $(f\overline{t}_n \to r \Leftarrow P\ \square\ C) \in [\mathcal{P}]_\perp\ y\ s' \in Pat_{\mathcal{D}}$. *Como además* $\mathcal{T}_{i1} \equiv \mathcal{T}_1 : \mathcal{P} \vdash_{\mathcal{D}} e_i|_q \to s \Leftarrow \Pi\ y\ \mathcal{T}_{i2} : \mathcal{P} \vdash_{\mathcal{D}} e_i[s]_q \to t_i \Leftarrow \Pi$, *por* hipótesis de inducción, $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$. *Entonces* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} f\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi$, *donde* $\mathcal{T} \equiv \mathbf{DF}_{\mathcal{P}}\ (f\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_{i-1}, \mathcal{T}_i, \mathcal{T}_{i+1}, \ldots, \mathcal{T}_n, \mathcal{T}_c, \mathcal{T}_r, \mathcal{T}_{s'}])$.

- $n+1 \leq i\ k+1$*:*
  En este caso $\mathcal{T}_1 : \mathcal{P} \vdash_{\mathcal{D}} a_i|_q \to s \Leftarrow \Pi\ y\ \mathcal{T}_2 : \mathcal{P} \vdash_{\mathcal{D}} f\ \overline{e}_n\ a_1\ \ldots\ a_{i-1}\ a_i[s]_q\ a_{i+1}\ \ldots\ a_k \to t \Leftarrow \Pi$. *Observamos que* $\mathcal{T}_2 \equiv \mathbf{DF}_{\mathcal{P}}\ (f\ \overline{e}_n\ a_1\ \ldots\ a_{i-1}\ a_i[s]_q\ a_{i+1}\ \ldots\ a_k \to t \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_n, \mathcal{T}_c, \mathcal{T}_r, \mathcal{T}_{s'2}])$, *donde* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$ *para todo* $i \in \{1, \ldots, n\}$, $\mathcal{T}_c : \mathcal{P} \vdash_{\mathcal{D}} P\ \square\ C \Leftarrow \Pi$, $\mathcal{T}_r : \mathcal{P} \vdash_{\mathcal{D}} r \to s' \Leftarrow \Pi\ y\ \mathcal{T}_{s'2} : \mathcal{P} \vdash_{\mathcal{D}} s'\ a_1\ \ldots\ a_{i-1}\ a_i[s]_q\ a_{i+1}\ \ldots\ a_k \to t \Leftarrow \Pi$, *usando* $(f\overline{t}_n \to r \Leftarrow P\ \square\ C) \in [\mathcal{P}]_\perp\ y\ s' \in Pat_{\mathcal{D}}$. *Ya que* $q \in Pos(a_i)$, *ha de existir* $q' \in Pos(s'\overline{a}_k)$ *para el que* $\mathcal{T}_{s'1} \equiv \mathcal{T}_1 : \mathcal{P} \vdash_{\mathcal{D}} s'\overline{a}_k|_{q'} = a_i|_q \to s \Leftarrow \Pi\ y\ \mathcal{T}_{s'2} : \mathcal{P} \vdash_{\mathcal{D}} s'\overline{a}_k[s]_{q'} = s'\ a_1\ \ldots\ a_{i-1}\ a_i[s]_q\ a_{i+1}\ \ldots\ a_k \to t \Leftarrow \Pi$. *Por* hipótesis de inducción, $\mathcal{T}_{s'} : \mathcal{P} \vdash_{\mathcal{D}} s'\overline{a}_k \to t \Leftarrow \Pi$. *Entonces* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} f\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi$, *donde* $\mathcal{T} \equiv \mathbf{DF}_{\mathcal{P}}\ (f\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_n, \mathcal{T}_c, \mathcal{T}_r, \mathcal{T}_{s'}])$.

- $e = p\overline{e}_n$ *con* $p \in PF^n$. *Probamos* $(1) \Rightarrow (2)$*:*
  Asumimos $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} p\overline{e}_n \to t \Leftarrow \Pi$. *En este caso* $\mathcal{T} \equiv \mathbf{PF}\ (p\overline{e}_n \to t \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_n])$, *donde* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i \to t_i \Leftarrow \Pi$ *para todo* $1 \leq i \leq n\ y\ \Pi \models_{\mathcal{D}} p\overline{t}_n \to t$. *Como* $i \cdot q \in Pos(p\overline{e}_n)$, *se tiene que* $q \in Pos(e_i)$. *Por* hipótesis de inducción, *ha de existir* $s_i \in Pat_{\mathcal{D}}$ *tal que* $\mathcal{T}_{i1} : \mathcal{P} \vdash_{\mathcal{D}} e_i|_q \to s_i \Leftarrow \Pi$, $\mathcal{T}_{i2} : \mathcal{P} \vdash_{\mathcal{D}} e_i[s_i]_q \to t_i \Leftarrow \Pi\ y\ |\ \mathcal{T}_{i1}\ |, |\ \mathcal{T}_{i2}\ | \leq |\ \mathcal{T}_i\ |$. *Tomando* $s \equiv s_i$, *tenemos* $\mathcal{T}_1 \equiv \mathcal{T}_{i1} : \mathcal{P}$

$\vdash_{\mathcal{D}} p\bar{e}_n|_{i\cdot q} = e_i|_q \rightarrow s_i \Leftarrow \Pi$ *y* $\mathcal{T}_2 : \mathcal{P} \vdash_{\mathcal{D}} p\bar{e}_n[s_i]_{i\cdot q} \rightarrow t \Leftarrow \Pi$, *donde* $\mathcal{T}_2 \equiv$ **PF** $(p\ e_1\ \ldots\ e_{i-1}\ e_i[s_i]_q\ e_{i+1}\ \ldots\ e_n \rightarrow t \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_{i-1}, \mathcal{T}_{i2}, \mathcal{T}_{i+1}, \ldots, \mathcal{T}_n\ ])$ *debido a que* $\Pi \models_{\mathcal{D}} p\bar{t}_n \rightarrow t$. *Es más,* $\mid \mathcal{T}_1 \mid = \mid \mathcal{T}_{i1} \mid \leq \mid \mathcal{T}_i \mid \leq \mid \mathcal{T}_i \mid + \sum_{\setminus i} \mid \mathcal{T}_i \mid + 1 = \mid \mathcal{T} \mid y \mid \mathcal{T}_2 \mid = 1 + \mid \mathcal{T}_{i2} \mid + \sum_{\setminus i} \mid \mathcal{T}_i \mid \leq 1 + \mid \mathcal{T}_i \mid + \sum_{\setminus i} \mid \mathcal{T}_i \mid = \mid \mathcal{T} \mid$. *Por tanto,* $\mid \mathcal{T}_1 \mid, \mid \mathcal{T}_2 \mid \leq \mid \mathcal{T} \mid$.

- $e = p\bar{e}_n$ *con* $p \in PF^n$. *Probamos* (2) $\Rightarrow$ (1)*:*
  Asumimos $\mathcal{T}_1 : \mathcal{P} \vdash_{\mathcal{D}} p\bar{e}_n|_{i\cdot q} = e_i|_q \rightarrow s \Leftarrow \Pi$ *y* $\mathcal{T}_2 : \mathcal{P} \vdash_{\mathcal{D}} p\bar{e}_n[s]_{i\cdot q} = p\ e_1\ \ldots\ e_{i-1}\ e_i[s]_q\ e_{i+1}\ \ldots\ e_n \rightarrow t \Leftarrow \Pi$ *con* $s \in Pat_{\mathcal{D}}$ *y* $q \in Pos(e_i)$. *En esta situación,* $\mathcal{T}_2 \equiv$ **PF** $(p\ e_1\ \ldots\ e_{i-1}\ e_i[s]_q\ e_{i+1}\ \ldots\ e_n \rightarrow t \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_{i-1}, \mathcal{T}_{i2}, \mathcal{T}_{i+1}, \ldots, \mathcal{T}_n\ ])$, *donde* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i \rightarrow t_i \Leftarrow \Pi$ *para todo* $i \in \{1, \ldots, i-1, i+1, \ldots, n\}$, $\mathcal{T}_{i2} : \mathcal{P} \vdash_{\mathcal{D}} e_i[s]_q \rightarrow t_i \Leftarrow \Pi$ *y* $\Pi \models_{\mathcal{D}} p\bar{t}_n \rightarrow t$. *Como además* $\mathcal{T}_{i1} \equiv \mathcal{T}_1 : \mathcal{P} \vdash_{\mathcal{D}} e_i|_q \rightarrow s \Leftarrow \Pi$ *y* $\mathcal{T}_{i2} : \mathcal{P} \vdash_{\mathcal{D}} e_i[s]_q \rightarrow t_i \Leftarrow \Pi$, *por* hipótesis de inducción, $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i \rightarrow t_i \Leftarrow \Pi$. *Entonces,* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} p\bar{e}_n \rightarrow t \Leftarrow \Pi$, *tomando* $\mathcal{T} \equiv$ **PF** $(p\bar{e}_n \rightarrow t \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_{i-1}, \mathcal{T}_i, \mathcal{T}_{i+1}, \ldots, \mathcal{T}_n\ ])$ *debido a que* $\Pi \models_{\mathcal{D}} p\bar{t}_n \rightarrow t$.

□

A.3.4. Lema de corrección del cálculo $CDNC(\mathcal{D})$

Demostramos en esta subsección las principales propiedades del cálculo de resolución de objetivos $CDNC(\mathcal{D})$ con respecto a su corrección.

Demostración 38 (Demostración del Lema 14) *Dentro de esta demostración asumimos que, para cada regla del cálculo* $CDNC(\mathcal{D})$, G, G' *y* G_i *son exactamente como se han presentado en el Capítulo 4.*

(1) Para cada regla del cálculo $CDNC(\mathcal{D})$, *comenzamos proporcionando breves explicaciones justificando la preservación de las condiciones de admisibilidad de objetivos dadas en la Definición 24.*

SS
LN: *Puesto que* X *es una variable producida y debido a la linealidad de* G, σ_0 *no modifica los lados derechos de* P.
EX: $pvar(P') = pvar(P) \setminus \{X\} \subseteq evar(G) \setminus \{X\} = evar(G')$.
NC: *Como* X *es producida, sólo las producciones* $e \rightarrow Z$ *con* $X \in var(e)$ *se ven afectadas por* σ_0. *En estos casos, para cada* $Y \in var(t)$ *se crea la relación* $Y \gg_P Z$, *donde previamente teníamos* $Y \gg_P X$, $X \gg_P Z$. *Por tanto,* $\gg_P^*$ *no se ve alargada.*
SL: σ_0 *no modifica* σ *y* $pvar(P') \subseteq pvar(P)$.
DT: σ_0 *no modifica las variables de los árboles de producciones demandadas en* P *ni en* t. *Es más,* $t \rightarrow X$ *no es una producción demandada y* X *no aparece en* $pvar(P)$ *debido a la condición de admisibilidad* **LN** *en* G.

IM
LN: *Debido a la linealidad de G y a que $X_1, \ldots, X_m$ son variables nuevas, σ_0 no modifica las variables en los lados derechos de las producciones en P y cada variable nueva es producida una sola vez.*
EX: $pvar(P') = (pvar(P) \setminus \{X\}) \cup \{\overline{X}_m\} \subseteq (evar(G) \setminus \{X\}) \cup \{\overline{X}_m\} = evar(G')$.
NC: *Puesto que X es producida, X no aparece en $\overline{e}_m$. De este modo, solamente las producciones $e \to X'$ en P con $X \in var(e)$ se ven afectadas por σ_0. En estos casos, para cada $Y \in var(h\overline{e}_m)$ se crea la relación $Y \gg_P X'$, donde previamente teníamos $Y \gg_P X$ y $X \gg_P X'$. Por tanto, $\gg_P^*$ no se ve alargada.*
SL: *σ_0 no modifica σ y $pvar(P') = (pvar(P) \setminus \{X\}) \cup \{\overline{X}_m\}$ con $\overline{X}_m$ variables nuevas.*
DT: *σ_0 solamente introduce variables nuevas y no modifica las variables en los árboles de producciones demandadas en P. Es más, todas las producciones introducidas no poseen un árbol. De este modo, $h\overline{e}_m \to X$ no es una producción con un árbol y X no aparece en $pvar(P)$ debido a la condición de admisibilidad* **LN** *en G.*

EL
LN, NC, SL, DT*: Trivial.*
EX*: $pvar(P') = pvar(P) \setminus \{X\} \subseteq evar(G) \setminus \{X\} = evar(G')$.*

PF
LN*: Trivial, puesto que $\overline{X}_q$ son variables nuevas y distintas.*
EX*: Todas las nuevas variables $\overline{X}_q$ están cuantificadas existencialmente.*
NC*: Las variables $\overline{X}_q$ son nuevas, y por tanto no aparecen en ningún lado izquierdo de producciones en G'. En consecuencia, ningún ciclo de variables producidas se crea.*
SL*: σ no se modifica y las variables $\overline{X}_q$ son nuevas.*
DT*: Trivial, ya que $p\overline{e}_n \to X$ no es una producción con un árbol, todas las producciones introducidas $e_q \to X_q$ no tienen un árbol y $\overline{X}_q$ son variables nuevas.*

DT_1
LN, EX, NC, SL*: Trivial.*
DT*: Todas las variables en $\mathcal{T}_{f\overline{X}_n}$ son nuevas y $X \in dvar_{\mathcal{D}}(P \ \square \ S)$.*

DT_2
LN*: Trivial, ya que X' es una nueva variable.*
EX*: La nueva variable X' está cuantificada existencialmente.*
NC*: Como X' es una variable producida nueva, solamente aparece en $X'\overline{a}_k \to X$. En este caso, para cada variable $Y \in var(f\overline{e}_n)$, se crea la relación $Y \gg_P X'$, $X' \gg_P X$ donde previamente teníamos $Y \gg_P X$. Por tanto, ningún ciclo de variables producidas es creado debido a que $\gg_P^*$ no se ve extendida.*

SL: *σ no se modifica y la variable X' es nueva.*
DT: *La variable X' es demandada por una suspensión $X'\overline{a}_k \to X$ con $X \in dvar_{\mathcal{D}}(P \square S)$ (véase la* Definition 25*) y todas las variables en $\mathcal{T}_{f\overline{X}_n}$ son nuevas.*

FV
LN: *Como F no es una variable producida, σ_0 no modifica los lados derechos de las producciones en G'.*
EX: *Por la misma razón, $pvar(P') = \{X\} \cup pvar(P) \subseteq evar(G) \subseteq evar(G) \cup \{\overline{X}_p\} = evar(G')$.*
NC: *Gracias a que F no es una variable producida, solamente las producciones $e \to Z \in P$ con $F \in var(e)$ son las que se ven afectadas por σ_0. En tales casos, para cada $Y \in var(h\overline{X}_p)$, se crea la relación $Y \gg_P Z$. Sin embargo, $Y \gg_P Z$ no puede formar parte de ningún ciclo de variables, porque cada variable $Y \in var(h\overline{X}_p)$ es nueva y no aparece en ningún lado derecho de una producción en el nuevo objetivo.*
SL: *Como F no es producida y $\overline{X}_p$ son variables nuevas, σ_0 solo puede introducir variables nuevas que no son producidas en la sustitución respuesta.*
DT: *Por la misma razón, σ_0 solo introduce variables nuevas y no modifica las variables de los árboles de producciones demandadas en P.*

CSS
LN, EX, NC, SL: *Trivial.*
DT: *Trivial. P y S no se ven modificadas y puesto que $e|_{pos(X,\tau)}$ no es una variable, el árbol $\underline{caso}$ $(\tau, X, [\mathcal{T}_1, \ldots, \mathcal{T}_k])$ no define una variable demandada.*

DI
LN: *Como Y no es una variable producida, σ_0 no modifica las variables en el lado derecho de las producciones de P.*
EX: *Por la misma razón, $pvar(P)$ no se ve modificado, mientras que R, $\overline{U}$ es extendido a $\overline{Y}_{m_i}$, R, $\overline{U}$.*
NC: *Gracias a que G es un objetivo admisible y $\overline{Y}_{m_i}$ son variables nuevas, tras la aplicación de σ_0, estas variables pueden aparecer sólo en los lados izquierdos de producciones en P' pero no en los lados derechos. En consecuencia, no se crea ningún ciclo de variables producidas en G'.*
SL: *Ni Y ni $\overline{Y}_{m_i}$ contienen variables producidas.*
DT: *σ_0 solamente introduce variables nuevas y el árbol $\underline{caso}$ $(\tau, X, [\mathcal{T}_1, \ldots, \mathcal{T}_k])$ define la variable $Y \notin pvar(P)$.*

DN
LN: *R' es una nueva variable nueva.*
EX: *$pvar(P') = pvar(P) \cup \{R\} \cup \{R'\} \subseteq evar(G) \cup \{R'\} = evar(G')$.*
NC: *Puesto que $R' \in var(e[R']_{pos(X,\tau)})$, $R' \gg_P R$. En este caso, para cada $Y \in var(e|_{pos(X,\tau)})$, $Y \gg_P R'$. Sin embargo, como $Y \in var(e)$, se sigue que $Y \gg_P R$*

directamente. De este modo, R no aparece en e (y por tanto, no aparece en $e|_{pos(X,\tau)}$) debido a la condición de admisibilidad **NC**, *y R' tampoco aparece en e debido a que es una variable nueva. Por tanto, no se crean ciclos de variables producidas.*
SL: *σ no se modifica y la variable R' es nueva.*
DT: *R' es una variable nueva. Más aún, R' es una variable demandada porque $R' = e[R']_{pos(X,\tau)}|_{pos(X,\tau)}$ para $< e[R']_{pos(X,\tau)}$, caso $(\tau, X, [\mathcal{T}_1, \ldots, \mathcal{T}_k]) > \rightarrow R$, donde R es demandada. Finalmente, P y S no se modifican y $e|_{pos(X,\tau)}$ no es una variable.*

RRA
LN: *Puesto que $\tau \rightarrow r_i \Leftarrow P_i \ \square \ C_i$ es una variante de una regla de programa en $\mathcal{P}$, τ es lineal y $\overline{R}_m \in var(\tau)$ son variables nuevas y distintas. Más aún, las variables en los lados derechos de las producciones en P_i son también nuevas y distintas con respecto a las variables de τ (véase la definición de CFLP($\mathcal{D}$)-programa dada en la Sección 2.6), y por tanto con respecto a las variables $\overline{R}_m$. Por la misma razón, σ_c no modifica las variables en los lados derechos de P_i.*
EX: *$pvar(P') = pvar(P) \cup pvar(P_i\sigma_c) \cup \{\overline{R}_m\} \cup \{R\} = (pvar(P) \cup \{R\}) \cup (pvar(P_i) \cup \{\overline{R}_m\}) \subseteq evar(G) \cup \{\overline{X}\} = evar(G')$.*
NC: *Ni R ni las variables nuevas $\overline{R}_m$ aparecen en P y e (y así tampoco aparecen en $\sigma_f(R_j)$) por la condición de admisibilidad* **NC** *en G. Si $R_j \in var(\sigma_c(r_i))$, $R_j \gg_P R$ y para cada $Y \in var(\sigma_f(R_j))$, $Y \gg_P R_j$. Sin embargo, como $Y \in var(e)$, se sigue que $Y \gg_P R$ directamente. Por otra parte, R no aparece en $P_i\sigma_c$, y los lados derechos de $P_i\sigma_c$ no aparecen en $var(\sigma_f(R_j))$. En consecuencia, $\gg_P^*$ no se ve extendida.*
SL: *σ no se ve modificada y $\overline{R}_m$ y las variables producidas de $P_i\sigma_c$ son nuevas.*
DT: *Las variables en regla $(\tau \rightarrow r_1 \Leftarrow P_1 \ \square \ C_1 \mid \ldots \mid r_k \Leftarrow P_k \ \square \ C_k)$ no aparecen en el resto del objetivo. Cuando son eliminadas, estas variables son nuevas en el objetivo G'. Es más, en $P_i\sigma_c$ no hay producciones con un árbol asociado.*

CS
LN: *Como $\chi = pvar(P)$ y el resolutor de restricciones satisface el requerimiento de que $\chi \cap (dom(\sigma_i) \cup ran(\sigma_i)) = \emptyset$, tenemos que $pvar(P) \cap dom(\sigma_i) = \emptyset$ y entonces σ_i no modifica los lados derechos de las producciones en P.*
EX: *Por la misma razón, $pvar(P_i) = pvar(P) \subseteq evar(G) \subseteq evar(G) \cup \{\overline{Y}_i\} = evar(G_i)$.*
NC: *Puesto que también $pvar(P) \cap ran(\sigma_i) = \emptyset$ e $\overline{Y}_i$ son nuevas, ninguna variable producida es introducida por σ_i en los nuevos lados izquierdos de producciones en P. Por tanto, ningún ciclo de variables producidas se crea en G.*
SL: *Puesto que $pvar(P_i) = pvar(P)$ y $pvar(P) \cap var(\sigma_i) = \emptyset$, ninguna variable producida entra en la sustitución respuesta.*
DT: *σ_i solamente introduce variables nuevas y no modifica las variables en los lados derechos de producciones en P. Más aún, por los requerimientos pedidos a un resolutor de restricciones, se tiene que $var(S_i) \cap pvar(P) = \emptyset$ o bien que $dvar_{\mathcal{D}}(S_i)$*

$\cap\, pvar(P) \neq \emptyset$. *Como los lados derechos de producciones demandadas son variables producidas y demandadas, si están demandadas por S entonces también aparecen en S_i y son demandadas en G_i por S_i. Análogamente si estas variables están siendo demandadas por P.*

AC
LN*: Trivial, ya que $\overline{X}_q$ son variables nuevas y distintas entre sí.*
EX*: Todas las variables nuevas $\overline{X}_q$ están cuantificadas existencialmente.*
NC*: Las variables $\overline{X}_q$ son nuevas, y por tanto no aparecen en ningún lado izquierdo de las producciones de G'. Así pues, no se crea ningún ciclo de variables producidas.*
SL*: σ no se modifica y las variables $\overline{X}_q$ son nuevas.*
DT*: Trivial, ya que todas las nuevas producciones introducidas no son producciones con un árbol definicional, $\overline{X}_q$ son variables nuevas, y ni P ni S se modifican.*

(2) Procedemos considerando las reglas de fallo de $CDNC(\mathcal{D})$ una a una.

CC *Asumamos que $\Pi \;\square\; \theta \in Ans_{\mathcal{P}}(G)$. Por definición, existe una sustitución $\hat{\theta}$ tal que $\hat{\theta} =_{\backslash evar(G)} \theta$ y $< e, \underline{caso}\ (\tau, X, [\mathcal{T}_1, \ldots, \mathcal{T}_k]) > \rightarrow R$ es $CRWL(\mathcal{D})$-deducible para $e\hat{\theta} \rightarrow R\hat{\theta} \Leftarrow \Pi$, usando en el último paso la $CRWL(\mathcal{D})$-regla $\mathbf{DF}_{\mathcal{P}}$ con una instancia parcial $(f\bar{t}_n \rightarrow r \Leftarrow P \;\square\; C) \in [\mathcal{P}]_{\perp}$ ($e\hat{\theta} = f\overline{e_n\hat{\theta}}$ y R es una variable demandada con $\hat{\theta}(R) \neq \perp$, de acuerdo con el* Lema de Demanda*). Se sigue que $\mathcal{P} \vdash_{\mathcal{D}} e_i\hat{\theta} \rightarrow t_i \Leftarrow \Pi$ es $CRWL(\mathcal{D})$-deducible para todo $1 \leq i \leq n$. Como la forma de $e|_{pos(X,\tau)} = h \ldots$, $e\hat{\theta}$ tiene el símbolo h en un argumento $e_i\hat{\theta}$. Puesto que $\tau \preccurlyeq f\bar{t}_n$, $f\bar{t}_n$ tiene el símbolo h_j $(1 \leq j \leq k)$ en el argumento t_i. Además, como $h \notin \{h_1, \ldots, h_k\}$, no puede ser cierto que $\mathcal{P} \vdash_{\mathcal{D}} e_i\hat{\theta} \rightarrow t_i \Leftarrow \Pi$ es demostrable en $CRWL(\mathcal{D})$ si $Sat_{\mathcal{D}}(\Pi)$. En consecuencia, $Ans_{\mathcal{P}}(G)$ incluye solo respuestas triviales.*

SF *Asumamos que $\Pi \;\square\; \theta \in Ans_{\mathcal{P}}(G)$. Por definición, ha de existir una sustitución $\hat{\theta}$ para la que $\hat{\theta} =_{\backslash evar(G)} \theta$ y $\Pi \models_{\mathcal{D}} S\hat{\theta}$. Sea $\mu \in Sol_{\mathcal{D}}(\Pi)$. Entonces, $\hat{\theta}\mu \in Sol_{\mathcal{D}}(S)$. Puesto que $solve^{\mathcal{D}}(S, \chi) = \blacksquare$, tenemos que $Sol_{\mathcal{D}}(S) = \emptyset$ (debido a los requerimientos de los resolutores de restricciones). De aquí también tenemos que $Sol_{\mathcal{D}}(S\hat{\theta}) = \emptyset$, y entonces $Sol_{\mathcal{D}}(\Pi) = \emptyset$ (esto implica que $Insat_{\mathcal{D}}(\Pi)$). En consecuencia, $Ans_{\mathcal{P}}(G)$ solo incluye respuestas triviales.*

En ambos casos, $Ans_{\mathcal{P}}(G)$ incluye solo respuestas triviales, y entonces $\Pi \;\square\; \theta \in Ans_{\mathcal{P}}(G)$ si $Insat_{\mathcal{D}}(\Pi)$. Probamos que $Sol_{\mathcal{P}}(G) = \emptyset$: sabemos que $\mu \in Sol_{\mathcal{P}}(G) \Leftrightarrow (\emptyset \;\square\; \mu) \in Ans_{\mathcal{P}}(G)$. Obviamente, $Sol_{\mathcal{D}}(\emptyset) = Val_{\mathcal{D}}$ y tenemos que $Sat_{\mathcal{D}}(\emptyset)$. De este modo, $(\emptyset \;\square\; \mu) \notin Ans_{\mathcal{P}}(G)$ debido a que es una respuesta no trivial y entonces $\mu \notin Sol_{\mathcal{P}}(G)$. Por tanto, $Sol_{\mathcal{P}}(G) = \emptyset$.

(3) Procedemos de nuevo regla por regla.

SS *Asumamos que* $\Pi \sqcap \theta \in Ans_{\mathcal{P}}(G')$. *Existe* $\hat{\theta} =_{\setminus evar(G')} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\sigma_0\hat{\theta}$, $Y\hat{\theta} \equiv s\hat{\theta}$ *para cada* $\{Y \mapsto s\} \in \sigma$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (P \sqcap C)\sigma_0\hat{\theta} \Leftarrow \Pi$. *Definimos* $\hat{\theta}'(X) = t\hat{\theta}$ *y* $\hat{\theta}'(Y) = \hat{\theta}(Y)$ *para* $Y \neq X$. *Se cumple que* $\sigma_0\hat{\theta} = \hat{\theta}'$. *Entonces,* $\Pi \models_{\mathcal{D}} S\hat{\theta}'$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (P \sqcap C)\hat{\theta}' \Leftarrow \Pi$. *Como* $X \in pvar(P)$, *tenemos que* $X \notin var(\sigma)$ *por la condición de admisibilidad* **SL**, *y* $Y\hat{\theta}' \equiv Y\hat{\theta} \equiv s\hat{\theta} \equiv s\hat{\theta}'$ *para cada* $\{Y \mapsto s\} \in \sigma$. *Más aún, como* $X \notin var(t)$, $X\hat{\theta}' \equiv t\theta \equiv t\hat{\theta}'$. *Entonces,* $\Pi \models_{\mathcal{D}} t\hat{\theta}' \sqsupseteq X\hat{\theta}'$, *y usando la* Propiedad de Aproximación, *también tenemos que* $\mathcal{P} \vdash_{\mathcal{D}} (t \to X)\hat{\theta}' \Leftarrow \Pi$. *Finalmente, puesto que* $\hat{\theta} =_{\setminus\{X\}} \hat{\theta}'$ *concluimos que* $\Pi \sqcap \theta \in Ans_{\mathcal{P}}(G)$.

IM *Supongamos que* $\Pi \sqcap \theta \in Ans_{\mathcal{P}}(G')$. *Entonces, ha de existir* $\hat{\theta} =_{\setminus evar(G')} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\sigma_0\hat{\theta}$, $Y\hat{\theta} \equiv s\hat{\theta}$ *para cada* $\{Y \mapsto s\} \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \sqcap C)\sigma_0\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{T}_i \equiv \mathcal{P} \vdash_{\mathcal{D}} (e_i \to X_i)\sigma_0\hat{\theta} \Leftarrow \Pi$ *para cada* $1 \leq i \leq m$. *Como* $X \notin var(e_i)$ *por la condición de admisibilidad* **NC** *y* $\overline{X}_m$ *son nuevas variables en* G', *también tenemos que* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} (e_i \to X_i)\hat{\theta} \Leftarrow \Pi$. *Definimos ahora* $\hat{\theta}'(X) = h\overline{X_m\hat{\theta}}$, $\hat{\theta}'(X_i) = X_i$ *para todo* $1 \leq i \leq m$ *y* $\hat{\theta}'(Y) = \hat{\theta}(Y)$ *para todo* $Y \notin \{X, \overline{X}_m\}$. *Se satisface que* $\sigma_0\hat{\theta} =_{\setminus\{\overline{X}_m\}} \hat{\theta}'$. *Puesto que* $\overline{X}_m$ *son variables nuevas en* G', $\Pi \models_{\mathcal{D}} S\hat{\theta}'$, $\mathcal{P} \vdash_{\mathcal{D}} (P \sqcap C)\hat{\theta}' \Leftarrow \Pi$ *y* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i\hat{\theta}' \to X_i\hat{\theta} \Leftarrow \Pi$. *Además, como* X *es una variable producida en* G, $X \notin var(\sigma)$ *por la condición de admisibilidad* **SL**, *y entonces* $Y\hat{\theta}' \equiv Y\hat{\theta} \equiv s\hat{\theta} \equiv s\hat{\theta}'$ *para cada* $\{Y \mapsto s\} \in \sigma$. *Finalmente,* $\mathcal{T} \equiv$ **DC** $(h\overline{e_m\hat{\theta}'} \to h\overline{X_m\hat{\theta}}) \Leftarrow \Pi$, $[\mathcal{T}_1, \ldots, \mathcal{T}_m])$ *es un árbol de prueba para* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (h\overline{e}_m \to X)\hat{\theta}' \Leftarrow \Pi$. *Por tanto, puesto que* $\hat{\theta} =_{\setminus\{\overline{X}_m\}} \hat{\theta}'$, *concluimos que* $\Pi \sqcap \theta \in Ans_{\mathcal{P}}(G)$.

EL *Asumimos que* $\Pi \sqcap \theta \in Ans_{\mathcal{P}}(G')$. *Existe* $\hat{\theta} =_{\setminus evar(G')} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $Y\hat{\theta} \equiv s\hat{\theta}$ *para cada* $\{Y \mapsto s\} \in \sigma$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (P \sqcap C)\hat{\theta} \Leftarrow \Pi$. *Definimos* $\hat{\theta}'(X) = \bot$ *y* $\hat{\theta}'(Y) = \hat{\theta}(Y)$ *para todo* $Y \neq X$ *(observamos que* $\hat{\theta}'$ *está bien definido, porque* X *es una variable producida pero no demandada en* G*). Se satisface que* $\hat{\theta} =_{\setminus\{X\}} \hat{\theta}'$. *Puesto que* $X \notin var(P \sqcap C \sqcap S \sqcap \sigma)$, *directamente se tiene que* $\Pi \models_{\mathcal{D}} S\hat{\theta}'$, $Y\hat{\theta}' \equiv Y\hat{\theta} \equiv s\hat{\theta} \equiv s\hat{\theta}'$ *para cada* $\{Y \mapsto s\} \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \sqcap C)\hat{\theta}' \Leftarrow \Pi$. *Es más,* $\mathcal{T} \equiv$ **TI** $(e\hat{\theta}' \to \bot \Leftarrow \Pi, [\,])$ *es un árbol de prueba para* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (e \to X)\hat{\theta}' \Leftarrow \Pi$. *Por tanto, como* $\hat{\theta} =_{\setminus\{X\}} \hat{\theta}'$, *concluimos que* $\Pi \sqcap \theta \in Ans_{\mathcal{P}}(G)$.

PF *Supongamos que* $\Pi \sqcap \theta \in Ans_{\mathcal{P}}(G')$. *Ha de existir* $\hat{\theta} =_{\setminus evar(G')} \theta$ *para el que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $\Pi \models_{\mathcal{D}} (p\bar{t}_n \to! X)\hat{\theta}$, $Y\hat{\theta} \equiv s\hat{\theta}$ *para cada* $\{Y \mapsto s\} \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \sqcap C)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (e_j \to X_j)\hat{\theta} \Leftarrow \Pi$ *para cada* $1 \leq j \leq q$ *(si* $q = 0$ *estas pruebas son omitidas). Definimos* $\hat{\theta}'(X_j) = X_j$ *para cada* $1 \leq j \leq q$ *y* $\hat{\theta}'(Y) = \hat{\theta}(Y)$ *para cada* $Y \notin \{X_1, \ldots, X_q\}$. *Se cumple que* $\hat{\theta}' =_{\setminus\{\overline{X}_q\}} \hat{\theta}$. *Ya que* $\overline{X}_q$ *son variables nuevas,* $\Pi \models_{\mathcal{D}} S\hat{\theta}'$, $Y\hat{\theta}' \equiv Y\hat{\theta} \equiv s\hat{\theta} \equiv s\hat{\theta}'$ *para cada* $\{Y \mapsto s\} \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \sqcap C)\hat{\theta}' \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}} e_j\hat{\theta}' \to X_j\hat{\theta} \Leftarrow \Pi$ *para cada* $1 \leq j \leq q$. *Puesto que* $e_j \notin Pat_{\mathcal{D}}$ *para cualquier* $1 \leq j \leq q$, *tenemos que* $\mathcal{P} \vdash_{\mathcal{D}} e_j\hat{\theta}' \to t_j\hat{\theta} \Leftarrow \Pi$ *para cada* $1 \leq j \leq q$. *Es más, para cada* $e_i \in Pat_{\mathcal{D}}$ *sabemos que* $e_i \equiv t_i$, *y trivialmente* $\Pi \models_{\mathcal{D}} e_i\hat{\theta}' \sqsupseteq t_i\hat{\theta}$. *Por la*

Propiedad de Aproximación, *podemos obtener árboles de prueba* $\mathcal{P} \vdash_{\mathcal{D}} e_i\hat{\theta}' \rightarrow t_i\hat{\theta} \Leftarrow \Pi$. *En consecuencia, tenemos árboles de prueba* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i\hat{\theta}' \rightarrow t_i\hat{\theta} \Leftarrow \Pi$ *para cada* $1 \leq i \leq n$. *Entonces, podemos construir un árbol de prueba* $\mathcal{T} \equiv \mathbf{PF}\ (p\overline{e}_n\hat{\theta}' \rightarrow X\hat{\theta} \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_n])$ *(*$X\hat{\theta} \neq \perp$ *ya que* $X \in dvar_{\mathcal{D}}(G')$ *y* $\Pi \sqsubset \theta \in Ans_{\mathcal{P}}(G')$ *con* $\hat{\theta} =_{\backslash evar(G)} \theta$*, y por el* Lema de Demanda $\hat{\theta}(X) \neq \perp$*). Finalmente, como* $\overline{X}_q$ *son variables nuevas y distintas de* X, $\hat{\theta}(X) \equiv \hat{\theta}'(X)$, *y tenemos que* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (p\overline{e}_n \rightarrow X)\hat{\theta}' \Leftarrow \Pi$. *Por tanto, como* $\hat{\theta} =_{\backslash\{\overline{X}_q\}} \hat{\theta}'$, *concluimos que* $\Pi \sqsubset \theta \in Ans_{\mathcal{P}}(G)$.

$\mathbf{DT_1}$ *Asumamos que* $\Pi \sqsubset \theta \in Ans_{\mathcal{P}}(G')$. *Existe* $\hat{\theta} =_{\backslash evar(G')} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $Y\hat{\theta} \equiv s\hat{\theta}$ *para cada* $\{Y \mapsto s\} \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \sqsubset C)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (f\ \overline{e}_n \rightarrow X)\hat{\theta} \Leftarrow \Pi$ *(el árbol definicional* $\mathcal{T}_{f\overline{X}_n}$ *se usa solo para controlar la computación). Entonces, podemos concluir directamente que* $\Pi \sqsubset \theta \in Ans_{\mathcal{P}}(G)$.

$\mathbf{DT_2}$ *Supongamos que* $\Pi \sqsubset \theta \in Ans_{\mathcal{P}}(G')$. *Existe entonces* $\hat{\theta} =_{\backslash evar(G')} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $Y\hat{\theta} \equiv s\hat{\theta}$ *para cada* $\{Y \mapsto s\} \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \sqsubset C)\hat{\theta} \Leftarrow \Pi$, $\mathcal{P} \vdash_{\mathcal{D}} (X'\ \overline{a}_k \rightarrow X)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (f\ \overline{e}_n \rightarrow X')\hat{\theta} \Leftarrow \Pi$ *(el árbol definicional* $\mathcal{T}_{f\overline{X}_n}$ *es utilizado solo para controlar la computación). Por el* Lema de Demanda *tenemos que* $\hat{\theta}(X') \neq \perp$, *ya que* X' *es una variable demandada (debido a las condiciones de la regla, sabemos que* $X \in dvar_{\mathcal{D}}(G)$, *y en consecuencia* $X \in dvar_{\mathcal{D}}(G')$ *puesto que* P *y* S *en* G *no son modificados, y a que, por definición de variable demandada,* X' *es también demandada en* G' *de acuerdo con la condición de admisibilidad* **DT** *para* G'*). Por tanto, tenemos que* $\mathcal{T}' \equiv \mathbf{DF}_{\mathcal{P}}\ ((f\overline{e}_n \rightarrow X')\hat{\theta} \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_n, \mathcal{T}_c, \mathcal{T}_r])$ *es un árbol de prueba para* $\mathcal{T}' : \mathcal{P} \vdash_{\mathcal{D}} (f\ \overline{e}_n \rightarrow X')\hat{\theta} \Leftarrow \Pi$, *usando* $(f\ \overline{t}_n \rightarrow r \Leftarrow P' \sqsubset C') \in [\mathcal{P}]_{\perp}$, *y donde* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i\hat{\theta} \rightarrow t_i \Leftarrow \Pi$ $(1 \leq i \leq n)$, $\mathcal{T}_c : \mathcal{P} \vdash_{\mathcal{D}} P' \sqsubset C' \Leftarrow \Pi$ *y* $\mathcal{T}_r : \mathcal{P} \vdash_{\mathcal{D}} r \rightarrow \hat{\theta}(X') \Leftarrow \Pi$. *Ahora, puesto que también tenemos* $\mathcal{T}_s : \mathcal{P} \vdash_{\mathcal{D}} \hat{\theta}(X')\ \overline{a_k\hat{\theta}} \rightarrow \hat{\theta}(X) \Leftarrow \Pi$ *y* $\hat{\theta}(X') \in Pat_{\mathcal{D}}$, *podemos construir el árbol de prueba* $\mathcal{T} \equiv \mathbf{DF}_{\mathcal{P}}\ ((f\ \overline{e}_n\ \overline{a}_k \rightarrow X)\hat{\theta} \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_n, \mathcal{T}_c, \mathcal{T}_r, \mathcal{T}_s\ ])$ $(k > 0)$ *para* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (f\ \overline{e}_n\ \overline{a}_k \rightarrow X)\hat{\theta} \Leftarrow \Pi$ $(k > 0)$. *Por tanto, podemos concluir que* $\Pi \sqsubset \theta \in Ans_{\mathcal{P}}(G)$.

FV *Asumamos que* $\Pi \sqsubset \theta \in Ans_{\mathcal{P}}(G')$. *Existe* $\hat{\theta} =_{\backslash evar(G')} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\sigma_0\hat{\theta}$, $F\hat{\theta} \equiv h\overline{X_m\hat{\theta}}$ *para* $\{F \mapsto h\overline{X}_m\} \in \sigma_0$, $Y\hat{\theta} \equiv s\hat{\theta}$ *para cada* $\{Y \mapsto s\} \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \sqsubset C)\sigma_0\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (h\ \overline{X}_m\ \overline{a}_k \rightarrow X)\sigma_0\hat{\theta} \Leftarrow \Pi$. *Definimos* $\hat{\theta}'(X_i) = X_i$ *para cada* $1 \leq i \leq m$ *y* $\hat{\theta}'(Y) = \hat{\theta}(Y)$ *para todo* $Y \notin \{\overline{X}_m\}$. *Se cumple que* $\sigma_0\hat{\theta} =_{\backslash\{\overline{X}_m\}} \hat{\theta}'$. *Como* $\overline{X}_m$ *son variables nuevas,* $\Pi \models_{\mathcal{D}} S\hat{\theta}'$, $Y\hat{\theta}' \equiv Y\hat{\theta} \equiv s\hat{\theta} \equiv s\hat{\theta}'$ *para cada* $\{Y \mapsto s\} \in \sigma$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (P \sqsubset C)\hat{\theta}' \Leftarrow \Pi$. *Más aún, ya que* $F \notin \{\overline{X}_m\}$ *y* $\hat{\theta}'(F) = h\overline{X_m\hat{\theta}}$, *tenemos que* $\mathcal{P} \vdash_{\mathcal{D}} (F\ \overline{a}_k \rightarrow X)\hat{\theta}' \Leftarrow \Pi$. *Por tanto, como* $\hat{\theta} =_{\backslash\{F,\overline{X}_m\}} \hat{\theta}'$, *concluimos que* $\Pi \sqsubset \theta \in Ans_{\mathcal{P}}(G)$.

CSS *Asumamos que* $\Pi \sqsubset \theta \in Ans_{\mathcal{P}}(G')$. *Entonces, ha de existir* $\hat{\theta} =_{\backslash evar(G')} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $Y\hat{\theta} \equiv s\hat{\theta}$ *para cada* $\{Y \mapsto s\} \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \sqsubset C)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}}$

$(e \to R)\hat{\theta} \Leftarrow \Pi$ *(el árbol definicional es utilizado sólo para controlar la computación). Tenemos directamente entonces que* $\Pi \ \square \ \theta \in Ans_{\mathcal{P}}(G)$.

DI *Asumamos que* $\Pi \ \square \ \theta \in Ans_{\mathcal{P}}(G')$. *Ha de existir* $\hat{\theta} =_{\backslash evar(G')} \theta$ *para la que* $\Pi \models_{\mathcal{D}} S\sigma_0\hat{\theta}$, $Y\hat{\theta} \equiv (h_i\overline{Y}_{m_i})\hat{\theta}$ *para* $\{Y \mapsto h_i\overline{Y}_{m_i}\} \in \sigma_0$, $Z\hat{\theta} \equiv s\hat{\theta}$ *para cada* $\{Z \mapsto s\} \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \ \square \ C)\sigma_0\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (e \to R)\sigma_0\hat{\theta} \Leftarrow \Pi$ *(el árbol definicional se usa sólo para controlar la computación). Debido a la condición de admisibilidad* **NC**, $Y \neq R$. *Puesto que* $dom(\sigma_0) = \{Y\}$, $R\sigma_0\hat{\theta} = R\hat{\theta}$. *Más aún, como* $e|_{pos(X,\tau)} = Y$, $(e|_{pos(X,\tau)})\sigma_0\hat{\theta} = (h_i\overline{Y}_{m_i})\hat{\theta} = Y\hat{\theta}$. *Es más, para todo* $Z \notin \{R, Y\}$ *se cumple que* $Z\sigma_0\hat{\theta} = Z\hat{\theta}$. *Por tanto,* $\sigma_0\hat{\theta} = \hat{\theta}$. *Entonces,* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $Z\hat{\theta} \equiv s\hat{\theta}$ *para cada* $\{Z \mapsto s\} \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \ \square \ C)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (e \to R)\hat{\theta} \Leftarrow \Pi$. *En consecuencia, ya que* $evar(G) \subseteq evar(G')$ *y* $\overline{Y}_{m_i}$ *son variables nuevas que no aparecen en* G, $\theta =_{\backslash evar(G)} \hat{\theta}$, *y finalmente tenemos que* $\Pi \ \square \ \theta \in Ans_{\mathcal{P}}(G)$.

DN *Asumamos que* $\Pi \ \square \ \theta \in Ans_{\mathcal{P}}(G')$. *Existe entonces* $\hat{\theta} =_{\backslash evar(G')} \theta$ *para la que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $Y\hat{\theta} \equiv s\hat{\theta}$ *para cada* $\{Y \mapsto s\} \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \ \square \ C)\hat{\theta} \Leftarrow \Pi$, $\mathcal{P} \vdash_{\mathcal{D}} (e|_{pos(X,\tau)} \to R')\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (e[R']_{pos(X,\tau)} \to R\ )\hat{\theta} \Leftarrow \Pi$. *Como* $\hat{\theta} \in Sub_{\mathcal{D}}$, *también tenemos que* $\mathcal{P} \vdash_{\mathcal{D}} e\hat{\theta}|_{pos(X,\tau)} \to \hat{\theta}(R') \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}} e\hat{\theta}[\hat{\theta}(R')]_{pos(X,\tau)} \to \hat{\theta}(R) \Leftarrow \Pi$ *son demostrables en CRWL*$(\mathcal{D})$, *donde* $\hat{\theta}(R') \in Pat_{\mathcal{D}}$. *Por la* Propiedad de Particionado, $\mathcal{P} \vdash_{\mathcal{D}} e\hat{\theta} \to \hat{\theta}(R) \Leftarrow \Pi$, *y entonces* $\mathcal{P} \vdash_{\mathcal{D}} (e \to R)\hat{\theta} \Leftarrow \Pi$. *Así pues, como* $evar(G) \subseteq evar(G')$ *y* R' *es una nueva variable,* $\hat{\theta} =_{\backslash evar(G)} \theta$, *y finalmente tenemos que* $\Pi \ \square \ \theta \in Ans_{\mathcal{P}}((G)$.

RRA *Asumamos ahora que* $\Pi \ \square \ \theta \in Ans_{\mathcal{P}}(G')$. *Ha de existir* $\hat{\theta} =_{\backslash evar(G')} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $Y\hat{\theta} \equiv s\hat{\theta}$ *para cada* $\{Y \mapsto s\} \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \ \square \ C)\hat{\theta} \Leftarrow \Pi$, $\mathcal{T}_{c_i} : \mathcal{P} \vdash_{\mathcal{D}} (P_i \ \square \ C_i)\sigma_c\hat{\theta} \Leftarrow \Pi$, $\mathcal{T}_{r_i} : \mathcal{P} \vdash_{\mathcal{D}} (r_i\sigma_c \to R)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (\sigma_f(R_j) \to R_j)\hat{\theta} \Leftarrow \Pi$ *para cada* $1 \leq j \leq m$. *Por el* Lema de Demanda, *tenemos que* $\hat{\theta}(R) \neq \bot$ *(*R *es una variable demandada en* G *y* G'*). Asumamos que* τ *es de la forma* $f\overline{t}_n$, *y entonces* $e = \tau\sigma = f\overline{t_n\sigma}$ *y* $e\hat{\theta} = f\overline{t_n\sigma\hat{\theta}}$. *Por tanto, una demostración de* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (e \to R)\hat{\theta} \Leftarrow \Pi$ *usa la CRWL*$(\mathcal{D})$*-regla* $\mathbf{DF}_{\mathcal{P}}$ *con una instanciación parcial mediante* $\sigma_c\hat{\theta}$ *de la regla de programa* $(f\overline{t}_n \to r_i \Leftarrow P_i \ \square \ C_i) \in \mathcal{P}$ *(para* $1 \leq i \leq k$*) tal que* $\mathcal{T} \equiv \mathbf{DF}_{\mathcal{P}}(f\overline{t_n\sigma\hat{\theta}} \to \hat{\theta}(R) \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_n, \mathcal{T}_{c_i}, \mathcal{T}_{r_i}])$, *donde* $\mathcal{T}_j : \mathcal{P} \vdash_{\mathcal{D}} (t_j\sigma \to t_j\sigma_c)\hat{\theta} \Leftarrow \Pi$ *para todo* $1 \leq j \leq n$. *Por otra parte, para cada variable* $X \in var(t_j)$, $\mathcal{P} \vdash_{\mathcal{D}} \hat{\theta}(\sigma(X)) \to \hat{\theta}(\sigma_c(X)) \Leftarrow \Pi$ *es demostrable en CRWL*$(\mathcal{D})$*:*

- *Si* $X \notin dom_f(\sigma)$, *se tiene que* $\mathcal{P} \vdash_{\mathcal{D}} \hat{\theta}(\sigma(X)) \to \hat{\theta}(\sigma_c(X)) \Leftarrow \Pi$ *porque* $\sigma(X) = \sigma_c(X)$ *y* $\Pi \models_{\mathcal{D}} \hat{\theta}(\sigma_c(X)) \sqsupseteq \hat{\theta}(\sigma_c(X))$ *con* $\hat{\theta}(\sigma_c(X)) \in Pat_{\mathcal{D}}$.
- *Si* $X \in dom_f(\sigma) = \{R_1, \ldots, R_m\}$, $\hat{\theta}(\sigma(X)) = \hat{\theta}(\sigma_f(X))$ *y* $\mathcal{P} \vdash_{\mathcal{D}} \hat{\theta}(\sigma_f(X)) \to \hat{\theta}(X)\ (= \hat{\theta}(\sigma_c(X))) \Leftarrow \Pi$ *es demostrable en CRWL*$(\mathcal{D})$ *por hipótesis inicial.*

Como $evar(G) \subseteq evar(G')$ *y* $\overline{X}$ *son variables nuevas,* $\hat{\theta} =_{\setminus evar(G)} \theta$*, y entonces* $\Pi \square \theta \in Ans_{\mathcal{P}}(G)$*.*

CS *Asumamos que* $\Pi_i \square \theta_i \in Ans_{\mathcal{P}}(G_i)$ $(1 \leq i \leq k)$*. Ha de existir* $\hat{\theta}_i =_{\setminus evar(G_i)} \theta_i$ *para la que* $\Pi_i \models_{\mathcal{D}} S_i\hat{\theta}_i$*,* $X\hat{\theta}_i \equiv t\hat{\theta}_i$ *para cada* $\{X \mapsto t\} \in \sigma_i$*,* $Y\hat{\theta}_i \equiv s\hat{\theta}_i$ *para cada* $\{Y \mapsto s\} \in \sigma$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (P \square C)\sigma_i\hat{\theta}_i \Leftarrow \Pi_i$*. Definimos* $\hat{\theta'}_i(Y) = Y$ *para todo* $Y \in \overline{Y}_i \setminus dom(\sigma_i)$ *y* $\hat{\theta'}_i(Z) = \hat{\theta}_i(Z)$ *para* $Z \in (\setminus\{\overline{Y}_i\}) \cup dom(\sigma_i)$*. Como* $X\hat{\theta}_i \equiv t\hat{\theta}_i$ *para cada* $\{X \mapsto t\} \in \sigma_i$*, se cumple que* $\sigma_i\hat{\theta}_i =_{(\setminus\{\overline{Y}_i\}) \cup dom(\sigma_i)} \hat{\theta'}_i$*. Puesto que* $\overline{Y}_i$ *son variable nuevas,* $Y\hat{\theta'}_i \equiv Y\hat{\theta}_i \equiv s\hat{\theta}_i \equiv s\hat{\theta'}_i$ *para cada* $\{Y \to s\} \in \sigma$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (P \square C)\hat{\theta'}_i \Leftarrow \Pi_i$*. Probamos ahora que* $\Pi_i \models_{\mathcal{D}} S\hat{\theta'}_i$*. Sea* $\mu \in Sol_{\mathcal{D}}(\Pi_i)$*. Debido a que* $\Pi_i \models_{\mathcal{D}} S_i\hat{\theta}_i$*, se tiene que* $\mu \in Sol_{\mathcal{D}}(S_i\hat{\theta}_i)$*. De aquí,* $\hat{\theta}_i\mu \in Sol_{\mathcal{D}}(S_i)$*. Más aún, como* $X\hat{\theta}_i \equiv t\hat{\theta}_i$ *para cada* $\{X \mapsto t\} \in \sigma_i$ *tenemos también que* $X\hat{\theta}_i\mu \equiv t\hat{\theta}_i\mu$ *para cada* $\{X \mapsto t\} \in \sigma_i$*. Por tanto, ha de existir* $\hat{\theta'}_i\mu =_{\setminus S} \hat{\theta}_i\mu$ *para la que* $\hat{\theta}_i\mu \in Sol_{\mathcal{D}}(S_i)$ *y* $X\hat{\theta}_i\mu \equiv t\hat{\theta}_i\mu$ *para cada* $\{X \mapsto t\} \in \sigma_i$*. Por definición,* $\hat{\theta}_i\mu \in Sol_{\mathcal{D}}(\exists_{\setminus S}.\ S_i \square \sigma_i)$ *(*$1 \leq i \leq k$*) y entonces* $\hat{\theta}_i\mu \in \bigcup_{i=1}^{k} Sol_{\mathcal{D}}(\exists_{\setminus S}.\ S_i \square \sigma_i)$*. Usando los requerimientos de un resolutor de restricciones, se sigue también que* $\hat{\theta}_i\mu \in Sol_{\mathcal{D}}(S)$*. Entonces,* $\mu \in Sol_{\mathcal{D}}(S\hat{\theta}_i)$*. Por tanto, concluimos que* $\Pi_i \square \theta_i \in Ans_{\mathcal{P}}(G)$ *para cada* $1 \leq i \leq k$*.*

AC *Supongamos que* $\Pi \square \theta \in Ans_{\mathcal{P}}(G')$*. Ha de existir* $\hat{\theta} =_{\setminus evar(G')} \theta$ *para la que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$*,* $\Pi \models_{\mathcal{D}} (p\bar{t}_n \to! t)\hat{\theta}$*,* $X\hat{\theta} \equiv s\hat{\theta}$ *para cada* $\{X \mapsto s\} \in \sigma$*,* $\mathcal{P} \vdash_{\mathcal{D}} (P \square C)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (e_j \to X_j)\hat{\theta} \Leftarrow \Pi$ *para cada* $1 \leq j \leq q$ *(si* $q = 0$ *estas pruebas se omiten). Definimos* $\hat{\theta'}(X_j) = X_j$ *para cada* $1 \leq j \leq q$ *y* $\hat{\theta'}(Y) = \hat{\theta}(Y)$ *para cada* $Y \notin \{X_1, \ldots, X_q\}$*. Se cumple que* $\hat{\theta'} =_{\setminus\{\overline{X}_q\}} \hat{\theta}$*. Como* $\overline{X}_q$ *son variable nuevas,* $\Pi \models_{\mathcal{D}} S\hat{\theta'}$*,* $X\hat{\theta'} \equiv X\hat{\theta} \equiv s\hat{\theta} \equiv s\hat{\theta'}$ *para cada* $\{X \mapsto s\} \in \sigma$*,* $\mathcal{P} \vdash_{\mathcal{D}} (P \square C)\hat{\theta'} \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}} e_j\hat{\theta'} \to X_j\hat{\theta} \Leftarrow \Pi$ *para cada* $1 \leq j \leq q$*. Debido a que* $e_j \notin Pat_{\mathcal{D}}$ *para todo* $1 \leq j \leq q$*, tenemos que* $\mathcal{P} \vdash_{\mathcal{D}} e_j\hat{\theta'} \to t_j\hat{\theta} \Leftarrow \Pi$ *para cada* $1 \leq j \leq q$*. Es más, para cada* $e_i \in Pat_{\mathcal{D}}$ *sabemos que* $e_i \equiv t_i$*, y trivialmente* $\Pi \models_{\mathcal{D}} e_i\hat{\theta'} \sqsupseteq t_i\hat{\theta}$*. Por la* Propiedad de Aproximación *obtenemos árboles de prueba* $\mathcal{P} \vdash_{\mathcal{D}} e_i\hat{\theta'} \to t_i\hat{\theta} \Leftarrow \Pi$*. Por tanto, tenemos árboles de prueba* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i\hat{\theta'} \to t_i\hat{\theta} \Leftarrow \Pi$ *para cada* $1 \leq i \leq n$*. Como* $\Pi \models_{\mathcal{D}} (p\bar{t}_n \to! t)\hat{\theta}$*, podemos construir un árbol de prueba* $\mathcal{T} \equiv \mathbf{AC}\ (p\bar{e}_n\hat{\theta'} \to t\hat{\theta} \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_n])$ *(*$t\hat{\theta} \neq \bot$ *debido a que* $t \notin \mathcal{V}ar$ *o bien* $t \in dvar_{\mathcal{D}}(G')$ *y podemos aplicar el* Lema de Demanda*). Finalmente, ya que* $\overline{X}_q$ *son variables nuevas,* $t\hat{\theta} \equiv t\hat{\theta'}$*, y se tiene entonces que* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (p\bar{e}_n \to! t)\hat{\theta'} \Leftarrow \Pi$*. Por tanto, como* $\hat{\theta} =_{\setminus\{\overline{X}_q\}} \hat{\theta'}$*, concluimos que* $\Pi \square \theta \in Ans_{\mathcal{P}}(G)$*.*

□

A.3.5. Lema de progreso del cálculo $CDNC(\mathcal{D})$

Demostramos ahora las principales propiedades del cálculo de resolución de objetivos $CDNC(\mathcal{D})$ con respecto a su completitud bajo el supuesto de que el resolutor utilizado sea idealmente completo.

Demostración 39 (Demostración del apartado (2) del Lema 15)

(2) En cada uno de los siguientes casos, G' y G_i son los objetivos que se obtienen mediante la aplicación de la correspondiente transformación del cálculo $CDNC(\mathcal{D})$.

SS *Sea* $\Pi \ \square \ \theta \in Ans_{\mathcal{P}}(G)$. *Entonces existe* $\hat{\theta} =_{\backslash evar(G)} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $Z\hat{\theta} \equiv s\hat{\theta}$ *para cada* $\{Z \mapsto s\} \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \ \square \ C)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (t \rightarrow X)\hat{\theta} \Leftarrow \Pi$. *Por la* Propiedad de Aproximación, $\Pi \models_{\mathcal{D}} t\hat{\theta} \sqsupseteq X\hat{\theta}$. *Si consideramos* $\hat{\theta}' = \sigma_0\hat{\theta}$, *se cumple que* $\hat{\theta}' =_{\backslash\{X\}} \hat{\theta}$, $\hat{\theta}' \sqsupseteq \hat{\theta}$ *y* $\sigma_0\hat{\theta}' = \hat{\theta}'$. *Entonces, es posible "elevar"* $\hat{\theta}$ *para obtener* $\hat{\theta}' \sqsupseteq \hat{\theta}$ *tal que* $\hat{\theta}' =_{\backslash\{X\}} \hat{\theta}$ *y* $\sigma_0\hat{\theta}' \sqsupseteq \hat{\theta}$. *Se sigue que* $\Pi \models_{\mathcal{D}} S\sigma_0\hat{\theta}'$ *(si* X *aparece en* S, $X\sigma_0\hat{\theta}' \equiv t\hat{\theta}' \equiv t\hat{\theta}$ *debido a que* $X \notin var(t)$, *y sabemos que* $\sigma_0\hat{\theta}' \sqsupseteq \hat{\theta}$*),* $Z\hat{\theta}' \equiv s\hat{\theta}'$ *para cada* $\{Z \mapsto s\} \in \sigma$ *(la variable* X *no aparece en* σ *porque es una variable producida) y* $((P \ \square \ C)\hat{\theta} \Leftarrow \Pi) \succcurlyeq_{\mathcal{D}} ((P \ \square \ C)\sigma_0\hat{\theta}' \Leftarrow \Pi)$. *Usando la* Propiedad de Implicación, *también tenemos que* $\mathcal{P} \vdash_{\mathcal{D}} (P \ \square \ C)\sigma_0\hat{\theta}' \Leftarrow \Pi$. *De aquí, si consideramos* $\hat{\theta}' =_{\backslash evar(G')} \theta'$ *entonces* $\Pi \ \square \ \theta' \in Ans_{\mathcal{P}}(G')$. *Claramente,* $Sol_{\mathcal{D}}(\Pi \ \square \ \theta) \subseteq Sol_{\mathcal{D}}(\exists_{\backslash G}.\ \Pi \ \square \ \theta')$.

IM *Sea* $\Pi \ \square \ \theta \in Ans_{\mathcal{P}}(G)$. *Existe* $\hat{\theta} =_{\backslash evar(G)} \theta$ *para la que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $Z\hat{\theta} \equiv s\hat{\theta}$ *para cada* $\{Z \mapsto s\} \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \ \square \ C)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (h\bar{e}_m \rightarrow X)\hat{\theta} \Leftarrow \Pi$. *Como* $X \in dvar_{\mathcal{D}}(P \ \square \ S)$, *usando el* Lema de Demanda *tenemos que* $\hat{\theta}(X) \neq \bot$. *Más aún, como por hipótesis* $\Pi \ \square \ \theta$ *es una respuesta no trivial para* G, *tenemos que* $Sat_{\mathcal{D}}(\Pi)$. *Por tanto, un árbol de prueba* $\mathcal{T}$ *para* $\mathcal{P} \vdash_{\mathcal{D}} (h\bar{e}_m \rightarrow X)\hat{\theta} \Leftarrow \Pi$ *tiene la forma* $\mathcal{T} \equiv \mathbf{R}\ ((h\bar{e}_m \rightarrow X)\hat{\theta} \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_m])$ *con* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i\hat{\theta} \rightarrow t_i \Leftarrow \Pi$ *para cada* $1 \leq i \leq m$. *Si* $\hat{\theta}(X)$ *es de la forma* $h\bar{t}_m$ *entonces* $\mathbf{R}$ *es la regla* **DC**, *y si* $\hat{\theta}(X)$ *es una variable, entonces* $\Pi \models_{\mathcal{D}} h\bar{t}_m \sqsupseteq \hat{\theta}(X)$ *y* $\mathbf{R}$ *es la regla* **IR**. *Definimos* $\hat{\theta}'(X_i) = t_i$ *para todo* $1 \leq i \leq m$ *y* $\hat{\theta}'(Y) = \hat{\theta}(Y)$ *para todo* $Y \notin \{\overline{X}_m\}$. *Se cumple que* $\sigma_0\hat{\theta}' =_{\backslash\{\overline{X}_m\}} \hat{\theta}$ *si* $\mathbf{R}$ *es* **DC** *y* $\Pi \models_{\mathcal{D}} \sigma_0\hat{\theta}' \sqsupseteq_{\backslash\{\overline{X}_m\}} \hat{\theta}$ *si* $\mathbf{R}$ *es* **IR**. *En ambos casos, como* $\overline{X}_m$ *son nuevas variables y* X *es una variable producida, esto implica que* $\Pi \models_{\mathcal{D}} S\sigma_0\hat{\theta}'$, $Z\hat{\theta}' \equiv s\hat{\theta}'$ *para cada* $\{Z \mapsto s\} \in \sigma$ *y* $(e_i\hat{\theta} \rightarrow t_i \Leftarrow \Pi) \succcurlyeq_{\mathcal{D}} (e_i\sigma_0\hat{\theta}' \rightarrow t_i \Leftarrow \Pi)$ *para cada* $1 \leq i \leq m$. *Usando la* Propiedad de Implicación, $\mathcal{T}'_i : \mathcal{P} \vdash_{\mathcal{D}} e_i\sigma_0\hat{\theta}' \rightarrow t_i \Leftarrow \Pi$ $(1 \leq i \leq m)$. *Como* $\hat{\theta}'(\sigma_0(X_i)) = t_i$, *tenemos que* $\mathcal{T}'_i : \mathcal{P} \vdash_{\mathcal{D}} (e_i \rightarrow X_i)\sigma_0\hat{\theta}' \Leftarrow \Pi$ *para cada* $1 \leq i \leq m$. *Más aún, también tenemos que* $\overline{\mathcal{T}'} : \mathcal{P} \vdash_{\mathcal{D}} (P \ \square \ C)\sigma_0\hat{\theta}' \Leftarrow \Pi$ *aplicando de nuevo la* Propiedad de Implicación. *De aquí, si tomamos* $\theta' =_{\backslash evar(G')} \hat{\theta}'$, *entonces* $\Pi \ \square \ \theta' \in Ans_{\mathcal{P}}(G')$. *Claramente,* $Sol_{\mathcal{D}}(\Pi \ \square \ \theta) \subseteq Sol_{\mathcal{D}}(\exists_{\backslash G}.\ \Pi \ \square \ \theta')$.

EL *Sea* $\Pi \ \square \ \theta \in Ans_{\mathcal{P}}(G)$. *Ha de existir* $\hat{\theta} =_{\backslash evar(G)} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $Z\hat{\theta} \equiv s\hat{\theta}$ *para cada* $\{Z \mapsto s\} \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \ \square \ C)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (e \to X)\hat{\theta} \Leftarrow \Pi$. *Por tanto, tenemos que* $\Pi \ \square \ \theta \in Ans_{\mathcal{P}}(G')$ *y* $Sol_{\mathcal{D}}(\Pi \ \square \ \theta) \subseteq Sol_{\mathcal{D}}(\exists_{\backslash G}.\ \Pi \ \square \ \theta)$ *porque* $X \notin var(P \ \square \ C \ \square \ S \ \square \ \sigma)$.

PF *Sea* $\Pi \ \square \ \theta \in Ans_{\mathcal{P}}(G)$. *Existe* $\hat{\theta} =_{\backslash evar(G)} \theta$ *para la que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $Z\hat{\theta} \equiv s\hat{\theta}$ *para cada* $\{Z \mapsto s\} \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \ \square \ C)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (p\overline{e}_n \to X)\hat{\theta} \Leftarrow \Pi$. *Como* $X \in dvar_{\mathcal{D}}(P \ \square \ S)$, *por el* Lema de Demanda *sabemos que* $\hat{\theta}(X) \neq \bot$. *Adicionalmente, tenemos que* $Sat_{\mathcal{D}}(\Pi)$ *porque* $\Pi \ \square \ \theta$ *es una respuesta no trivial. Más aún,* $\mathcal{T} \equiv \mathbf{PF}\ ((p\overline{e}_n \to X)\hat{\theta} \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_n])$ *con* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i\hat{\theta} \to t'_i \Leftarrow \Pi$ *para cada* $1 \leq i \leq n$ *y* $\Pi \models_{\mathcal{D}} p\overline{t'}_n \to! \hat{\theta}(X)$. *Para cada* $1 \leq i \leq n$, *si* $e_i \in Pat_{\mathcal{D}}$ *entonces* $\Pi \models_{\mathcal{D}} t'_i \sqsubseteq e_i\hat{\theta}$ *por la* Propiedad de Aproximación. *Definimos* $\hat{\theta}'(X_j) = t'_j$ *para cada* $1 \leq j \leq q$ *y* $\hat{\theta}'(Y) = \hat{\theta}(Y)$ *para todo* $Y \notin \{\overline{X}_q\}$ *(si* $q = 0$ *consideramos* $\hat{\theta}' = \hat{\theta}$*). Tenemos árboles de prueba* $\mathcal{T}_j : \mathcal{P} \vdash_{\mathcal{D}} e_j\hat{\theta} \to t'_j \Leftarrow \Pi$ *para cada* $1 \leq j \leq q$. *Como* $\overline{X}_q$ *son variables nuevas, también tenemos que* $\Pi \models_{\mathcal{D}} S\hat{\theta}'$, $Z\hat{\theta}' \equiv s\hat{\theta}'$ *para cada* $\{Z \mapsto s\} \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \ \square \ C)\hat{\theta}' \Leftarrow \Pi$ *y* $\mathcal{T}_j : \mathcal{P} \vdash_{\mathcal{D}} (e_j \to X_j)\hat{\theta}' \Leftarrow \Pi$ *para cada* $1 \leq j \leq q$. *Por otra parte, para cada* $1 \leq i \leq n$, *si* $e_i \in Pat_{\mathcal{D}}$ *entonces sabemos que* $t_i \equiv e_i$ *y que* $\Pi \models_{\mathcal{D}} t'_i \sqsubseteq e_i\hat{\theta}$. *De aquí,* $\Pi \models_{\mathcal{D}} t'_i \sqsubseteq t_i\hat{\theta}'$. *En otro caso,* $t'_j = \hat{\theta}'(X_j)$, $t_j \equiv X_j$ *y tenemos que* $t'_j = t_j\hat{\theta}'$. *Por tanto, como* $\hat{\theta}(X) = \hat{\theta}'(X)$ *y ya que* $\Pi \models_{\mathcal{D}} p\overline{t'}_n \to! \hat{\theta}(X)$, *también tenemos que* $\Pi \models_{\mathcal{D}} (p\overline{t}_n \to! X)\hat{\theta}'$. *Finalmente, si consideramos* $\theta' =_{\backslash evar(G')} \hat{\theta}'$, *entonces* $\Pi \ \square \ \theta' \in Ans_{\mathcal{P}}(G')$. *Claramente,* $Sol_{\mathcal{D}}(\Pi \ \square \ \theta) \subseteq Sol_{\mathcal{D}}(\exists_{\backslash G}.\ \Pi \ \square \ \theta')$.

DT$_1$ *Sea* $\Pi \ \square \ \theta \in Ans_{\mathcal{P}}(G)$. *Ha de existir* $\hat{\theta} =_{\backslash evar(G)} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $Z\hat{\theta} \equiv s\hat{\theta}$ *para cada* $\{Z \mapsto s\} \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \ \square \ C)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (f\overline{e}_n \to X)\hat{\theta} \Leftarrow \Pi$. *Como los árboles definicionales se usan sólo para controlar la computación, tenemos que* $\Pi \ \square \ \theta \in Ans_{\mathcal{P}}(G')$ *y* $Sol_{\mathcal{D}}(\Pi \ \square \ \theta) \subseteq Sol_{\mathcal{D}}(\exists_{\backslash G}.\ \Pi \ \square \ \theta)$.

DT$_2$ *Sea* $\Pi \ \square \ \theta \in Ans_{\mathcal{P}}(G)$. *Ha de existir* $\hat{\theta} =_{\backslash evar(G)} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $Z\hat{\theta} \equiv t\hat{\theta}$ *para cada* $\{Z \mapsto t\} \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \ \square \ C)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (f\ \overline{e}_n\ \overline{a}_k \to X)\hat{\theta} \Leftarrow \Pi$. *Puesto que* $X \in dvar_{\mathcal{D}}(P \ \square \ S)$, *el* Lema de Demanda *asegura que* $\hat{\theta}(X) \neq \bot$. *Más aún, como* $\Pi \ \square \ \theta$ *es una respuesta no trivial de* G, *tenemos que* $Sat_{\mathcal{D}}(\Pi)$. *Por tanto,* $\mathcal{T} \equiv \mathbf{DF}_{\mathcal{P}}\ (f\ \overline{e_n\hat{\theta}}\ \overline{a_k\hat{\theta}} \to \hat{\theta}(X) \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_n, \mathcal{T}_c, \mathcal{T}_r, \mathcal{T}_s])$ *usando* $(f\ \overline{t}_n \to r \Leftarrow P \ \square \ C) \in [\mathcal{P}]_{\bot}$ *y* $s \in Pat_{\mathcal{D}}$, *donde* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i\hat{\theta} \to \overline{t_i} \Leftarrow \Pi$ *(*$1 \leq i \leq n$*),* $\mathcal{T}_c : \mathcal{P} \vdash_{\mathcal{D}} P \ \square \ C \Leftarrow \Pi$, $\mathcal{T}_r : \mathcal{P} \vdash_{\mathcal{D}} r \to s \Leftarrow \Pi$ *y* $\mathcal{T}_s : \mathcal{P} \vdash_{\mathcal{D}} s\overline{a_k\hat{\theta}} \to \hat{\theta}(X) \Leftarrow \Pi$. *Definimos* $\hat{\theta}'(X') = s$ *y* $\hat{\theta}'(Y) = \hat{\theta}(Y)$ *para todo* $Y \neq X'$. *Se sigue que* $\hat{\theta}' =_{\backslash \{X'\}} \hat{\theta}$, *y puesto que* X' *es una nueva variable que no aparece en* G, *tenemos que* $\Pi \models_{\mathcal{D}} S\hat{\theta}'$, $Z\hat{\theta}' \equiv Z\hat{\theta} \equiv t\hat{\theta} \equiv t\hat{\theta}'$ *para cada* $\{Z \mapsto t\} \in \sigma$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (P \ \square \ C)\hat{\theta}' \Leftarrow \Pi$. *Adicionalmente, a partir de la deducción de* $\mathcal{T}_s$ *es posible una derivación* $\mathcal{P} \vdash_{\mathcal{D}} \hat{\theta}'(X')\ \overline{a_k\hat{\theta}'} \to$

$\hat{\theta}'(X) \Leftarrow \Pi$, *y entonces* $\mathcal{P} \vdash_{\mathcal{D}} (X'\ \overline{a}_k \to X)\hat{\theta}' \Leftarrow \Pi$. *Más aún, tenemos que* $\mathcal{T}_i : \mathcal{P} \vdash_{\mathcal{D}} e_i\hat{\theta}' \to t_i \Leftarrow \Pi$ *(*$1 \leq i \leq n$*),* $\mathcal{T}_c : \mathcal{P} \vdash_{\mathcal{D}} P \ \square\ C \Leftarrow \Pi$ *y* $\mathcal{T}_r : \mathcal{P} \vdash_{\mathcal{D}} r \to \hat{\theta}'(X') \Leftarrow \Pi$. *Como* $Sat_{\mathcal{D}}(\Pi)$ *y* $\hat{\theta}'(X') = s \neq \bot$ *(en otro caso, el árbol de prueba* $\mathcal{T}_s$ *no es posible en el cálculo CRWL(*$\mathcal{D}$*) con* $k > 0$*) podemos construir* $\mathcal{T}' \equiv \mathbf{DF}_{\mathcal{P}}\ (f\ \overline{e_n\hat{\theta}'} \to \hat{\theta}'(X') \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_n, \mathcal{T}_c, \mathcal{T}_r])$ *usando* $(f\ \overline{t}_n \to r \Leftarrow P \ \square\ C) \in [\mathcal{P}]_{\bot}$. *En consecuencia, tenemos que* $\mathcal{T}' : \mathcal{P} \vdash_{\mathcal{D}} (f\ \overline{e}_n \to X')\hat{\theta}' \Leftarrow \Pi$. *Finalmente, si tenemos que* $\theta' =_{\setminus evar(G')} \hat{\theta}'$, *entonces* $\Pi \ \square\ \theta' \in Ans_{\mathcal{P}}(G')$ *y* $Sol_{\mathcal{D}}(\Pi \ \square\ \theta) \subseteq Sol_{\mathcal{D}}(\exists_{\setminus G}.\ \Pi \ \square\ \theta')$.

FV *Sea* $\Pi \ \square\ \theta \in Ans_{\mathcal{P}}(G)$. *Ha de existir* $\hat{\theta} =_{\setminus evar(G)} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $Z\hat{\theta} \equiv s\hat{\theta}$ *para cada* $\{Z \mapsto s\} \in \sigma$, $\overline{\mathcal{T}} : \mathcal{P} \vdash_{\mathcal{D}} (P \ \square\ C)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (F\ \overline{a}_k \to X)\hat{\theta} \Leftarrow \Pi$. *Puesto que* $X \in dvar_{\mathcal{D}}(P \ \square\ S)$, *usando el* Lema de Demanda *se tiene que* $\hat{\theta}(X) \neq \bot$. *Más aún,* F *es una variable demandada y* $\hat{\theta}(F) \neq \bot$. *Como* $k > 0$, *sabemos que* $\hat{\theta}(F) \notin \mathcal{B}^{\mathcal{D}} \cup \mathcal{V}ar$ *(en otro caso, el árbol de prueba* $\mathcal{T}$ *no es posible en el cálculo CRWL(*$\mathcal{D}$*)). En consecuencia,* $\hat{\theta}(F)$ *ha de ser de la forma* $h\overline{t}_m \in Pat_{\mathcal{D}}$. *Definimos* $\hat{\theta}'(X_i) = t_i$ *para cada* $1 \leq i \leq m$, $\hat{\theta}'(F) = h\overline{t}_m$ *y* $\hat{\theta}'(Y) = \hat{\theta}(Y)$ *para todo* $Y \notin \{\overline{X}_m, F\}$. *Se satisface que* $\sigma_0\hat{\theta}' = \hat{\theta}'$ *y que* $\sigma_0\hat{\theta}' =_{\setminus\{\overline{X}_m\}} \hat{\theta}$. *Como* $\overline{X}_m$ *son variables nuevas, esto implica que* $\Pi \models_{\mathcal{D}} S\sigma_0\hat{\theta}'$, $Z\hat{\theta}' \equiv s\hat{\theta}'$ *para cada* $\{Z \mapsto s\} \in \sigma$, $\hat{\theta}'(F) \equiv h\overline{X_m\hat{\theta}'}$ *para* $\{F \mapsto h\overline{X}_m\} \in \sigma_0$ *y* $\overline{\mathcal{T}} : \mathcal{P} \vdash_{\mathcal{D}} (P \ \square\ C)\sigma_0\hat{\theta}' \Leftarrow \Pi$. *Más aún, de* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (F\ \overline{a}_k \to X)\hat{\theta} \Leftarrow \Pi$ *también tenemos que* $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (h\ \overline{X}_m\ \overline{a}_k \to X)\sigma_0\hat{\theta}' \Leftarrow \Pi$. *Finalmente, podemos considerar* $\theta' =_{\setminus evar(G')} \hat{\theta}'$, *y entonces* $\Pi \ \square\ \theta' \in Ans_{\mathcal{P}}(G')$. *Claramente,* $Sol_{\mathcal{D}}(\Pi \ \square\ \theta) \subseteq Sol_{\mathcal{D}}(\exists_{\setminus G}.\ \Pi \ \square\ \theta')$.

CSS *Sea* $\Pi \ \square\ \theta \in Ans_{\mathcal{P}}(G)$. *Entonces existe* $\hat{\theta} =_{\setminus evar(G)} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $Z\hat{\theta} \equiv s\hat{\theta}$ *para cada* $\{Z \mapsto s\} \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \ \square\ C)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (e \to R)\hat{\theta} \Leftarrow \Pi$ *(el árbol definicional sólo es usado para controlar la computación). Por tanto, tenemos que* $\Pi \ \square\ \theta \in Ans_{\mathcal{P}}(G')$ *y que* $Sol_{\mathcal{D}}(\Pi \ \square\ \theta) \subseteq Sol_{\mathcal{D}}(\exists_{\setminus G}.\ \Pi \ \square\ \theta)$.

DI *Consideramos* $\Pi \ \square\ \theta \in Ans_{\mathcal{P}}(G)$. *Ha de existir* $\hat{\theta} =_{\setminus evar(G)} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $Z\hat{\theta} \equiv s\hat{\theta}$ *para cada* $\{Z \mapsto s\} \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \ \square\ C)\hat{\theta} \Leftarrow \Pi$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (e \to R)\hat{\theta} \Leftarrow \Pi$ *(de nuevo, el árbol definicional solamente se utiliza para controlar la computación). Usando el* Lema de Demanda, $\hat{\theta}(R) \neq \bot$ *(*$R \in dvar_{\mathcal{D}}(P \ \square\ C)$ *por las condiciones de admisibilidad de G). Debido a la estructura del árbol definicional caso* $(\tau, X, [\mathcal{T}_1, \ldots, \mathcal{T}_k])$, *y puesto que* $e|_{pos(X,\tau)} = Y$, *se sigue que* $\hat{\theta}(Y) = h_i\overline{t}_{m_i} \in Pat_{\mathcal{D}}$, *donde* h_i *(*$1 \leq i \leq k$*) es un símbolo pasivo de aridad* m_i *y* $t_i \in Pat_{\mathcal{D}}$. *Como* $\Pi \ \square\ \theta$ *es una respuesta no trivial de G, se tiene que* $Sat_{\mathcal{D}}(\Pi)$. *Por tanto, la CRWL(*$\mathcal{D}$*)-deducción* $\mathcal{P} \vdash_{\mathcal{D}} (e \to R)\hat{\theta} \Leftarrow \Pi$ *usa en el último paso la regla* $DF_{\mathcal{P}}$ *con una instanciación parcial de una regla de programa en* $\mathcal{P}$ *de la forma* $h_i\overline{s}_{m_i}$ *en la posición* $pos(X, \tau)$. *A partir de aquí, podemos definir* $\hat{\theta}'(Y_j) = t_j$ *para cada* $1 \leq j \leq m_i$ *y* $\hat{\theta}'(X) = \hat{\theta}(X)$ *para cualquier* $X \notin \{\overline{Y}_{m_i}\}$. *Esto implica que* $\sigma_0\hat{\theta}' =_{\setminus\{\overline{Y}_{m_i}\}} \hat{\theta}$. *Debido a que* $\overline{Y}_{m_i}$ *son variables nuevas, tenemos que* $\Pi \models_{\mathcal{D}} S\sigma_0\hat{\theta}'$, $\mathcal{P} \vdash_{\mathcal{D}} (P \ \square\ C)\sigma_0\hat{\theta}' \Leftarrow \Pi$ *y*

$Z\sigma_0\hat{\theta}' \equiv s\sigma_0\hat{\theta}'$ para cada $\{Z \mapsto s\} \in \sigma$. Más aún, como $Y \notin \{\overline{Y}_{m_i}\}$, se tiene que $\hat{\theta}'(Y) = \hat{\theta}(Y) = h_i\bar{t}_{m_i}$ y que $(h\overline{Y}_{m_i})\hat{\theta}' = h_i\overline{Y_{m_i}\hat{\theta}'} = h_i\bar{t}_{m_i}$. Se sigue entonces que $Y\hat{\theta}' = (h\overline{Y}_{m_i})\hat{\theta}'$ para $\{Y \mapsto h\overline{Y}_{m_i}\} \in \sigma_0$. Como $e|_{pos(X,\tau)} = Y$ y $\hat{\theta}'(\sigma_0(Y)) = \hat{\theta}(Y)$ $= h_i\bar{t}_{m_i}$, $\mathcal{P} \vdash_{\mathcal{D}} (e \to R)\sigma_0\hat{\theta}' \Leftarrow \Pi$ es demostrable con la misma deducción que $\mathcal{P}$ $\vdash_{\mathcal{D}} (e \to R)\hat{\theta} \Leftarrow \Pi$. Por tanto, si tomamos $\theta' =_{\setminus evar(G')} \hat{\theta}'$ tenemos que $\Pi \square \theta' \in Ans_{\mathcal{P}}(G')$, y claramente $Sol_{\mathcal{D}}(\Pi \square \theta) \subseteq Sol_{\mathcal{D}}(\exists_{\setminus G}.\ \Pi \square \theta')$.

DN *Sea $\Pi \square \theta \in Ans_{\mathcal{P}}(G)$. Existe $\hat{\theta} =_{\setminus evar(G)} \theta$ para la que $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $Z\hat{\theta} \equiv s\hat{\theta}$ para cada $\{Z \mapsto s\} \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \square C)\hat{\theta} \Leftarrow \Pi$ y $\mathcal{P} \vdash_{\mathcal{D}} (e \to R)\hat{\theta} \Leftarrow \Pi$ (el árbol definicional solamente es usado para controlar la computación). Usando el* Lema de Demanda*, $\hat{\theta}(R) \neq \bot$ ($R \in dvar_{\mathcal{D}}(P \square C)$ por las condiciones de admisibilidad de G). Puesto que $pos(X,\tau) \in Pos(e)$, también tenemos que $pos(X,\tau) \in Pos(e\hat{\theta})$. Adicionalmente, se cumplen las siguientes propiedades:*

- *$e\hat{\theta}$ y $e\hat{\theta}|_{pos(X,\tau)}$ tienen el mismo símbolo de función en la raíz que e y $e|_{pos(X,\tau)}$, respectivamente.*
- *Como $\tau \preceq e$ y $\tau \preceq \tau\hat{\theta}$, entonces $\tau\hat{\theta} \preceq e\hat{\theta}$ y $\tau \preceq e\hat{\theta}$ con τ en la raíz del árbol definicional caso $(\tau, X, [\mathcal{T}_1, \ldots, \mathcal{T}_k])$.*

Usando la Propiedad de Particionado*, $\mathcal{P} \vdash_{\mathcal{D}} e\hat{\theta}|_{pos(X,\tau)} \to s \Leftarrow \Pi$ y $\mathcal{P} \vdash_{\mathcal{D}} e\hat{\theta}[s]_{pos(X,\tau)}$ $\to \hat{\theta}(R) \Leftarrow \Pi$ para $s \in Pat_{\mathcal{D}}$ ($s \neq \bot$). A partir de aquí, podemos definir $\hat{\theta}'(R') = s$ y $\hat{\theta}'(X) = \hat{\theta}(X)$ para cualquier $X \neq R'$. Además, como R' es una nueva variable, tenemos pruebas para $\mathcal{P} \vdash_{\mathcal{D}} (e|_{pos(X,\tau)} \to R')\hat{\theta}' \Leftarrow \Pi$ y $\mathcal{P} \vdash_{\mathcal{D}} (e[R']_{pos(X,\tau)} \to R)\hat{\theta}'$ $\Leftarrow \Pi$. Más aún, $\Pi \models_{\mathcal{D}} S\hat{\theta}'$, $Z\hat{\theta}' \equiv s\hat{\theta}'$ para cada $\{Z \mapsto s\} \in \sigma$ y $\mathcal{P} \vdash_{\mathcal{D}} (P \square C)\hat{\theta}'$ $\Leftarrow \Pi$. Por tanto, si consideramos $\theta' =_{\setminus evar(G')} \hat{\theta}'$, tenemos que $\Pi \square \theta' \in Ans_{\mathcal{P}}(G')$ y que $Sol_{\mathcal{D}}(\Pi \square \theta) \subseteq Sol_{\mathcal{D}}(\exists_{\setminus G}.\ \Pi \square \theta')$.*

RRA *Sea $\Pi \square \theta \in Ans_{\mathcal{P}}(G)$. Entonces, ha de existir $\hat{\theta} =_{\setminus evar(G)} \theta$ para la que $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $Z\hat{\theta} \equiv s\hat{\theta}$ para cada $\{Z \mapsto s\} \in \sigma$, $\mathcal{P} \vdash_{\mathcal{D}} (P \square C)\hat{\theta} \Leftarrow \Pi$ y $\mathcal{P} \vdash_{\mathcal{D}} (e \to R)\hat{\theta} \Leftarrow \Pi$ (el árbol definicional se usa sólo para controlar la computación). Usando el* Lema de Demanda*, $\hat{\theta}(R) \neq \bot$ ($R \in dvar_{\mathcal{D}}(P \square C)$ por las condiciones de admisibilidad de G). Podemos asumir que τ es de la forma $f\bar{t}_n$ y entonces $e = \tau\sigma_0$ $= f\overline{t_n\sigma_0}$ y $e\hat{\theta} = f\overline{t_n\sigma_0\hat{\theta}}$. Puesto que $\Pi \square \theta$ es una respuesta no trivial, se tiene $Sat_{\mathcal{D}}(\Pi)$. En consecuencia, $\mathcal{P} \vdash_{\mathcal{D}} (e \to R)\hat{\theta} \Leftarrow \Pi$ usa la CRWL($\mathcal{D}$)-regla $\mathbf{DF}_{\mathcal{P}}$ con una instanciación parcial $(\tau \to r_i \Leftarrow P_i \square C_i)\mu$ de una regla de programa $(\tau \to r_i$ $\Leftarrow P_i \square C_i) \in \mathcal{P}$ (para $1 \leq i \leq k$), donde $\mu \in Sub_{\mathcal{D}}$ con $dom(\mu) \subseteq var(\tau \to r_i \Leftarrow$ $P_i \square C_i)$. Por tanto, el árbol definicional de la deducción $\mathcal{T} : \mathcal{P} \vdash_{\mathcal{D}} (e \to R)\hat{\theta} \Leftarrow \Pi$ ha de ser de la forma $\mathcal{T} \equiv \mathbf{DF}_{\mathcal{P}}\ ((e \to R)\hat{\theta} \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_n, \mathcal{T}_c, \mathcal{T}_r])$, donde $\mathcal{T}_j$: $\mathcal{P} \vdash_{\mathcal{D}} t_j\sigma_0\hat{\theta} \to t_j\mu \Leftarrow \Pi$ ($1 \leq j \leq n$), $\mathcal{T}_c : \mathcal{P} \vdash_{\mathcal{D}} (P_i \square C_i)\mu \Leftarrow \Pi$ y $\mathcal{T}_r : \mathcal{P} \vdash_{\mathcal{D}} r_i\mu$ $\to \hat{\theta}(R) \Leftarrow \Pi$. A partir de aquí, podemos definir $\hat{\theta}'$ como sigue:*

- *Para cualquier variable* $X \in var(\tau \to r_i \Leftarrow P_i \ \square \ C_i) \setminus dom_c(\sigma_0)$, $\hat{\theta}'(X) =_{def} \mu(X)$. *En particular,* $\hat{\theta}'(R_j) = \mu(R_j)$ *para cada* $R_j \in dom_f(\sigma_0)$.

- *Para cualquier variable* $Y \in dom_c(\sigma_0)$, $\hat{\theta}'(Y) =_{def} \hat{\theta}(\sigma_0(Y))$.

- *Para cualquier variable* $Z \notin var(\tau \to r_i \Leftarrow P_i \ \square \ C_i)$, $\hat{\theta}'(Z) =_{def} \hat{\theta}(Z)$.

Entonces $\hat{\theta} =_{\setminus evar(G')} \hat{\theta}'$. *Es más:*

- *Para cualquier* $R_j \in dom_f(\sigma_0)$ $(1 \leq j \leq m)$, $\mathcal{T}'_j : \mathcal{P} \vdash_{\mathcal{D}} \hat{\theta}'(\sigma_f(R_j)) \to \hat{\theta}'(R_j) (= \mu(R_j)) \Leftarrow \Pi$ *es deducible con una prueba que es extraída de una deducción* $\mathcal{T}_j : \mathcal{P} \vdash_{\mathcal{D}} t_j\sigma_0\hat{\theta} \to t_j\mu \Leftarrow \Pi$ $(1 \leq j \leq n)$ *in* $\mathcal{T}$.

- *Usando la* Propiedad de Implicación *y puesto que* $\Pi \models_{\mathcal{D}} r_i\sigma_c\hat{\theta}' \sqsupseteq r_i\mu$, $\hat{\theta}'(R) = \hat{\theta}(R)$ *y* $\mathcal{T}$ *tienen una deducción* $\mathcal{T}_r : \mathcal{P} \vdash_{\mathcal{D}} r_i\mu \to \hat{\theta}(R) \Leftarrow \Pi$, *se sigue que* $\mathcal{P} \vdash_{\mathcal{D}} r_i\sigma_c\hat{\theta}' \to \hat{\theta}'(R) \Leftarrow \Pi$ *es también deducible. Probamos ahora que* $\Pi \models_{\mathcal{D}} \hat{\theta}'(\sigma_c(X)) \sqsupseteq \mu(X)$ *para cada* $X \in var(r_i)$ *(esto implica que* $\Pi \models_{\mathcal{D}} r_i\sigma_c\hat{\theta}' \sqsupseteq r_i\mu$*):*

  - *Si* $X \notin dom_c(\sigma_0)$, *entonces* $\hat{\theta}'(\sigma_c(X)) = \hat{\theta}'(X) = \mu(X)$ *y* $\Pi \models_{\mathcal{D}} \hat{\theta}'(\sigma_c(X)) \sqsupseteq \mu(X)$.
  - *Si* $X \in dom_c(\sigma_0)$, *por la* Propiedad de Aproximación $\hat{\theta}(\sigma_c(X)) = \hat{\theta}(\sigma_c(X))$ *y* $\Pi \models_{\mathcal{D}} \hat{\theta}(\sigma_c(X)) \sqsupseteq \mu(X)$ *debido a que* $\hat{\theta}(\sigma_c(X)), \mu(X) \in Pat_{\mathcal{D}}$ *y a que* $\mathcal{P} \vdash_{\mathcal{D}} \hat{\theta}(\sigma_c(X)) \to \mu(X) \Leftarrow \Pi$ *tiene una derivación en* $\mathcal{T}_j : \mathcal{P} \vdash_{\mathcal{D}} t_j\sigma_0\hat{\theta} \to t_j\mu \Leftarrow \Pi$ $(1 \leq j \leq n)$ *de* $\mathcal{T}$. *Por tanto,* $\Pi \models_{\mathcal{D}} \hat{\theta}'(\sigma_c(X)) \sqsupseteq \mu(X)$.

- $\mathcal{P} \vdash_{\mathcal{D}} (P_i \ \square \ C_i)\sigma_c\hat{\theta}' \Leftarrow \Pi$ *es también CRWL*$(\mathcal{D})$*-deducible, ya que* $(P_i \ \square \ C_i)\sigma_c\hat{\theta}' \sqsupseteq (P_i \ \square \ C_i)\mu$ *y en* $\mathcal{T}_c$ *tenemos deducciones para* $\mathcal{T}_c : \mathcal{P} \vdash_{\mathcal{D}} (P_i \ \square \ C_i)\mu \Leftarrow \Pi$.

- *Por las condiciones de admisibilidad de* G *y la definición de* $\hat{\theta}'$, *también tenemos que* $\Pi \models_{\mathcal{D}} S\hat{\theta}'$, $Z\hat{\theta}' \equiv Z\hat{\theta} \equiv s\hat{\theta} \equiv s\hat{\theta}'$ *para cada* $\{Z \mapsto s\} \in \sigma$ *y* $\mathcal{P} \vdash_{\mathcal{D}} (P \ \square \ C)\hat{\theta}') \Leftarrow \Pi$.

En consecuencia, si tomamos $\theta' =_{\setminus evar(G')} \hat{\theta}'$ *tenemos que* $\Pi \ \square \ \theta' \in Ans_{\mathcal{P}}(G')$, *y claramente* $Sol_{\mathcal{D}}(\Pi \ \square \ \theta) \subseteq Sol_{\mathcal{D}}(\exists_{\setminus G}. \ \Pi \ \square \ \theta')$.

CS *Consideramos* $\Pi \ \square \ \theta \in Ans_{\mathcal{P}}(G)$. *Entonces existe* $\hat{\theta} =_{\setminus evar(G)} \theta$ *tal que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, $X\hat{\theta} \equiv t\hat{\theta}$ *para cualquier* $\{X \mapsto t\} \in \sigma$ *y* $\overline{\mathcal{T}} : \mathcal{P} \vdash_{\mathcal{D}} (P \square C)\hat{\theta} \Leftarrow \Pi$. *Para cada* $1 \leq i \leq k$, *definimos* $\hat{\theta}_i = umg(\hat{\theta}, \sigma_i)$ *como el* unificador de máxima generalidad *[BN98] de las sustituciones* $\hat{\theta}$ *y* σ_i *(existen* $\rho_i, \delta_i \in Sub_{\mathcal{D}}$ *para los que* $\hat{\theta}_i = \hat{\theta}\rho_i$ *y* $\hat{\theta}_i = \sigma_i\delta_i$*) y* $\Pi_i = \Pi\rho_i \wedge S_i\delta_i$. *Tenemos entonces que:*

- $\Pi_i \models_{\mathcal{D}} S_i\hat{\theta}_i$. *Sea* $\mu \in Sol_{\mathcal{D}}(\Pi_i)$. *Por definición de* Π_i, $\mu \in Sol_{\mathcal{D}}(\Pi\rho_i)$ *y* $\mu \in Sol_{\mathcal{D}}(S_i\delta_i)$. *De aquí,* $\rho_i\mu \in Sol_{\mathcal{D}}(\Pi)$ *y* $\delta_i\mu \in Sol_{\mathcal{D}}(S_i)$. *Como* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, *se tiene que* $\rho_i\mu \in Sol_{\mathcal{D}}(S\hat{\theta})$ *y* $\delta_i\mu \in Sol_{\mathcal{D}}(S_i)$. *Entonces,* $\hat{\theta}\rho_i\mu \in Sol_{\mathcal{D}}(S)$ *y* $\delta_i\mu \in Sol_{\mathcal{D}}(S_i)$. *Puesto que* $\hat{\theta}_i = \hat{\theta}\rho_i$, *tenemos que* $\hat{\theta}_i\mu \in Sol_{\mathcal{D}}(S)$ *y* $\delta_i\mu \in Sol_{\mathcal{D}}(S_i)$ $(1 \leq i \leq k)$. *Usando el requerimiento de un resolutor idealmente completo de restricciones de que* $Sol_{\mathcal{D}}(S) = \bigcup_{i=1}^{k} Sol_{\mathcal{D}}(\exists_{\setminus S}.\ S_i \square \sigma_i)$, *se sigue que también* $\hat{\theta}_i\mu \in Sol_{\mathcal{D}}(S_i)$, *y entonces* $\mu \in Sol_{\mathcal{D}}(S_i\hat{\theta}_i)$.

- $X\hat{\theta}_i \equiv t\hat{\theta}_i$ *para cada* $\{X \mapsto t\} \in \sigma\sigma_i$. *Si* $\{X \mapsto t\} \in \sigma$ *entonces* $X\hat{\theta}_i \equiv X\hat{\theta}\rho_i \equiv t\hat{\theta}\rho_i \equiv t\hat{\theta}_i$. *Si* $\{X \mapsto t\} \in \sigma_i$ *entonces* $X\hat{\theta}_i \equiv X\sigma_i\delta_i \equiv t\delta_i \equiv t\sigma_i\delta_i \equiv t\hat{\theta}_i$.

- $\overline{\mathcal{T}}_i : \mathcal{P} \vdash_{\mathcal{D}} (P \square C)\sigma_i\hat{\theta}_i \Leftarrow \Pi_i$. *Tenemos que* $((P \square C)\hat{\theta} \Leftarrow \Pi) \succcurlyeq_{\mathcal{D}} ((P \square C)\hat{\theta}_i \Leftarrow \Pi_i)$ *(si* $(e \to t) \in P$ *entonces existe* ρ_i *para el que* $\Pi_i \models_{\mathcal{D}} e\hat{\theta}\rho_i = e\hat{\theta}_i$, $\Pi_i \models_{\mathcal{D}} t\hat{\theta}\rho_i = t\hat{\theta}_i$ *y* $\Pi_i \models_{\mathcal{D}} \Pi\rho_i$. *Análogamente para cada* $(p\overline{e}_n \to!\ t) \in C$*)* *y* $\hat{\theta}_i = \sigma_i\delta_i = (\sigma_i\sigma_i)\delta_i = \sigma_i(\sigma_i\delta_i) = \sigma_i\hat{\theta}_i$. *Entonces,* $((P \square C)\hat{\theta} \Leftarrow \Pi) \succcurlyeq_{\mathcal{D}} ((P \square C)\sigma_i\hat{\theta}_i \Leftarrow \Pi_i)$. *Como* $\overline{\mathcal{T}} : \mathcal{P} \vdash_{\mathcal{D}} (P \square C)\hat{\theta} \Leftarrow \Pi$, *usando la* Propiedad de Implicación *también tenemos que* $\overline{\mathcal{T}}_i : \mathcal{P} \vdash_{\mathcal{D}} (P \square C)\sigma_i\hat{\theta}_i \Leftarrow \Pi_i$.

Si consideramos ahora $\theta_i =_{\setminus evar(G_i)} \hat{\theta}_i$ *para todo* $1 \leq i \leq k$, *entonces* $\Pi_i \square \theta_i \in Ans_{\mathcal{D}}(G_i)$. *Finalmente, probamos que* $Sol_{\mathcal{D}}(\Pi \square \theta) \subseteq \bigcup_{i=1}^{k} Sol_{\mathcal{D}}(\exists_{\setminus G}.\ \Pi_i \square \theta_i)$. *Sea* $\mu \in Sol_{\mathcal{D}}(\Pi \square \theta)$. *Por definición,* $\mu \in Sol_{\mathcal{D}}(\Pi)$ *y* $X\mu \equiv t\mu$ *para cualquier* $\{X \mapsto t\} \in \theta$. *Puesto que* $\Pi \models_{\mathcal{D}} S\hat{\theta}$, *tenemos que* $\mu \in Sol_{\mathcal{D}}(S\hat{\theta})$, *y de aquí* $\hat{\theta}\mu \in Sol_{\mathcal{D}}(S)$. *Ahora, usando de nuevo el requerimiento de un resolutor idealmente completo de restricciones de que* $Sol_{\mathcal{D}}(S) = \bigcup_{i=1}^{k} Sol_{\mathcal{D}}(\exists_{\setminus S}.\ S_i \square \sigma_i)$, *se sigue que* $\mu \in Sol_{\mathcal{D}}(\exists_{\setminus G}.\ S_i\hat{\theta} \square \sigma_i\hat{\theta})$ $(1 \leq i \leq k)$. *Por definición, ha de existir* $\mu' =_{\setminus G} \mu$ *tal que* $\mu' \in Sol_{\mathcal{D}}(S_i\hat{\theta})$ *y* $X\mu' \equiv t\mu'$ *para cada* $\{X \mapsto t\} \in \sigma_i\hat{\theta}$. *Como* $\hat{\theta}_i = umg(\hat{\theta}, \sigma_i)$, *tenemos que* $X\mu' \equiv t\mu'$ *para cualquier* $\{X \mapsto t\} \in \hat{\theta}_i$. *Más aún, debido a que* $\hat{\theta}_i = \sigma_i\delta_i$, $S_i\sigma_i = S_i$ *y* $\Pi_i = \Pi\rho_i \wedge S_i\delta_i$, *se tiene que* $\mu' \in Sol_{\mathcal{D}}(\Pi_i)$. *Se sigue entonces que* $\mu \in Sol_{\mathcal{D}}(\exists_{\setminus G}.\ \Pi_i \square \theta_i)$ $(1 \leq i \leq k)$. *En consecuencia,* $\mu \in \bigcup_{i=1}^{k} Sol_{\mathcal{D}}(\exists_{\setminus G}.\ \Pi_i \square \theta_i)$.

AC *Análogo al caso de* **PF**. □

A.3.6. Propiedades del algoritmo de transformación $COIS$

Demostramos finalmente en esta subsección las principales propiedades del algoritmo de transformación de $CFLP(\mathcal{D})$-programas presentado en la Subsección 4.3.8.

Demostración 40 (Demostración del Teorema 10) *Por la construcción de* $\mathcal{P}'$ *como un* $COISS(\mathcal{D})$ *usando el* algoritmo de transformación, *ha de existir* $n \in \mathbb{N}$ *y* $CFLP(\mathcal{D})$*-programas* $\mathcal{P}_i$ $(0 \leq i \leq n)$ *tales que* $\mathcal{P}_0 = \mathcal{P}$, $\mathcal{P}_n = \mathcal{P}'$, *y para todo* $0 < i \leq n$, *la relación entre* $\mathcal{P}_{i-1}$ *y* $\mathcal{P}_i$ *corresponde a la aplicación del tercer caso en la especificación de la función* $COIS$*: existe un patrón de llamada* τ *con raíz* f *y una*

posición demandada (pero no uniformemente demandada) $p \in VPos(\tau)$ *por* $(\mathcal{P}_{i-1})_f$ *y* $\mathcal{P}_i = (\mathcal{P}_{i-1} \setminus \{f\}) \cup \mathcal{P}_{f_1} \cup \mathcal{P}_{f_2} \cup \{\tau \to \tau_1, \tau \to \tau_2\}$*, donde:*

- f_1*,* f_2 *son nuevos símbolos de función con la misma aridad que* f*.*
- $\tau_1 = \tau\ \{f \mapsto f_1\}$ *y* $\tau_2 = \tau\ \{f \mapsto f_2\}$*.*
- $\mathcal{P}_{i-1} \setminus \{f\}$ *es el subconjunto de reglas de programa de* $\mathcal{P}_{i-1}$ *que no definen* f*.*
- $\mathcal{P}_{f_1}$ *es el subconjunto de reglas de programa de* $\mathcal{P}_{i-1}$ *que definen* f *y que demandan* p*, con todos los lados izquierdos afectados por el cambio* $\{f \mapsto f_1\}$*.*
- $\mathcal{P}_{f_2}$ *es el subconjunto de reglas de programa de* $\mathcal{P}_{i-1}$ *que definen* f *pero que sin embargo no demandan* p*, con todos los lados izquierdos afectados por el cambio* $\{f \mapsto f_2\}$*.*

Probamos que para todo $0 < i \leq n$ *se verifica que* $\mathcal{P}_{i-1} \vdash_{\mathcal{D}} e \to t \Leftarrow \Pi$ *si y sólo si* $\mathcal{P}_i \vdash_{\mathcal{D}} e \to t \Leftarrow \Pi$*.*

$\Rightarrow$*) Supongamos que* $\mathcal{P}_{i-1} \vdash_{\mathcal{D}} e \to t \Leftarrow \Pi$ *(*$0 < i \leq n$*). Razonamos por inducción sobre el tamaño de esta CRWL*$(\mathcal{D})$*-deducción y consideramos casos de acuerdo con la última regla de inferencia usada en la derivación. Si esta CRWL*$(\mathcal{D})$*-regla usada es* **TI***,* **RR** *o bien* **SP** *entonces el resultado se satisface directamente con la misma regla. Si la regla es* **DC***,* **IR***,* **FV***,* **PF** *o bien* **AC** *el resultado también se cumple usando la* hipótesis de inducción *y la misma regla de deducción. Finalmente, supongamos que la última regla que se ha usado es* $\mathbf{DF}_{\mathcal{P}}$*. Consideramos dos subcasos:*

- Subcaso 1*:* $e = f\overline{e}_n\overline{a}_k$ *con* f *el símbolo de función en* $\mathcal{P}_{i-1}$ *afectado por el paso de transformación de* $\mathcal{P}_{i-1}$ *a* $\mathcal{P}_i$*. En este caso,* $\mathcal{T} \equiv \mathbf{DF}_{\mathcal{P}_{\mathbf{i-1}}}\ (f\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_n, \mathcal{T}_c, \mathcal{T}_r, \mathcal{T}_s])$ *usando* $(f\overline{t}_n \to r \Leftarrow P\ \square\ C) \in [\mathcal{P}_{i-1}]_\perp$*,* $s \in Pat_{\mathcal{D}}$ *y donde* $\mathcal{T}_j : \mathcal{P}_{i-1} \vdash_{\mathcal{D}} e_j \to t_j \Leftarrow \Pi$ *para todo* $1 \leq j \leq n$*,* $\mathcal{T}_c : \mathcal{P}_{i-1} \vdash_{\mathcal{D}} P\ \square\ C \Leftarrow \Pi$*,* $\mathcal{T}_r : \mathcal{P}_{i-1} \vdash_{\mathcal{D}} r \to s \Leftarrow \Pi$ *y* $\mathcal{T}_s : \mathcal{P}_{i-1} \vdash_{\mathcal{D}} s\overline{a}_k \to t \Leftarrow \Pi$*. Por* hipótesis de inducción*,* $\mathcal{T}'_j : \mathcal{P}_i \vdash_{\mathcal{D}} e_j \to t_j \Leftarrow \Pi$ *para todo* $1 \leq j \leq n$*,* $\mathcal{T}'_c : \mathcal{P}_i \vdash_{\mathcal{D}} P\ \square\ C \Leftarrow \Pi$*,* $\mathcal{T}'_r : \mathcal{P}_i \vdash_{\mathcal{D}} r \to s \Leftarrow \Pi$ *y* $\mathcal{T}'_s : \mathcal{P}_i \vdash_{\mathcal{D}} s\overline{a}_k \to t \Leftarrow \Pi$*. Más aún, como* $(f\overline{t}_n \to r \Leftarrow P\ \square\ C) \in [\mathcal{P}_{i-1}]_\perp$*, por definición de* $\mathcal{P}_i$ *han de existir instancias* $(f\overline{t}_n \to f_k\overline{t}_n) \in [\mathcal{P}_i]_\perp$ *y* $(f_k\overline{t}_n \to r \Leftarrow P\ \square\ C) \in [\mathcal{P}_i]_\perp$ *con* $k = 1$ *or* 2*. Por tanto, podemos construir* $\mathcal{T}' \equiv \mathbf{DF}_{\mathcal{P}_{\mathbf{i}}}\ (f\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi, [\mathcal{T}'_1, \ldots, \mathcal{T}'_n, \mathcal{T}'_{f_k}, \mathcal{T}'_s])$ *donde* $\mathcal{T}'_{f_k} \equiv \mathbf{DF}_{\mathcal{P}_{\mathbf{i}}}\ (f_k\overline{t}_n \to s \Leftarrow \Pi, [\mathcal{T}''_1, \ldots, \mathcal{T}''_n, \mathcal{T}'_c, \mathcal{T}'_r])$ *con* $\mathcal{T}''_j : \mathcal{P}_i \vdash_{\mathcal{D}} t_j \to t_j \Leftarrow \Pi$ *para todo* $1 \leq j \leq n$*. En consecuencia,* $\mathcal{T}' : \mathcal{P}_i \vdash_{\mathcal{D}} e \to t \Leftarrow \Pi$*.*
- Subcaso 2*:* $e = g\overline{e}_n\overline{a}_k$*, donde* g *es un símbolo de función en* $\mathcal{P}_{i-1}$ *diferente del símbolo de función* f *afectado por el paso de transformación de* $\mathcal{P}_{i-1}$ *a* $\mathcal{P}_i$*. En este caso,* $\mathcal{T} \equiv \mathbf{DF}_{\mathcal{P}_{\mathbf{i-1}}}\ (g\overline{e}_n\overline{a}_k \to t \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_n, \mathcal{T}_c, \mathcal{T}_r, \mathcal{T}_s])$ *usando* $(g\overline{t}_n \to r \Leftarrow P\ \square\ C) \in [\mathcal{P}_{i-1}]_\perp$*,* $s \in Pat_{\mathcal{D}}$ *y donde* $\mathcal{T}_j : \mathcal{P}_{i-1} \vdash_{\mathcal{D}} e_j \to t_j \Leftarrow \Pi$

para todo $1 \leq j \leq n$, $\mathcal{T}_c : \mathcal{P}_{i-1} \vdash_{\mathcal{D}} P \ \square \ C \Leftarrow \Pi$, $\mathcal{T}_r : \mathcal{P}_{i-1} \vdash_{\mathcal{D}} r \rightarrow s \Leftarrow \Pi$ *y* $\mathcal{T}_s : \mathcal{P}_{i-1} \vdash_{\mathcal{D}} s\overline{a}_k \rightarrow t \Leftarrow \Pi$. *Por* hipótesis de inducción, $\mathcal{T}'_j : \mathcal{P}_i \vdash_{\mathcal{D}} e_j \rightarrow t_j \Leftarrow \Pi$ *para todo* $1 \leq j \leq n$, $\mathcal{T}'_c : \mathcal{P}_i \vdash_{\mathcal{D}} P \ \square \ C \Leftarrow \Pi$, $\mathcal{T}'_r : \mathcal{P}_i \vdash_{\mathcal{D}} r \rightarrow s \Leftarrow \Pi$ *y* $\mathcal{T}'_s : \mathcal{P}_i \vdash_{\mathcal{D}} s\overline{a}_k \rightarrow t \Leftarrow \Pi$. *Más aún, como* $(g\overline{t}_n \rightarrow r \Leftarrow P \ \square \ C) \in [\mathcal{P}_{i-1}]_\perp$ *donde* $g \neq f$, *por definición de* $\mathcal{P}_i$, $(g\overline{t}_n \rightarrow r \Leftarrow P \ \square \ C) \in [\mathcal{P}_i]_\perp$. *Entonces, podemos construir* $\mathcal{T}' \equiv \mathbf{DF}_{\mathcal{P}_i}\ (g\overline{e}_n\overline{a}_k \rightarrow t \Leftarrow \Pi, [\mathcal{T}'_1, \ldots, \mathcal{T}'_n, \mathcal{T}'_c, \mathcal{T}'_r, \mathcal{T}'_s])$. *En consecuencia,* $\mathcal{T}' : \mathcal{P}_i \vdash_{\mathcal{D}} e \rightarrow t \Leftarrow \Pi$.

$\Leftarrow$) *Asumimos ahora que* $\mathcal{P}_i \vdash_{\mathcal{D}} e \rightarrow t \Leftarrow \Pi$. *Razonamos por inducción sobre el tamaño de esta CRWL(*$\mathcal{D}$*)-deducción y distinguimos casos de acuerdo con la última regla de inferencia usada en la derivación. Si la CRWL(*$\mathcal{D}$*)-regla usada es* **TI**, **RR** *o bien* **SP** *entonces el resultado se cumpliría directamente con la misma regla. En otro caso, si la regla usada es* **DC**, **IR**, **FV**, **PF** *o bien* **AC**, *entonces el resultado se satisface aplicando la* hipótesis de inducción *y la misma regla de deducción. Finalmente, supongamos que la última regla usada en la deducción es* $\mathbf{DF}_{\mathcal{P}}$. *Distinguimos dos subcasos diferentes:*

- Subcaso 1*:* $e = f\overline{e}_n\overline{a}_k$ *con* f *el símbolo de función en* $\mathcal{P}_{i-1}$ *afectado por el paso de transformación de* $\mathcal{P}_{i-1}$ *a* $\mathcal{P}_i$. *En este caso,* $\mathcal{T} \equiv \mathbf{DF}_{\mathcal{P}_i}\ (f\overline{e}_n\overline{a}_k \rightarrow t \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_n, \mathcal{T}_{f_k}, \mathcal{T}_s])$ *usando* $(f\overline{t}_n \rightarrow f_k\overline{t}_n) \in [\mathcal{P}_i]_\perp$ *(*$k = 1$ *or* 2*) y* $s \in Pat_{\mathcal{D}}$. *Entonces,* $\mathcal{T}_j : \mathcal{P}_i \vdash_{\mathcal{D}} e_j \rightarrow t_j \Leftarrow \Pi$ *para todo* $1 \leq j \leq n$, $\mathcal{T}_s : \mathcal{P}_i \vdash_{\mathcal{D}} s\overline{a}_k \rightarrow t \Leftarrow \Pi$ *y* $\mathcal{T}_{f_k} \equiv \mathbf{DF}_{\mathcal{P}_i}\ (f_k\overline{t}_n \rightarrow s \Leftarrow \Pi, [\mathcal{T}'_1, \ldots, \mathcal{T}'_n, \mathcal{T}_c, \mathcal{T}_r])$ *con* $\mathcal{T}'_j : \mathcal{P}_i \vdash_{\mathcal{D}} t_j \rightarrow s_j \Leftarrow \Pi$ *para cualquier* $1 \leq j \leq n$, $\mathcal{T}_c : \mathcal{P}_i \vdash_{\mathcal{D}} P \ \square \ C \Leftarrow \Pi$ *y* $\mathcal{T}_r : \mathcal{P}_i \vdash_{\mathcal{D}} r \rightarrow s \Leftarrow \Pi$, *usando* $(f_k\overline{s}_n \rightarrow r \Leftarrow P \ \square \ C) \in [\mathcal{P}_i]_\perp$. *Por* hipótesis de inducción, $\mathcal{T}^*_j : \mathcal{P}_{i-1} \vdash_{\mathcal{D}} e_j \rightarrow t_j \Leftarrow \Pi$ *para todo* $1 \leq j \leq n$ *y* $\mathcal{T}'^*_j : \mathcal{P}_{i-1} \vdash_{\mathcal{D}} t_j \rightarrow s_j \Leftarrow \Pi$ *para todo* $1 \leq j \leq n$ *(de aquí, usando las propiedades de CRWL(*$\mathcal{D}$*),* $\mathcal{T}''_j : \mathcal{P}_{i-1} \vdash_{\mathcal{D}} e_j \rightarrow s_j \Leftarrow \Pi$ *para cualquier* $1 \leq j \leq n$*),* $\mathcal{T}'_c : \mathcal{P}_{i-1} \vdash_{\mathcal{D}} P \ \square \ C \Leftarrow \Pi$, $\mathcal{T}'_r : \mathcal{P}_{i-1} \vdash_{\mathcal{D}} r \rightarrow s \Leftarrow \Pi$ *y* $\mathcal{T}'_s : \mathcal{P}_{i-1} \vdash_{\mathcal{D}} s\overline{a}_k \rightarrow t \Leftarrow \Pi$. *Como* $(f\overline{t}_n \rightarrow f_k\overline{t}_n) \in [\mathcal{P}_i]_\perp$ *y* $(f_k\overline{s}_n \rightarrow r \Leftarrow P \ \square \ C) \in [\mathcal{P}_i]_\perp$ *con* $k = 1$ *or* 2, *existe una instancia* $(f\overline{s}_n \rightarrow r \Leftarrow P \ \square \ C) \in [\mathcal{P}_{i-1}]_\perp$, *debido a la definición de* $\mathcal{P}_i$ *a partir de* $\mathcal{P}_{i-1}$ *y porque* f *es un símbolo de función que aparece en* $\mathcal{P}_{i-1}$ *por hipótesis. Entonces,* $\mathcal{T}' \equiv \mathbf{DF}_{\mathcal{P}_{i-1}}\ (f\overline{e}_n\overline{a}_k \rightarrow t \Leftarrow \Pi, [\mathcal{T}''_1, \ldots, \mathcal{T}''_n, \mathcal{T}'_c, \mathcal{T}'_r, \mathcal{T}'_s])$. *En consecuencia,* $\mathcal{T}' : \mathcal{P}_{i-1} \vdash_{\mathcal{D}} e \rightarrow t \Leftarrow \Pi$.

- Subcaso 2*:* $e = g\overline{e}_n\overline{a}_k$, *donde* g *es un símbolo de función en* $\mathcal{P}_{i-1}$ *diferente del símbolo de función* f *afectado por el paso de transformación de* $\mathcal{P}_{i-1}$ *a* $\mathcal{P}_i$. *En este caso,* $\mathcal{T} \equiv \mathbf{DF}_{\mathcal{P}_i}\ (g\overline{e}_n\overline{a}_k \rightarrow t \Leftarrow \Pi, [\mathcal{T}_1, \ldots, \mathcal{T}_n, \mathcal{T}_c, \mathcal{T}_r, \mathcal{T}_s])$ *usando* $(g\overline{t}_n \rightarrow r \Leftarrow P \ \square \ C) \in [\mathcal{P}_i]_\perp$, $s \in Pat_{\mathcal{D}}$ *y donde* $\mathcal{T}_j : \mathcal{P}_i \vdash_{\mathcal{D}} e_j \rightarrow t_j \Leftarrow \Pi$ *para cualquier* $1 \leq j \leq n$, $\mathcal{T}_c : \mathcal{P}_i \vdash_{\mathcal{D}} P \ \square \ C \Leftarrow \Pi$, $\mathcal{T}_r : \mathcal{P}_i \vdash_{\mathcal{D}} r \rightarrow s \Leftarrow \Pi$ *y* $\mathcal{T}_s : \mathcal{P}_i \vdash_{\mathcal{D}} s\overline{a}_k \rightarrow t \Leftarrow \Pi$. *Por* hipótesis de inducción, $\mathcal{T}'_j : \mathcal{P}_{i-1} \vdash_{\mathcal{D}} e_j \rightarrow t_j \Leftarrow \Pi$ *para todo* $1 \leq j \leq n$, $\mathcal{T}'_c : \mathcal{P}_{i-1} \vdash_{\mathcal{D}} P \ \square \ C \Leftarrow \Pi$, $\mathcal{T}'_r : \mathcal{P}_{i-1} \vdash_{\mathcal{D}} r \rightarrow s \Leftarrow \Pi$ *y* $\mathcal{T}'_s : \mathcal{P}_{i-1} \vdash_{\mathcal{D}} s\overline{a}_k \rightarrow t \Leftarrow \Pi$. *Es más, como* $(g\overline{t}_n \rightarrow r \Leftarrow P \ \square \ C) \in [\mathcal{P}_i]_\perp$ *donde* $g \neq f$, *por definición de* $\mathcal{P}_i$, $(g\overline{t}_n \rightarrow r \Leftarrow P \ \square \ C) \in [\mathcal{P}_{i-1}]_\perp$. *Entonces,*

podemos construir $\mathcal{T}' \equiv \mathbf{DF}_{\mathcal{P}_{i-1}}\ (g\overline{e}_n\overline{a}_k \rightarrow t \Leftarrow \Pi,\ [\mathcal{T}'_1,\ \ldots,\ \mathcal{T}'_n,\ \mathcal{T}'_c,\ \mathcal{T}'_r,\ \mathcal{T}'_s])$. *En consecuencia,* $\mathcal{T}' : \mathcal{P}_{i-1} \vdash_{\mathcal{D}} e \rightarrow t \Leftarrow \Pi$.

□

A.4. Resultados presentados en el Capítulo 6

En esta última sección del Apéndice damos las demostraciones de los principales resultados que aparecen enunciados en el Capítulo 6 referentes al cálculo semántico $CNPC(\mathcal{D})$ y al cálculo de resolución de objetivos $RGSC(\mathcal{D})$. Asímismo, incluimos las demostraciones de todos aquellos resultados auxiliares que son utilizados en este capítulo y en las pruebas de los resultados principales, como el *Lema de Conjunción* y sus correspondientes lemas auxiliares.

A.4.1. Corrección semántica del cálculo $CNPC(\mathcal{D})$

Comenzamos demostrando la corrección semántica del cálculo de prueba negativo $CNPC(\mathcal{D})$ enunciada en el Teorema 15, el cual permite demostrar que cualquier *aca* que haya sido derivado por medio de un árbol de prueba negativo es consecuencia lógica de la teoría negativa asociada al programa utilizado.

Demostración 41 (Demostración del Teorema 15) *Para cada una de las reglas de inferencia dadas en el cálculo* $CNPC(\mathcal{D})$ *probamos que un modelo arbitrario* $\mathcal{I} \vDash_{\mathcal{D}} \mathcal{P}^-$*, para el que las premisas de la regla son válidas en* $\mathcal{I}$*, también verifica que la conclusión de la regla es válida en* $\mathcal{I}$*. Una vez verificado esto, la tesis del Teorema 15 se sigue de una sencilla inducción sobre la profundidad p de un NPT testigo de* $\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} G$*, como se ha visto en la Subsección 6.4.1. En lo que sigue, obsérvese que la suposición de que* $\mathcal{I}$ *es modelo de* $\mathcal{P}^-$ *solo es necesaria al razonar el caso de la regla de inferencia* $\mathbf{(DF)}_f$.

(SF) $Sol_{\mathcal{I}}(R \sqcap S) = Sol_{\mathcal{I}}(R) \cap Sol_{\mathcal{D}}(S) \subseteq Sol_{\mathcal{D}}(S) \subseteq Sol_{\mathcal{D}}(D)$.

(CJ) *Sea* $\mu \in Sol_{\mathcal{I}}((R_1 \wedge R_2) \sqcap S)$*. Entonces,* $\mu \in Sol_{\mathcal{I}}(R_1 \sqcap S)$ *y* $\mu \in Sol_{\mathcal{I}}(R_2)$*. Por la hipótesis de que las premisas son válidas en* $\mathcal{I}$*, ha de existir* $i \in I$ *para el que* $\mu \in Sol_{\mathcal{D}}(\exists\overline{Z}_i.\ S_i)$*. Más aún, ha de existir* $\mu' =_{\setminus\overline{Z}_i} \mu$ *tal que* $\mu' \in Sol_{\mathcal{D}}(S_i)$*. Como* $\overline{Z}_i \cap var(R_2) = \emptyset$*, también obtenemos que* $\mu' \in Sol_{\mathcal{I}}(R_2)$*. En consecuencia,* $\mu' \in Sol_{\mathcal{I}}(R_2 \wedge S_i)$*, y por tanto, deducimos que también* $\mu' \in Sol_{\mathcal{I}}(R_2\ \&\ S_i)$ *por las propiedades del operador* &*. Por* hipótesis de inducción*, existe* $j \in J_i$ *tal que* $\mu' \in Sol_{\mathcal{D}}(\exists\overline{Z}_{ij}.\ S_{ij})$*. Más aún, existe* $\mu'' =_{\setminus\overline{Z}_{ij}} \mu'$ *tal que* $\mu'' \in Sol_{\mathcal{D}}(S_{ij})$*. Por tanto,* $\mu'' =_{\setminus\{\overline{Z}_i,\overline{Z}_{ij}\}} \mu$*, y podemos entonces concluir que* $\mu \in Sol_{\mathcal{D}}(\exists\overline{Z}_i, \overline{Z}_{ij}.\ S_{ij})$.

(TS) *Distinguimos varios casos de acuerdo con la forma del aca* φ*:*

- *Si* $\varphi \equiv (e \to t) \sqcap S \Rightarrow D$ *y* $e \to t$ *tiene un* conflicto *(es decir,* $e \equiv h\overline{e}_m \notin Pat_{\mathcal{D}}$ *es pasivo y* $t \in \mathcal{B}^{\mathcal{D}}$*, o bien* $t \equiv h'\overline{t}_{m'}$ *con* $h \neq h'$ *o* $m \neq m'$*), entonces* $Sol_{\mathcal{I}}(e \to t) = \emptyset$*. En consecuencia,* $Sol_{\mathcal{I}}((e \to t) \sqcap S) = Sol_{\mathcal{I}}(e \to t) \cap Sol_{\mathcal{D}}(S) = \emptyset \subseteq Sol_{\mathcal{D}}(D)$*.*
- *Si* $\varphi \equiv (e \to t) \sqcap S \Rightarrow D$ *y* $Sol_{\mathcal{D}}(D) = Val_{\mathcal{D}}$ *entonces* $Sol_{\mathcal{I}}((e \to t) \sqcap S) \subseteq Val_{\mathcal{D}} = Sol_{\mathcal{D}}(D)$*.*
- *Si* $\varphi \equiv (p\overline{e}_n \to!\ t) \sqcap S \Rightarrow D$ *y* $Sol_{\mathcal{D}}(D) = Val_{\mathcal{D}}$ *entonces* $Sol_{\mathcal{I}}(p\overline{e}_n \to!\ t \sqcap S) \subseteq Val_{\mathcal{D}} = Sol_{\mathcal{D}}(D)$*.*

(DC) *Sea* $\mu \in Sol_{\mathcal{I}}(h\overline{e}_m \to h\overline{t}_m \sqcap S)$*. Por definición,* $\mathcal{I} \Vdash_{\mathcal{D}} h\overline{e_m\mu} \to h\overline{t_m\mu}$*, es decir, existe una inferencia de la forma*

$$\frac{\cdots\ \mathcal{I} \Vdash_{\mathcal{D}} e_i\mu \to t_i\mu\ \cdots}{\mathcal{I} \Vdash_{\mathcal{D}} h\overline{e_m\mu} \to h\overline{t_m\mu}}\ \textbf{(DC)}$$

Puesto que $\mathcal{I} \Vdash_{\mathcal{D}} e_i\mu \to t_i\mu$ *para cada* $1 \leq i \leq m$*, por definición,* $\mu \in Sol_{\mathcal{I}}(e_i \to t_i)$*. Más aún, como* $\mu \in Sol_{\mathcal{D}}(S)$*, se obtiene que* $\mu \in Sol_{\mathcal{I}}(\overline{e_m \to t_m} \sqcap S)$*. Finalmente, por la hipótesis de que la premisa es válida en* $\mathcal{I}$*,* $\mu \in Sol_{\mathcal{D}}(D)$*.*

(IM) *Sea* $\mu \in Sol_{\mathcal{I}}(h\overline{e}_m \to X \sqcap S)$*. Entonces,* $\mu \in Sol_{\mathcal{I}}(h\overline{e}_m \to X)$ *y* $\mu \in Sol_{\mathcal{D}}(S)$*. Por definición,* $\mathcal{I} \Vdash_{\mathcal{D}} h\overline{e_m\mu} \to \mu(X)$*. Puesto que* $\mu \in Val_{\mathcal{D}}$*,* $\mu(X) \notin \mathcal{V}ar$*. Es más, como* $h\overline{e}_m \notin Pat_{\mathcal{D}}$ *es pasivo y no trivial, entonces* $\mu(X) = h\overline{t}_m$*. Así, ha de existir una inferencia de la forma*

$$\frac{\cdots\ \mathcal{I} \Vdash_{\mathcal{D}} e_i\mu \to t_i\ \cdots}{\mathcal{I} \Vdash_{\mathcal{D}} h\overline{e_m\mu} \to h\overline{t}_m}\ \textbf{(DC)}$$

Definimos ahora $\mu'(X_i) = t_i$ *para cada* $1 \leq i \leq m$*, y* $\mu'(Y) = \mu(Y)$ *para todo* $Y \notin \{\overline{X}_m\}$*. Obviamente,* $\mu' =_{\setminus\{\overline{X}_m\}} \mu$*. Más aún,* $e_i\mu = e_i\mu'$ *debido a que* $\overline{X}_m$ *son variables nuevas para cualquier* $1 \leq i \leq m$*. Por tanto, como* $\mathcal{I} \Vdash_{\mathcal{D}} e_i\mu \to t_i$*, se obtiene que* $\mathcal{I} \Vdash_{\mathcal{D}} e_i\mu' \to \mu'(X_i)$*, y entonces* $\mu' \in Sol_{\mathcal{I}}(e_i \to X_i)$ *para cada* $1 \leq i \leq m$*. Así pues,* $\mu' \in Sol_{\mathcal{I}}(\overline{e_m \to X_m})$*. Como además* $\overline{X}_m$ *son variables nuevas en* S *y* $\mu \in Sol_{\mathcal{D}}(S)$*, entonces también* $\mu' \in Sol_{\mathcal{D}}(S)$*. Por otra parte,* $(h\overline{X}_m)\mu' = h\overline{\mu'(X_m)} = h\overline{t}_m = \mu(X) = \mu'(X)$ *debido a que* X *es una variable distinta de* $\overline{X}_m$*, y entonces también* $\mu' \in Sol(\{X \mapsto h\overline{X}_m\})$*. En consecuencia,* $\mu' \in Sol_{\mathcal{I}}(\overline{e_m \to X_m} \sqcap (S\ @\ \{X \mapsto h\overline{X}_m\}))$*. Por la hipótesis de que la premisa es válida en* $\mathcal{I}$*, ha de existir* $i \in I$ *para el que* $\mu' \in Sol_{\mathcal{D}}(\exists\overline{Z}_i.\ S_i)$*. Finalmente, puesto que* $\mu =_{\setminus\{X_m\}} \mu'$*, deducimos que* $\mu \in Sol_{\mathcal{D}}(\overline{X}_m, \overline{Z}_i.\ S_i)$*.*

(AR)$_p$ *Sea* $\mu \in Sol_{\mathcal{I}}(p\overline{e}_n \to^?\ t \sqcap S)$*. Entonces,* $\mu \in Sol_{\mathcal{I}}(p\overline{e}_n \to^?\ t)$ *y* $\mu \in Sol_{\mathcal{D}}(S)$*. Por definición, ha de existir una inferencia de la forma*

$$\frac{\cdots\ \mathcal{I} \Vdash_{\mathcal{D}} e_i\mu \to t_i\ \cdots}{\mathcal{I} \Vdash_{\mathcal{D}} p\overline{e_n\mu} \to^?\ t\mu}\ \textbf{(PF)}_p \text{ or } \textbf{(AC)}_p$$

con $\models_{\mathcal{D}} p\bar{t}_n \rightarrow^{?} t\mu$. *Definamos ahora* $\mu'(X_i) = t_i$ *para cada* $1 \leq i \leq n$, *y* $\mu'(Z) = \mu(Z)$ *para toda* $Z \notin \{\overline{X}_n\}$. *Obviamente,* $\mu' =_{\setminus\{\overline{X}_n\}} \mu$. *Más aún,* $e_i\mu = e_i\mu'$ *para todo* $1 \leq i \leq n$, *debido a que* $\overline{X}_n$ *son variables nuevas. Por tanto, como* $\mathcal{I} \Vdash_{\mathcal{D}} e_i\mu \rightarrow t_i$, *también tenemos que* $\mathcal{I} \Vdash_{\mathcal{D}} e_i\mu' \rightarrow \mu'(X_i)$, *y entonces* $\mu' \in Sol_{\mathcal{I}}(\overline{e_n \rightarrow X_n})$. *Es más,* $\mu' \in Sol_{\mathcal{D}}(S)$ *debido a que* $\mu \in Sol_{\mathcal{D}}(S)$ *y a que* $\overline{X}_n$ *son variables nuevas en* S. *Por otra parte, como* $\models_{\mathcal{D}} p\bar{t}_n \rightarrow^{?} t\mu$, *tenemos que* $p^{\mathcal{D}}\overline{\mu'(X_n)} \rightarrow t\mu'$, *debido a que* $\overline{X}_n$ *son variables nuevas con respecto a* t.

Debido a que $p^{\mathcal{D}}$ *es radical (en el sentido especificado en la Definición 1 de la Subsección 2.2), podemos suponer que* $t\mu'$ *es* $\bot$ *o bien un patrón total. En el primer caso,* t *debe ser una variable* R *tal que* $\mu(R) = \bot$, *con lo cual* S_t *es* $S \mathbin{@} \{R \mapsto \bot\}$ *y* $\mu \in Sol_{\mathcal{I}}(S_t)$. *En el segundo caso,* S_t *es* $\blacklozenge$, $\mu' \in Sol_{\mathcal{D}}(p\overline{X}_n \rightarrow! t)$, *y con ello también* $\mu' \in Sol_{\mathcal{I}}(\overline{e_n \rightarrow X_n} \mathbin{\square} S \mathbin{@} p\overline{X}_n \rightarrow! t)$. *Por la hipótesis de que la premisa es válida en* $\mathcal{I}$, *ha de existir* $i \in I$ *tal que* $\mu' \in Sol_{\mathcal{D}}(\exists\overline{Z}_i.\ S_i)$. *Como* $\mu' =_{\setminus\{\overline{X}_n\}} \mu$, *se deduce que* $\mu \in \bigcup_{i\in I} Sol_{\mathcal{D}}(\exists\overline{X}_n, \overline{Z}_i.\ S_i)$, *y por tanto* $\mu \in Sol_{\mathcal{D}}(\bigvee_{i\in I} \exists\overline{X}_n, \overline{Z}_i.\ S_i)$.

(AR)$_{f,1}$ *Sea* $\mu \in Sol_{\mathcal{I}}(f\overline{e}_n \rightarrow t \mathbin{\square} S)$. *Entonces,* $\mu \in Sol_{\mathcal{I}}(f\overline{e}_n \rightarrow t)$ *y* $\mu \in Sol_{\mathcal{D}}(S)$. *Por definición, ha de existir una inferencia de la forma*

$$\frac{\cdots\ \mathcal{I} \Vdash_{\mathcal{D}} e_i\mu \rightarrow t_i\ \cdots}{\mathcal{I} \Vdash_{\mathcal{D}} f\ \overline{e_n}\mu \rightarrow t\mu}\ \textbf{(DF)}_{\mathcal{I}}$$

con $(f\bar{t}_n \rightarrow t\mu) \in \mathcal{I}$. *Definimos* $\mu'(X_i) = t_i$ *para cada* $1 \leq i \leq n$, *y* $\mu'(Z) = \mu(Z)$ *para todo* $Z \notin \{\overline{X}_n\}$. *Obviamente,* $\mu' =_{\setminus\{\overline{X}_n\}} \mu$. *Más aún*

- $\mathcal{I} \Vdash_{\mathcal{D}} e_i\mu' \rightarrow \mu'(X_i)$, *debido a que* $\mathcal{I} \Vdash_{\mathcal{D}} e_i\mu \rightarrow t_i$ *para cada* $1 \leq i \leq n$. *Entonces,* $\mu' \in Sol_{\mathcal{I}}(\overline{e_n \rightarrow X_n})$.
- *Como* $(f\bar{t}_n \rightarrow t\mu) \in \mathcal{I}$, *entonces* $\mathcal{I} \Vdash_{\mathcal{D}} f\bar{t}_n \rightarrow t\mu$ *(por la* Propiedad de Conservación*), o equivalentemente,* $\mathcal{I} \Vdash_{\mathcal{D}} f\mu'(X_n) \rightarrow t\mu'$ *ya que* $\overline{X}_n$ *son variables nuevas en* t. *Entonces,* $\mu' \in Sol_{\mathcal{I}}(f\overline{X}_n \rightarrow t)$.
- $\mu' \in Sol_{\mathcal{D}}(S)$, *puesto que* $\mu \in Sol_{\mathcal{D}}(S)$ *y* $\overline{X}_n$ *son variables nuevas en* S.

En consecuencia, $\mu' \in Sol_{\mathcal{I}}((\overline{e_n \rightarrow X_n} \wedge f\overline{X}_n \rightarrow t) \mathbin{\square} S)$. *Por la hipótesis de que la premisa es válida en* $\mathcal{I}$, *ha de existir* $i \in \mathcal{I}$ *tal que* $\mu' \in Sol_{\mathcal{D}}(\exists\overline{Z}_i.\ S_i)$. *Como* $\mu' =_{\setminus\{\overline{X}_n\}} \mu$, *se concluye que* $\mu \in Sol_{\mathcal{D}}(\exists\overline{X}_n, \overline{Z}_i.\ S_i)$.

(AR)$_{f,2}$ *Sea* $\mu \in Sol_{\mathcal{I}}(f\overline{e}_n\overline{a}_k \rightarrow t \mathbin{\square} S)$. *Entonces,* $\mu \in Sol_{\mathcal{I}}(f\overline{e}_n\ \overline{a}_k \rightarrow t)$ *y* $\mu \in Sol_{\mathcal{D}}(S)$. *Por definición, ha de existir una inferencia de la forma*

$$\frac{\cdots\ \mathcal{I} \Vdash_{\mathcal{D}} e_i\mu \rightarrow t_i\ \cdots\quad \mathcal{I} \Vdash_{\mathcal{D}} s\overline{a_k\mu} \rightarrow t\mu}{\mathcal{I} \Vdash_{\mathcal{D}} f\ \overline{e_n}\mu\ \overline{a_k}\mu \rightarrow t\mu}\ \textbf{(DF)}_{\mathcal{I}}$$

con $(f\bar{t}_n \rightarrow s) \in \mathcal{I}$. *Definimos* $\mu'(X_i) = t_i$ *para cada* $1 \leq i \leq n$, $\mu'(Y) = s$, *y* $\mu'(Z) = \mu(Z)$ *para todo* $Z \in \{\overline{X}_n, Y\}$. *Obviamente,* $\mu' =_{\{\overline{X}_n, Y\}} \mu$. *Más aún*

- $\mathcal{I} \Vdash_{\mathcal{D}} e_i\mu' \to \mu'(X_i)$, *porque* $\mathcal{I} \Vdash_{\mathcal{D}} e_i\mu \to t_i$ *para cada* $1 \leq i \leq n$. *Entonces,* $\mu' \in Sol_{\mathcal{I}}(\overline{e_n \to X_n})$.
- *Como* $(f\bar{t}_n \to s) \in \mathcal{I}$, *se tiene que* $\mathcal{I} \Vdash_{\mathcal{D}} f\bar{t}_n \to s$ *(de nuevo por la* Propiedad de Conservación*), o equivalentemente,* $\mathcal{I} \Vdash_{\mathcal{D}} f\mu'(X_n) \to \mu'(Y)$. *Entonces,* $\mu' \in Sol_{\mathcal{I}}(f\overline{X}_n \to Y)$.
- $\mathcal{I} \Vdash_{\mathcal{D}} s\overline{a_k\mu} \to t\mu$, *y también* $\mathcal{I} \Vdash_{\mathcal{D}} \mu'(Y)\ \overline{a_k\mu'} \to t\mu'$, *ya que* $\overline{X}_n, Y$ *son variables nuevas en* $\overline{a}_k$ *y* t. *Entonces,* $\mu' \in Sol_{\mathcal{I}}(Y\overline{a}_k \to t)$.
- $\mu' \in Sol_{\mathcal{D}}(S)$, *debido a que* $\mu \in Sol_{\mathcal{D}}(S)$ *y a que* $\overline{X}_n, Y$ *son variables nuevas en* S.

Por tanto, $\mu' \in Sol_{\mathcal{I}}((\overline{e_n \to X_n} \wedge f\overline{X}_n \to Y \wedge Y\overline{a}_k \to t) \sqcap S)$. *Por la hipótesis de que la premisa es válida en* $\mathcal{I}$, *existe* $i \in \mathcal{I}$ *para el que* $\mu' \in Sol_{\mathcal{D}}(\exists\overline{Z}_i.\ S_i)$. *Como* $\mu' =_{\setminus\{\overline{X}_n, Y\}} \mu$, *se concluye que* $\mu \in Sol_{\mathcal{D}}(\exists\overline{X}_n, Y, \overline{Z}_i.\ S_i)$.

(DF)$_f$ *Observamos en primer lugar que las premisas* $R_i[\ \overline{X_n \mapsto t_n},\ Y \mapsto t\ ]$ *se pueden escribir equivalentemente como* $R_i\theta$, *siendo* θ *la sustitución* $\{\overline{X_n \mapsto t_n}, Y \mapsto t\}$. *Supongamos que* $\mu \in Sol_{\mathcal{I}}(f\bar{t}_n \to t \sqcap S)$. *Entonces* $\mu \in Sol_{\mathcal{D}}(S)$ *y además* $\mu \in Sol_{\mathcal{I}}(f\bar{t}_n \to t)$, *que equivale a* $\theta\mu \in Sol_{\mathcal{I}}(f\overline{X}_n \to Y)$. *Además, el axioma* $(f)^-_{\mathcal{P}}$ *es válido en* $\mathcal{I}$ *debido a la hipótesis* $I \models_{\mathcal{D}} \mathcal{P}^-$. *Como* $\theta\mu \in Sol_{\mathcal{I}}(f\overline{X}_n \to Y)$, *necesariamente se tiene uno de los dos casos siguientes:*

(a) Existe $i \in I$ *tal que* $\theta\mu \in Sol_{\mathcal{I}}(\hat{R}_i)$; *o bien*

(b) $\theta\mu \in Sol_{\mathcal{D}}(\bot \to Y)$.

En el caso (a), se tiene que $\mu \in Sol_{\mathcal{I}}(\hat{R}_i\theta)$. *Considerando* $\overline{U}_i = var(R_i)\setminus\{\overline{X}_n, Y\}$, *debe existir* $\mu' =_{\setminus\overline{U}_i} \mu$ *tal que* $\mu' \in Sol_{\mathcal{I}}(R_i\theta)$. *Debido a* $\mu' =_{var(S)} \mu$, *se cumple también que* $\mu' \in Sol_{\mathcal{D}}(S)$. *Con ello,* $\mu' \in Sol_{\mathcal{I}}(R_i\theta \sqcap S)$, *que implica* $\mu' \in Sol_{\mathcal{D}}(D_i)$ *por la hipótesis de que las premisas de* **(DF**$_f$**)** *son válidas en* $\mathcal{I}$. *Finalmente, como* $\mu' =_{fvar(D_i)} \mu$, *se concluye en este caso que* $\mu \in Sol_{\mathcal{D}}(D_i)$. *En el caso (b),* $\theta\mu \in Sol_{\mathcal{D}}(\bot \to Y)$ *implica que debe cumplirse* $Y\theta\mu = \bot$. *Como* $\theta(Y) = t$, *debe ser* $t\mu = \bot$, *y esto solo es posible si* t *es una variable* R *tal que* $\mu(R) = \bot$. *En este caso,* S_t *es* $S \mathbin{@} \{R \mapsto \bot\}$ *y* $\mu \in Sol_{\mathcal{D}}(S_t)$.

□

A.4.2. $CNPC(\mathcal{D})$-admisibilidad del cálculo $RGSC(\mathcal{D})$

Demostramos a continuación que el cálculo de resolución de objetivos laxo $RGSC(\mathcal{D})$ es $CNPC(\mathcal{D})$-admisible, de acuerdo con el enunciado del Teorema 16. Gracias a este resultado quedará probada la existencia de cálculos de resolución de objetivos $CNPC(\mathcal{D})$-admisibles, lo que proporciona una base teórica a la suposición

pragmática de que los sistemas *CFLP* actuales como *Curry* o $\mathcal{TOY}$ son también $CNPC(\mathcal{D})$-admisibles. La demostración utiliza el Lema 16 (*Lema de Conjunción*), cuya demostración aparece más abajo en A.4.4.

Demostración 42 (Demostración del Teorema 16) *Según lo explicado en el planteamiento de esta demostración en la Subsección 6.4.2, basta probar que si* $(\underline{R} \wedge R') \square S \Vdash^{p}_{\mathcal{P},RGSC(\mathcal{D})} D$*, o equivalentemente,* $(\underline{R} \wedge R')$ & $\hat{S} \Vdash^{p}_{\mathcal{P},RGSC(\mathcal{D})} D$ *(con cierto espacio de búsqueda parcial de tamaño p), entonces* $\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} R \square S \Rightarrow D$*, o equivalentemente,* $\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} R$ & $S \Rightarrow D$ *(con un cierto NPT). Razonamos por inducción sobre el tamaño p del espacio de búsqueda parcial, y distinguimos casos según las diferentes reglas de transformación de objetivos del cálculo* $RGSC(\mathcal{D})$*. Estas se suponen formuladas del mismo modo que en la Subsección 6.4.2, excepto que los prefijos existenciales de los objetivos no se hacen explícitos aquí.*

Caso base*:* $p = 1$*. En este caso, el espacio de búsqueda parcial es una hoja* $(\underline{R} \wedge R') \square S \Vdash_{\mathcal{P},RGSC(\mathcal{D})} D$*, donde* $D \equiv S$*, y* $\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} R \square S \Rightarrow D$ *mediante un paso* **SF***, debido a que trivialmente* $Sol_{\mathcal{D}}(S) \subseteq Sol_{\mathcal{D}}(D)$*.*

Paso inductivo*:* $p > 1$*. En este caso, distinguimos varios subcasos de acuerdo con la regla de transformación de objetivos del cálculo* $RGSC(\mathcal{D})$ *que ha sido usada en la raíz de la computación* $(\underline{R} \wedge R') \square S \Vdash^{p}_{\mathcal{P},RGSC(\mathcal{D})} D$*.*

(DC) $(\underline{h\bar{e}_m \rightarrow h\bar{t}_m \wedge R_1} \wedge R') \square S \Vdash_{DC} (\underline{\overline{e_m \rightarrow t_m} \wedge R_1} \wedge R') \square S \Vdash^{q}_{\mathcal{P},RGSC(\mathcal{D})} D$*, con* $p = 1 + q$ *(es decir,* $q < p$*). Puesto que* $(\underline{\overline{e_m \rightarrow t_m} \wedge R_1} \wedge R') \square S \Vdash^{q}_{\mathcal{P},RGSC(\mathcal{D})} D$*, mediante la aplicación del* Lema de Conjunción*, se obtiene:*

(1) $(\underline{\overline{e_m \rightarrow t_m}} \wedge R_1 \wedge R') \square S \Vdash^{q'}_{\mathcal{P},RGSC(\mathcal{D})} \bigvee_{i \in I} \hat{S}_i$*, con* $q' \leq q < p$*. Por* hipótesis de inducción*,* $\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} \overline{e_m \rightarrow t_m} \square S \Rightarrow \bigvee_{i \in I} \hat{S}_i$*. Por aplicación de la regla* **DC** *del cálculo* $CNPC(\mathcal{D})$*:* $\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} h\bar{e}_m \rightarrow \bar{t}_m \square S \Rightarrow \bigvee_{i \in I} \hat{S}_i$*.*
(2) $\forall i \in I : (\underline{\overline{e_m \rightarrow t_m} \wedge R_1} \wedge R')$ & $\hat{S}_i \Vdash^{q'_i}_{\mathcal{P},RGSC(\mathcal{D})} D_i$*, con* $q'_i \leq q < p$ *y* $\bigvee_{i \in I} D_i = D$*. Por* hipótesis de inducción*,* $\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} R_1$ & $\hat{S}_i \Rightarrow D_i$ *para cada* $i \in I$*.*

Finalmente, mediante la aplicación de la regla **CJ** *del cálculo* $CNPC(\mathcal{D})$*:*

$$\frac{h\bar{e}_m \rightarrow h\bar{t}_m \square S \Rightarrow \bigvee_{i \in I} \hat{S}_i \quad \cdots (R_1 \ \& \ S_i) \Rightarrow D_i \cdots}{(h\bar{e}_m \rightarrow h\bar{t}_m \wedge R_1) \square S \Rightarrow \bigvee_{i \in I} D_i} \ \textbf{(CJ)}$$

En consecuencia, $\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} (h\bar{e}_m \rightarrow h\bar{t}_m \wedge R_1) \square S \Rightarrow D$*.*

(SP)$_1$ *En este caso,* $(\underline{t \rightarrow X \wedge R_1} \wedge R') \square S \Vdash_{SP_1} (\underline{R_1} \wedge R')$ & $S' \Vdash^{q}_{\mathcal{P},RGSC(\mathcal{D})} D$*, con* $q < p$ *y* $S' \equiv S \ @ \ \{X \mapsto t\}$*. Por* hipótesis de inducción*,* $\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} R_1$ & $S' \Rightarrow D$*. Finalmente, mediante la aplicación de las reglas* **(SF)** *y* **(CJ)** *del cálculo* $CNPC(\mathcal{D})$*:*

$$\frac{\dfrac{\square\ S\ @\ \{X \mapsto t\} \Rightarrow S'}{t \to X\ \square\ S \Rightarrow S'}\ (\mathbf{SF}) \qquad R_1\ \&\ S' \Rightarrow D}{(t \to X \wedge R_1)\ \square\ S \Rightarrow D}\ \mathbf{(CJ)}$$

Por tanto, $\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} (t \to X \wedge R_1)\ \square\ S \Rightarrow D$.

(SP)$_2$ *Análogo al caso de* **(SP)**$_1$.

(IM) *En este caso,* $(\underline{h\overline{e}_m \to X \wedge R_1} \wedge R')\ \square\ S \Vdash_{IM} (\underline{\overline{e_m \to X_m} \triangleq h\overline{X}_m \to X \wedge R_1} \wedge R')\ \square\ S \Vdash^q_{\mathcal{P},RGSC(\mathcal{D})} D$*, con* $q < p$*. Como* $(\underline{\overline{e_m \to X_m} \wedge h\overline{X}_m \to X \wedge R_1} \wedge R')\ \square\ S \Vdash^q_{\mathcal{P},RGSC(\mathcal{D})} D$*, aplicando el* Lema de Conjunción*, obtenemos:*

(1) $(\underline{\overline{e_m \to X_m}\ \wedge h\overline{X}_m \to X}\ \wedge R_1 \wedge R')\ \square\ S \Vdash^{q'}_{\mathcal{P},RGSC(\mathcal{D})} \bigvee_{i\in I} \hat{S}_i$*, con* $q' \leq q < p$*. Por* hipótesis de inducción*,* $\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} \overline{e_m \to X_m} \wedge h\overline{X}_m \to X\ \square\ S \Rightarrow \bigvee_{i\in I} \hat{S}_i$*. Equivalentemente,* $\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} \overline{e_m \to X_m}\ \square\ (S\ @\ \{X \mapsto h\overline{X}_m\}) \Rightarrow \bigvee_{i\in I} \hat{S}_i$*. Aplicando la regla* **IM** *del cálculo* $CNPC(\mathcal{D})$*:* $\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} h\overline{e}_m \to X\ \square\ S \Rightarrow \bigvee_{i\in I} \hat{S}_i$.

(2) $\forall i \in I : (\overline{e_m \to X_m} \wedge h\overline{X}_m \to X \wedge \underline{R_1} \wedge R')\ \&\ \hat{S}_i \Vdash^{q'_i}_{\mathcal{P},RGSC(\mathcal{D})} D_i$*, con* $q'_i \leq q < p$ *y* $\bigvee_{i\in I} D_i = D$*. Por* hipótesis de inducción*,* $\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} R_1\ \&\ S_i \Rightarrow D_i$ *para cada* $i \in I$.

Finalmente, aplicando la regla **CJ** *del cálculo* $CNPC(\mathcal{D})$*:*

$$\frac{h\overline{e}_m \to X\ \square\ S \Rightarrow \bigvee_{i\in I} \hat{S}_i \quad \cdots\ (R_1\ \&\ S_i) \Rightarrow D_i\ \cdots}{(h\overline{e}_m \to X \wedge R_1)\ \square\ S \Rightarrow \bigvee_{i\in I} D_i}\ \mathbf{(CJ)}$$

Por tanto, $\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} (h\overline{e}_m \to X \wedge R_1)\ \square\ S \Rightarrow D$.

(EL) *En este caso,* $(\underline{e \to X \wedge R_1} \wedge R')\ \square\ S \Vdash_{EL} (\underline{R_1} \wedge R')\ \square\ S \Vdash^q_{\mathcal{P},RGSC(\mathcal{D})} D$*, con* $q < p$ *y* $X \notin var((R_1 \wedge R')\ \square\ S)$*. Por* hipótesis de inducción*,* $\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} R_1\ \&\ S \Rightarrow D$*. Finalmente, aplicando las reglas* **(SF)** *y* **(CJ)** *del cálculo* $CNPC(\mathcal{D})$*:*

$$\frac{\dfrac{}{e \to X\ \square\ S \Rightarrow \hat{S}}\ (\mathbf{SF}) \qquad R_1\ \&\ S \Rightarrow D}{(e \to X \wedge R_1)\ \square\ S \Rightarrow D}\ \mathbf{(CJ)}$$

Por tanto, $\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} (e \to X \wedge R_1)\ \square\ S \Rightarrow D$.

(PF) $(\underline{p\overline{e}_n \to t \wedge R_1} \wedge R')\ \square\ S \Vdash_{PF} (\underline{\overline{e_n \to X_n} \wedge R_1} \wedge R')\ \&\ S' \Vdash^q_{\mathcal{P},RGSC(\mathcal{D})} D$*, con* $q < p$ *y* $S' = S\ @\ p\overline{X}_n \to! t$*. Como* $(\underline{\overline{e_n \to X_n} \wedge R_1} \wedge R')\ \&\ S' \Vdash^q_{\mathcal{P},RGSC(\mathcal{D})} D$*, aplicando el* Lema de Conjunción*, obtenemos:*

(1) $(\overline{\underline{e_n \to X_n}} \wedge R_1 \wedge R') \;\&\; S' \Vdash^{q'}_{\mathcal{P},RGSC(\mathcal{D})} \bigvee_{i\in I} \hat{S}_i$*, con* $q' \leq q < p$*. Por* hipótesis de inducción, $\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} \overline{e_n \to X_n} \;\&\; S' \Rightarrow \bigvee_{i\in I} \hat{S}_i$*. Equivalentemente,* $\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} \overline{e_n \to X_n} \;\square\; (S \;@\; p\overline{X}_n \to!\; t) \Rightarrow \bigvee_{i\in I} \hat{S}_i$*. Mediante la aplicación de la regla* $\mathbf{AR}_p$ *del cálculo* $CNPC(\mathcal{D})$*:* $\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} p\overline{e}_n \to t \;\square\; S \Rightarrow \bigvee_{i\in I} \hat{S}_i$*.*

(2) $\forall i \in I : (\overline{e_n \to X_n} \wedge \underline{R_1} \wedge R') \;\&\; \hat{S}_i \Vdash^{q'_i}_{\mathcal{P},RGSC(\mathcal{D})} D_i$*, con* $q'_i \leq q < p$ *y* $\bigvee_{i\in I} D_i = D$*. Por* hipótesis de inducción, $\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} R_1 \;\&\; S_i \Rightarrow D_i$ *para cada* $i \in I$*.*

Finalmente, aplicando la regla **CJ** *del cálculo* $CNPC(\mathcal{D})$*:*

$$\frac{p\overline{e}_n \;\to\; t \;\square\; S \;\Rightarrow\; \bigvee_{i\in I} \hat{S}_i \quad \cdots\; (R_1 \;\&\; \hat{S}_i) \;\Rightarrow\; D_i \;\cdots}{(p\overline{e}_n \;\to\; t \;\wedge\; R_1) \;\square\; S \;\Rightarrow\; \bigvee_{i\in I} D_i}\;\mathbf{(CJ)}$$

En consecuencia, $\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} (p\overline{e}_n \to t \wedge R_1) \;\square\; S \Rightarrow D$*.*

(SC) *En este caso,* $(\underline{R} \wedge R') \;\square\; S \Vdash_{SC} \bigvee_{i\in I} (\underline{R} \wedge R') \;\&\; \hat{S}_i \Vdash^{q_i}_{\mathcal{P},RGSC(\mathcal{D})} \bigvee_{i\in I} D_i$*, con* $q_i < p$ *para cada* $i \in I$*,* $D = \bigvee_{i\in I} D_i$ *y* $Sol_{\mathcal{D}}(S) = \bigcup_{i\in I} Sol_{\mathcal{D}}(\hat{S}_i)$*. Por* hipótesis de inducción, $\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} R \;\&\; S_i \Rightarrow D_i$ *para cada* $i \in I$*. Finalmente, mediante la aplicación de las reglas* **(SF)** *y* **(CJ)** *del cálculo* $CNPC(\mathcal{D})$*:*

$$\frac{\dfrac{}{\square\; S \;\Rightarrow\; \bigvee_{i\in I} \hat{S}_i}\;\mathbf{(SF)} \quad \cdots\; R \;\&\; S_i \;\Rightarrow\; D_i \;\cdots}{R \;\square\; S \;\Rightarrow\; \bigvee_{i\in I} D_i}\;\mathbf{(CJ)}$$

Por tanto, $\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} R \;\square\; S \Rightarrow D$*.*

(CF) *En este caso,* $(h\overline{e}_p \to h'\overline{t}_q \;\wedge R_1 \wedge R') \;\square\; S \Vdash_{CF} \blacksquare$*, con* $h \neq h'$ *o* $p \neq q$*. Aplicando las reglas* **(TS)**, **(SF)** *y* **(CJ)** *del cálculo* $CNPC(\mathcal{D})$*:*

$$\frac{\dfrac{}{(h\overline{e}_p \;\to\; h'\overline{t}_q) \;\square\; S \;\Rightarrow\; \blacklozenge}\;\mathbf{(TS)} \quad \dfrac{}{R_1 \;\&\; \blacklozenge \;\Rightarrow\; \blacklozenge}\;\mathbf{(SF)}}{(h\overline{e}_p \;\to\; h'\overline{t}_q \;\wedge\; R_1) \;\square\; S \;\Rightarrow\; \blacklozenge}\;\mathbf{(CJ)}$$

Por tanto, $\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} (h\overline{e}_p \to h'\overline{t}_q \wedge R_1) \;\square\; S \Rightarrow \blacklozenge$*.*

(FC) *En este caso,* $(\underline{R} \wedge R') \;\square\; S \Vdash_{FS} \blacksquare$*, con* $Sol_{\mathcal{D}}(S) = \emptyset$*. Puesto que* $Sol_{\mathcal{D}}(\blacklozenge) = \emptyset$*, aplicando las reglas* **(SF)** *y* **(CJ)** *del cálculo* $CNPC(\mathcal{D})$*:*

$$\frac{\dfrac{}{\square\; S \;\Rightarrow\; \blacklozenge}\;\mathbf{(SF)} \quad \dfrac{}{R \;\&\; \blacklozenge \;\Rightarrow\; \blacklozenge}\;\mathbf{(SF)}}{R \;\square\; S \;\Rightarrow\; \blacklozenge}\;\mathbf{(CJ)}$$

En consecuencia, $\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} R \;\square\; S \Rightarrow \blacklozenge$*.*

(DF$_f$) *Presentamos la demostración correspondiente al segundo caso de esta regla, es decir,* **(DF$_{f,2}$)***. El primer caso,* **(DF$_{f,1}$)***, es similar e incluso más sencillo.*

$(\underline{f\overline{e}_n\overline{a}_k \to t \ \wedge R_1} \wedge R') \sqcap S \Vdash_{DF} \bigvee_{i\in I} ((\overline{\underline{e_n \to X_n} \wedge Y\overline{a}_k \to t \wedge R'_i \wedge R_1} \wedge R') \sqcap S) \Vdash^{q_i}_{\mathcal{P},RGSC(\mathcal{D})} \bigvee_{i\in I} D_i$, *con* $q_i < p$ *para cada* $i \in I$, $D = \bigvee_{i\in I} D_i$ *y* $(f\overline{X}_n \to Y \Rightarrow \bigvee_{i\in I} \hat{R}'_i) \in_{var} \mathcal{P}^-$. *Debido a que tenemos el espacio de búsqueda parcial finito* $((\overline{\underline{e_n \to X_n} \wedge Y\overline{a}_k \to t \wedge R'_i \wedge R_1} \wedge R') \sqcap S) \Vdash^{q_i}_{\mathcal{P},RGSC(\mathcal{D})} D_i$ *para cada* $i \in I$, *aplicando el* Lema de Conjunción, *obtenemos:*

(1)$_i$ $(\overline{\underline{e_n \to X_n} \wedge Y\overline{a}_k \to t \wedge R'_i} \wedge R_1 \wedge R') \sqcap S \Vdash^{q'_i}_{\mathcal{P},RGSC(\mathcal{D})} \bigvee_l \hat{S}_{il}$, *con* $q'_i \leq q_i < p$ *para todo* $i \in I$.

(2)$_i$ $\forall\, l : (\overline{e_n \to X_n} \wedge Y\overline{a}_k \to t \wedge R'_i \wedge \underline{R_1} \wedge R') \ \&\ \hat{S}_{il} \Vdash^{q'_{ij}}_{\mathcal{P},RGSC(\mathcal{D})} D_{il}$, *con* $q'_{ij} \leq q < p$, *y* $D_i = \bigvee_l D_{il}$ *para todo* $i \in I$. *Por* hipótesis de inducción, $\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} R_1 \ \&\ S_{il} \Rightarrow D_{il}$ *para todo* $i \in I$ *y para todo* l.

Aplicamos de nuevo el Lema de Conjunción *al espacio de búsqueda parcial finito (1)*$_i$ *para cada* $i \in I$*:*

(1.1)$_i$ $(\overline{e_n \to X_n} \wedge Y\overline{a}_k \to t \wedge \underline{R'_i} \wedge R_1 \wedge R') \sqcap S \Vdash^{q''_i}_{\mathcal{P},RGSC(\mathcal{D})} \bigvee_j \hat{S}_{ij}$, *con* $q''_i \leq q'_i \leq q_i < p$ *para todo* $i \in I$. *Por* hipótesis de inducción, $\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} R'_i \sqcap S \Rightarrow \bigvee_j \hat{S}_{ij}$ *para cualquier* $i \in I$.

(1.2)$_i$ $\forall\, j : (\overline{\underline{e_n \to X_n} \wedge Y\overline{a}_k \to t} \wedge R'_i \wedge R_1 \wedge R') \ \&\ \hat{S}_{ij} \Vdash^{q''_{ij}}_{\mathcal{P},RGSC(\mathcal{D})} D_{ij}$, *con* $q''_{ij} \leq q'_i \leq q_i < p$ *y* $\bigvee_j D_{ij} = \bigvee_l \hat{S}_{il}$ *para todo* $i \in I$. *Aplicando la* hipótesis de inducción, $\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} (\overline{e_n \to X_n} \wedge Y\overline{a}_k \to t) \ \&\ S_{ij} \Rightarrow D_{ij}$.

Combinando las derivaciones que se obtienen a partir de (2)$_i$*, (1.1)*$_i$ *y (1.2)*$_i$ *por medio de las reglas de inferencia* **(CJ)**, $\mathbf{(AR)}_f$ *y* $\mathbf{(DF)}_f$ *del cálculo* $CNPC(\mathcal{D})$, *obtenemos las siguientes derivaciones:*

$$\frac{\cdots\ R'_i \sqcap S \Rightarrow \bigvee_j \hat{S}_{ij}\ \cdots}{f\overline{X}_n \to Y \sqcap S \Rightarrow (S \mathbin{@} \{Y \mapsto \bot\}) \vee (\bigvee_{ij} \hat{S}_{ij})}\ \mathbf{(DF)}_f$$

$\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} (\overline{e_n \to X_n} \wedge Y\overline{a}_k \to t) \ \&\ (S \mathbin{@} \{Y \mapsto \bot\}) \Rightarrow \blacklozenge$ *porque* $\bot\overline{a}_k \to t$ *no es derivable en* $CNPC(\mathcal{D})$ *si* $k > 0$.

$\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} (\overline{e_n \to X_n} \wedge Y\overline{a}_k \to t) \ \&\ S_{ij} \Rightarrow D_{ij}$.

Aplicando la regla **(CJ)**, *obtenemos:*

$$\frac{(\overline{e_n \to X_n} \wedge f\overline{X}_n \to Y \wedge Y\overline{a}_k \to t) \sqcap S \Rightarrow \bigvee_{ij} D_{ij} = \bigvee_{il} \hat{S}_{il}}{f\overline{e}_n\overline{a}_k \to t \sqcap S \Rightarrow \bigvee_{il} D_{il}}\ \mathbf{(AR)}_f$$

$\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} R_1 \;\&\; S_{il} \Rightarrow D_{il}$ *para todo* $i \in I$ *y para todo* l.

Finalmente, aplicando de nuevo la regla **(CJ)***:*

$\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} (f\overline{e}_n\overline{a}_k \rightarrow t \wedge R_1) \sqcap S \Rightarrow \bigvee_{il} D_{il}$.

Como $\bigvee_{il} D_{il} = \bigvee_{i \in I} D_i = D$*, concluimos que:*

$\mathcal{P}^- \vdash_{CNPC(\mathcal{D})} (f\overline{e}_n\overline{a}_k \rightarrow t \wedge R_1) \sqcap S \Rightarrow D$.

□

A.4.3. Lemas auxiliares del Lema de Conjunción

La demostración del *Lema de Conjunción* se basa en dos lemas técnicos sobre el cálculo $RGSC(\mathcal{D})$ que a continuación enunciamos y demostramos.

Lema 19 (*Lema Técnico 1*) *Supongamos que se verifican las siguientes hipótesis:*

(a) $(\underline{\widehat{R_1 \wedge R_2}}) \;\&\; \hat{S} \Vdash_{\mathbf{RL}} (\widehat{R'_1 \wedge R_2}) \;\&\; \hat{S}$ *(objetivo no fallido) es un paso de cómputo que utiliza una cierta regla* **(RL)** *de* $RGSC(\mathcal{D})$*, aplicada a la parte* R_1.

(b) $(\widehat{R'_1 \wedge \underline{R_2}}) \;\&\; \hat{S}' \Vdash^*_{\mathcal{P},\, RGSC(\mathcal{D})} (\widehat{R'_1 \wedge R''_2}) \;\&\; \hat{S}''$ *representa* n *pasos de cómputo para cierto* $n \geq 0$*, aplicados a descendientes de la parte* R_2.

Entonces se tiene también:

(c) $(\underline{\widehat{R_1 \wedge R_2}}) \;\&\; \hat{S}'' \Vdash_{\mathbf{RL'}} (\widehat{R'_1 \wedge R_2}) \;\&\; \hat{S}''$*, donde la regla* **(RL')** *no siempre es la propia regla* **(RL)**.

Demostración 43 *Observamos en primer lugar que este lema también se puede aplicar intercambiando los papeles de* R_1 *y* R_2*, porque en la Subsección 6.4.2 hemos convenido en interpretar la estructura sintáctica de los objetivos módulo la conmutatividad y asociatividad de* $\wedge$.

Para demostrar el lema, supongamos ciertas las hipótesis (a) y (b). En estas condiciones, se tendrá $S = \Pi \sqcap \sigma$, $S' = \Pi' \sqcap \sigma\sigma'$ *y* $S'' = \Pi'' \sqcap \sigma\sigma'\sigma''$*, siendo* σ *la sustitución idempotente incluida en el objetivo de partida y* σ', σ'' *las sustituciones idempotentes (quizás vacías) calculadas en el paso de cómputo (a) y la sucesión de pasos de cómputo (b), respectivamente. Debido a la corrección de las reglas de transformación de objetivos del cálculo* $RGSC(\mathcal{D})$*, podemos suponer que* $Sol_{\mathcal{D}}(\hat{S}'') \subseteq Sol_{\mathcal{D}}(\hat{S}') \subseteq Sol_{\mathcal{D}}(\hat{S})$*. Además, debido al comportamiento del operador* & *(véase la Definición 32 en la Sección 6.4), podemos hacer las siguientes suposiciones acerca de la forma de los objetivos que aparecen a la izquierda del signo* $\Vdash$ *en (a), (b) y (c), respectivamente:*

(a') $(\widehat{\underline{R_1} \wedge R_2})$ & $\hat{S}$ *es* $\exists \overline{U}.\ ((\underline{R_1} \wedge R_2) \sqcap S)$ *(sin pérdida de generalidad podemos suponer que la parte no resuelta* $\underline{R_1} \wedge R_2$ *ya está afectada por* σ*).*

(b') $(\widehat{R_1' \wedge \underline{R_2}})$ & $\hat{S}'$ *es* $\exists \overline{U}'.\ ((R_1' \wedge \underline{R_2})\sigma' \sqcap S')$.

(c') $(\widehat{\underline{R_1} \wedge R_2})$ & $\hat{S}''$ *es* $\exists \overline{U}''.\ ((\underline{R_1} \wedge R_2)\sigma'\sigma'' \sqcap S'')$.

El resto de la demostración es un razonamiento por distinción de casos, según la regla **(RL)** *aplicada en el paso (a) que se tiene por hipótesis. En cada caso, el objetivo (a') se corresponderá con el objetivo que aparece a la izquierda del signo* $\vdash\!\!\!\vdash$ *en la formulación de* **(RL)** *presentada en la Subsección 6.4.2. En aquellos casos en los que* **(RL)** *requiera la selección de un átomo* α*, tendremos que* α *está en* R_1*, con lo cual* $\alpha\sigma'\sigma''$ *estará en* $R_1\sigma'\sigma''$*, y argumentaremos como elegir* **RL'** *de manera que se pueda aplicar seleccionando* $\alpha\sigma'\sigma''$*.*

- **(RL)** *es* **(DC)***: En este caso,* α *es de la forma* $h\overline{e}_m \to h\overline{t}_m$ *o* $t \to t$*. En ambos subcasos,* **(DC)** *es aplicable a* $\alpha\sigma'\sigma''$*, y dicha aplicación proporciona (c).*

- **(RL)** *es* **(SP)***: En este caso,* α *es de la forma* $t \to X$ *o* $X \to t$*. En ambos subcasos, (a) calcula* $\sigma' = \{X \mapsto t\}$*, con lo cual* $\alpha\sigma'\sigma''$ *es de la forma* $t\sigma'' \to t\sigma''$*. Entonces* **(DC)** *es aplicable a* $\alpha\sigma'\sigma''$*, y dicha aplicación proporciona (c).*

- **(RL)** *es* **(IM)***: En este caso,* α *es de la forma* $h\overline{e}_m \to X$ *y la sustitución* σ' *calculada por (a) es vacía, con lo cual* $\alpha\sigma'\sigma''$ *es de la forma* $(h\overline{e}_m \to X)\sigma''$*. Puesto que* σ'' *ha sido calculada por (b), podemos suponer que o bien* σ'' *no vincula* X*, o bien* $\sigma''(X)$ *es de la forma* $h\overline{t}_m$*. En el primer subcaso, podemos obtener (c) aplicando* **(IM)** *a* $\alpha\sigma'\sigma''$*. En el segundo subcaso, (c) se obtiene aplicando* **(DC)** *a* $\alpha\sigma'\sigma''$*.*

- **(RL)** *es* **(EL)***: En este caso* α *es de la forma* $e \to X$ *y (a) es posible porque la variable* X *no aparece en ningún otro lugar en* $(\underline{R_1} \wedge R_2) \sqcap S$*. En estas condiciones, el cómputo (b) no puede vincular* X*, y como* σ' *es vacía en este caso,* $\alpha\sigma'\sigma''$ *es de la forma* $e\sigma'' \to X$*. Además,* X *no aparece en ningún otro lugar en* $(\underline{R_1} \wedge R_2)\sigma'\sigma'' \sqcap S''$ *porque los pasos del cómputo (b) solo introducen variables nuevas. Por tanto, (c) se puede obtener aplicando* **(EL)** *a* $\alpha\sigma'\sigma''$*.*

- **(RL)** *es* $\mathbf{(PF)}_p$*: En este caso* α *es de la forma* $p\overline{e}_n \to^? t$*, donde* $p \in PF^n$ *y* $\to^?$ *puede ser* $\to$ *o* $\to!$*. Además, si* $\to^?$ *es* $\to$ *y* t *es una variable* Z*, entonces se requiere que* $Z \in dvar_{\mathcal{D}}((\underline{R_1} \wedge R_2) \sqcap S)$*. El paso (a) calcula una sustitución vacía* σ' *y construye* R_1' *y* S' *del siguiente modo:*

  - R_1' *resulta de eliminar* $p\overline{e}_n \to^? t$ *de* R_1 *y añadir* $\overline{e_n \to X_n}$*, siendo* $\overline{X}_n$ *variables nuevas.*
  - S_1' *es* S_1 @ $p\overline{X}_n \to! t$*.*

En estas condiciones, el cómputo (b) no puede vincular $\overline{X}_n$*, y* $R_1'\sigma'\sigma''$ *incluye* $\alpha\sigma'\sigma''$ *que es de la forma* $(p\overline{e}_n \rightarrow^? t)\sigma''$*. En el supuesto de que* $\rightarrow^?$ *sea* $\rightarrow$ *y* t *sea una variable* Z*, tendremos que si* $\sigma'\sigma'' = \sigma''$ *no ha vinculado a* Z *se cumplirá que* $Z \in dvar_{\mathcal{D}}((\underline{R_1} \wedge R_2)\sigma'\sigma'' \square S'')$ *debido a* $Z \in dvar_{\mathcal{D}}((\underline{R_1} \wedge R_2) \square S)$ *y a la relación entre* $\hat{S}$ *y* $\hat{S}''$ *dada por (a) y (b).*

Por tanto, $\mathbf{(PF)}_p$ *es aplicable a* $\widehat{(\underline{R_1} \wedge R_2)}$ & $\hat{S}''$ *eligiendo el átomo* $\alpha'\sigma'\sigma''$*. Al realizar este paso de cómputo no se calcula ningún vínculo y se efectúan las siguientes transformaciones:*

- $\alpha'\sigma'\sigma'' = (p\overline{e}_n \rightarrow^? t)\sigma''$ *se elimina de* $R_1\sigma'\sigma''$ *y en su lugar se introducen nuevas producciones* $\overline{e_n \rightarrow Y_n}$*, siendo* $\overline{Y}_n$ *nuevas variables.*
- *La restricción* $p\,\overline{Y}_n \rightarrow! t\sigma''$ *se añade a* S''*.*

Teniendo en cuenta que S'' *ya incluía la restricción* $p\ \overline{X}_n \rightarrow! t$ *y que las variables* $\overline{X}_n$*,* $\overline{Y}_n$ *van afectadas de una cuantificación existencial, el objetivo resultante del paso (c) es equivalente (aunque no sintácticamente idéntico) a* $\widehat{(R_1' \wedge R_2)}$ & $\hat{S}''$*.*

- **(RL)** *es* $\mathbf{(DF)}_f$*: En este caso* α *es de la forma* $f\overline{e}_n \rightarrow t$ *o bien* $f\overline{e}_n\overline{a}_k \rightarrow t$ *con* $k > 0$*, donde* $f \in DF^n$*. Además, si* t *es una variable* Z*, se requiere que* $Z \in dvar_{\mathcal{D}}((\underline{R_1} \wedge R_2) \square S)$*. El paso (a) calcula una sustitución vacía* σ' *y modifica* R_1 *añadiendo nuevos átomos y nuevas variables existenciales, pero sin modificar* S*. El cómputo (b) no puede afectar a las nuevas variables existenciales introducidas por (a). El paso (c) se puede lograr aplicando* $\mathbf{(DF)}_f$ *a* $\alpha'\sigma'\sigma''$*. Si* t *era una variable* Z *que no ha sido vinculada por* $\sigma'\sigma'' = \sigma''$*, la condición de demanda necesaria para el paso (c) se puede argumentar de la misma manera que en el caso de la regla* $\mathbf{(PF)}_p$*.*

- **(RL)** *es* **(SC)***: En este caso,* R_1' *es idéntico a* R_1*, y el paso de cómputo requerido para (c) es de la forma* $\widehat{(R_1 \wedge R_2)}$ & $\hat{S}'' \vdash_{\mathbf{RL'}} \widehat{(R_1 \wedge R_2)}$ & $\hat{S}''$*, que se obtiene trivialmente usando la propia* **(SC)** *como* **(RL')***.*

- **(RL)** *es* **(CF)** *o* **(FC)***: Estos dos casos no son posibles debido a la hipótesis de que el paso (a) no conduce a un objetivo fallido.*

□

Como se ha dicho en la Subsección 6.4.2, interpretamos la notación "$(R_1 \wedge R_2)$" módulo la asociatividad y conmutatividad de $\wedge$. Por esta razón, el Lema 19 también se puede aplicar intercambiando los papeles de R_1 y R_2.

Lema 20 (*Lema Técnico 2*) *Supongamos que se verifican las siguiente hipótesis:*

(a) $(\widehat{\underline{R_1} \wedge R_2})$ & $\hat{S} \Vdash_{\mathbf{RL}} (\widehat{R'_1 \wedge R_2})$ & $\hat{S}'$ *es un paso de cómputo que utiliza una cierta regla* **(RL)** *de* $RGSC(\mathcal{D})$*, aplicada a la parte* R_1*.*

(b) $(\widehat{\underline{R_1} \wedge R_2})$ & $\hat{S} \Vdash^{*}_{\mathcal{P},\, RGSC(\mathcal{D})} (\widehat{R''_1 \wedge R_2})$ & $\hat{S}''$ *representa* n *pasos de cómputo para cierto* $n \geq 0$*, aplicados a descendientes de la parte* R'_1*.*

(c) $(\widehat{R'_1 \wedge \underline{R_2}})$ & $\hat{S}'' \Vdash^{p}_{\mathcal{P},\, RGSC(\mathcal{D})} D$*.*

Entonces se tiene también:

(d) $(\widehat{R_1 \wedge \underline{R_2}})$ & $\hat{S}'' \Vdash^{p}_{\mathcal{P},\, RGSC(\mathcal{D})} D$*.*

Demostración 44 *Observamos en primer lugar que este lema también se puede aplicar intercambiando el papel de* R_1 *y* R_2*, por las mismas razones ya mencionadas en relación con el Lema 19.*

Para demostrar el lema observamos que la hipótesis (c) garantiza la existencia de un espacio de búsqueda parcialmente desarrollado de tamaño p*, con raíz* $(\widehat{R'_1 \wedge \underline{R_2}})$ & $\hat{S}''$*, construido utilizando una función de selección que elige en el objetivo de cada nodo un átomo descendiente de la parte* R_2 *del objetivo raíz, y tal que la disyunción de las formas resueltas de las hojas es* D*. El lema afirma que se puede construir otro espacio de búsqueda parcialmente desarrollado semejante a este, pero ahora utilizando* $(\widehat{R_1 \wedge \underline{R_2}})$ & $\hat{S}''$ *como objetivo raíz. La única diferencia entre los dos objetivos raíz se localiza en la diferencia entre las partes* R'_1 *y* R_1*, formadas por átomos cuyos descendientes no está permitido seleccionar. Por ello, el espacio de búsqueda requerido en (d) se puede construir con la misma selección de átomos y aplicación de reglas de* $RGSC(\mathcal{D})$ *usada para la construcción del espacio de búsqueda dado por la hipótesis (c). Las hipótesis (a) y (b) se han incluido también en el enunciado del lema para clarificar el contexto en el que este se utiliza dentro de la demostración del Lema 16 (Lema de Conjunción).*

□

Al igual que el Lema 19 y por los mismos motivos, el Lema 20 también se puede aplicar intercambiando los papeles de R_1 y R_2.

A.4.4. Lema de Conjunción

Finalizamos esta última sección del apéndice con la demostración del *Lema de Conjunción*, el cual ha sido utilizado a su vez para probar el Teorema 16.

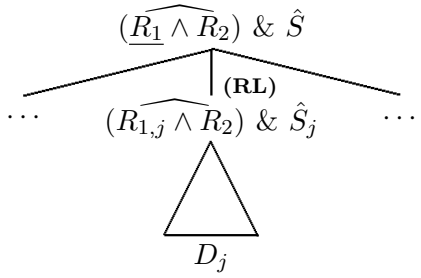
Demostración 45 (Demostración del Lema 16) *Razonamos por inducción sobre el tamaño* $p \geq 1$ *del espacio de búsqueda parcial* $(\widehat{R_1 \wedge R_2})$ & $\hat{S} \Vdash^{p}_{\mathcal{P},RGSC(\mathcal{D})} D$*,*

definido como el número total de nodos del árbol que lo representa.

Caso base*: $p = 1$. En este caso, $D \equiv \hat{S}$, y entonces $(\widehat{\underline{R_1} \wedge R_2})$ & $\hat{S} \Vdash^{1}_{\mathcal{P},RGSC(\mathcal{D})} \hat{S}$ con $q = 1 \leq p$, $(\widehat{R_1 \wedge \underline{R_2}})$ & $\hat{S} \Vdash^{1}_{\mathcal{P},RGSC(\mathcal{D})} \hat{S}$ con $q_1 \leq p$, y $\bigvee_{i \in I} D_i = \hat{S} \equiv D$.*

Paso inductivo*: distinguimos tres casos de acuerdo con la posición del átomo seleccionado por la regla* **(RL)** *del cálculo $RGSC(\mathcal{D})$ usada en la raíz del espacio de búsqueda parcial $(\widehat{R_1 \wedge R_2})$ & $\hat{S} \Vdash^{p}_{\mathcal{P},RGSC(\mathcal{D})} D$.*

- *La regla* **(RL)** *se aplica seleccionando un átomo de R_1. Entonces, tenemos la siguiente situación:*



correspondiendo a la computación $(\widehat{\underline{R_1} \wedge R_2})$ & $\hat{S} \Vdash_{\mathbf{RL}} \bigvee_{j \in J} (\widehat{R_{1,j} \wedge R_2})$ & $\hat{S}_j \Vdash^{p_j}_{\mathcal{P},RGSC(\mathcal{D})} \bigvee_{j \in J} D_j$ con $p_j < p$ para cada $j \in J$, y $D = \bigvee_{j \in J} D_j$. Como $(\widehat{R_{1,j} \wedge R_2})$ & $\hat{S}_j \Vdash^{p_j}_{\mathcal{P},RGSC(\mathcal{D})} D_j$ para todo $j \in J$, aplicando la hipótesis de inducción*, obtenemos:*

(1)$_j$ $(\widehat{\underline{R_{1,j}} \wedge R_2})$ & $\hat{S}_j \Vdash^{q'_j}_{\mathcal{P},RGSC(\mathcal{D})} \bigvee_{i \in I_j} \hat{S}_{j,i}$, con $q'_j \leq p_j$.

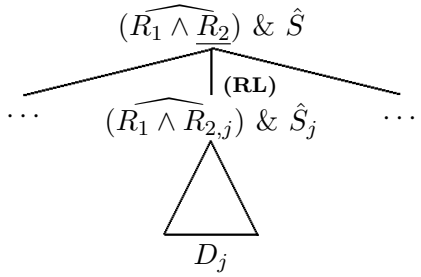
(2)$_j$ $\forall\, i \in I_j : (\widehat{R_{1,j} \wedge \underline{R_2}})$ & $\hat{S}_{j,i} \Vdash^{q'_{j,i}}_{\mathcal{P},RGSC(\mathcal{D})} D_{j,i}$, con $q'_{j,i} \leq p_j$ y $\bigvee_{i \in I_j} D_{j,i} = D_j$.

Entonces, podemos componer las siguientes computaciones:

(1) $(\widehat{\underline{R_1} \wedge R_2})$ & $\hat{S} \Vdash_{\mathbf{RL}} \bigvee_{j \in J} (\widehat{\underline{R_{1,j}} \wedge R_2})$ & $\hat{S}_j \Vdash^{q'_j}_{\mathcal{P},RGSC(\mathcal{D})} \bigvee_{j \in J} (\bigvee_{i \in I_j} \hat{S}_{j,i})$ aplicando (1)$_j$, con tamaño $1 + \sum_{j \in J} q'_j \leq 1 + \sum_{j \in J} p_j = p$.

(2) $\forall\, j \in J$ y $\forall\, i \in I_j : (\widehat{R_1 \wedge \underline{R_2}})$ & $\hat{S}_{j,i} \Vdash^{q'_{j,i}}_{\mathcal{P},RGSC(\mathcal{D})} D_{j,i}$ aplicando (2)$_j$ y el Lema Técnico 2*, con tamaño $q'_{j,i} \leq p_j < p$ y $\bigvee_{j \in J} (\bigvee_{i \in I_j} D_{j,i}) = \bigvee_{j \in J} D_j = D$.*

- *La regla* **(RL)** *se aplica seleccionando un átomo de R_2. Entonces, tenemos la siguiente situación:*



correspondiendo a la computación $(\widehat{R_1 \wedge \underline{R_2}})$ & $\hat{S} \vdash_{\mathbf{RL}} \bigvee_{j\in J} (\widehat{R_1 \wedge R_{2,j}})$ & $\hat{S}_j \vdash^{p_j}_{\mathcal{P},RGSC(\mathcal{D})} \bigvee_{j\in J} D_j$ *con* $p_j < p$ *para cada* $j \in J$, *y* $D = \bigvee_{j\in J} D_j$. *Puesto que* $(\widehat{R_1 \wedge R_{2,j}})$ & $\hat{S}_j \vdash^{p_j}_{\mathcal{P},RGSC(\mathcal{D})} D_j$ *para todo* $j \in J$, *aplicando la* hipótesis de inducción, *obtenemos:*

(1)$_j$ $(\widehat{\underline{R_1} \wedge R_{2,j}})$ & $\hat{S}_j \vdash^{q'_j}_{\mathcal{P},RGSC(\mathcal{D})} \bigvee_{i\in I_j} \hat{S}_{j,i}$, *con* $q'_j \leq p_j$.

(2)$_j$ $\forall\ i \in I_j : (\widehat{R_1 \wedge \underline{R_{2,j}}})$ & $\hat{S}_{j,i} \vdash^{q'_{j,i}}_{\mathcal{P},RGSC(\mathcal{D})} D_{j,i}$, *con* $q'_{j,i} \leq p_j$ *y* $\bigvee_{i\in I_j} D_{j,i} = D_j$.

Entonces, podemos componer las siguientes computaciones:

(1) $(\widehat{\underline{R_1} \wedge R_2})$ & $\hat{S} \vdash_{\mathbf{SC}} \bigvee_{j\in J} (\widehat{\underline{R_1} \wedge R_2})$ & $\hat{S}_j$, *ya que* $Sol_{\mathcal{D}}(\hat{S}) = \bigcup_{j\in J} Sol_{\mathcal{D}}(\hat{S}_j)$ *se puede suponer cierto porque* $(\widehat{R_1 \wedge R_{2,j}})$ & $\hat{S}_j$ *(*$j \in J$*) son todos los objetivos posibles que se derivan aplicando* **(RL)** *a un mismo átomo seleccionado del objetivo* $(\widehat{R_1 \wedge R_2})$ & $\hat{S}$. *Más aún,* $(\widehat{\underline{R_1} \wedge R_2})$ & $\hat{S}_j \vdash^{q'_j}_{\mathcal{P},RGSC(\mathcal{D})} \bigvee_{i\in I_j} \hat{S}_{j,i}$ *para cada* $j \in J$, *aplicando (1)*$_j$ *y el* Lema Técnico 2, *con tamaño* $1 + \sum_{j\in J} q'_j \leq 1 + \sum_{j\in J} p_j \leq p$.

(2) $\forall j \in J$ *y* $\forall i \in I_j : (\widehat{R_1 \wedge \underline{R_2}})$ & $\hat{S}_{j,i} \vdash_{\mathbf{RL'}} (\widehat{R_1 \wedge \underline{R_{2,j}}})$ & $\hat{S}_{j,i} \vdash^{q'_{j,i}}_{\mathcal{P},RGSC(\mathcal{D})} D_{j,i}$, *con tamaño* $1 + q'_{j,i} \leq 1 + p_j \leq p$. *En cada uno de estos cómputos, el primer paso resulta de aplicar el* Lema Técnico 1 *a* $(\widehat{R_1 \wedge \underline{R_2}})$ & $\hat{S} \vdash_{\mathbf{RL}} (\widehat{R_1 \wedge \underline{R_{2,j}}})$ & $\hat{S}_j$ *y el resto del cómputo se tiene por (2)*$_j$.

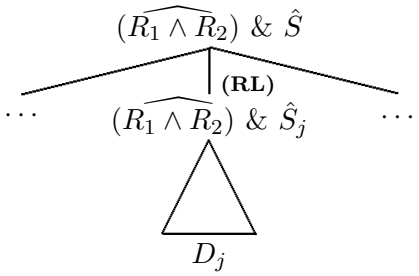
Obsérvese que si $J = \emptyset$, *y por tanto* $(\widehat{R_1 \wedge \underline{R_2}})$ & $\hat{S} \vdash_{\mathbf{RL}} \blacksquare$, *entonces no se puede aplicar el* Lema Técnico 1, *pero la tesis requerida por el* Lema de Conjunción *se demuestra muy sencillamente considerando:*

(1) $(\widehat{\underline{R_1} \wedge R_2})$ & $\hat{S} \vdash^{1}_{\mathcal{P},\ RGSC(\mathcal{D})} \hat{S}$ *(*I *con* $|I| = 1$*)*

(2) $(\widehat{R_1 \wedge \underline{R_2}})$ & $\hat{S} \vdash_{\mathcal{P},\ RGSC(\mathcal{D})} \blacksquare$ *(usando* **(RL)***)*

(3) $\bigvee_{i\in I} \hat{S}_i = \blacksquare$

- *La regla* **(RL)** *se aplica sin seleccionar ningún átomo del objetivo, y por tanto no es aplicada ni a* R_1 *ni a* R_2*. Entonces, tenemos la siguiente situación:*



correspondiendo a la computación $(\widehat{R_1 \wedge R_2})$ & $\hat{S} \vdash_{\mathbf{RL}} \bigvee_{j\in J} (\widehat{R_1 \wedge R_2})$ & $\hat{S}_j \vdash^{p_j}_{\mathcal{P},RGSC(\mathcal{D})} \bigvee_{j\in J} D_j$ *con* $p_j < p$ *para cada* $j \in J$*, y* $D = \bigvee_{j\in J} D_j$*. Como* $(\widehat{R_1 \wedge R_2})$ & $\hat{S}_j \vdash^{p_j}_{\mathcal{P},RGSC(\mathcal{D})} D_j$ *para todo* $j \in J$*, aplicando la* hipótesis de inducción*, obtenemos:*

(1)$_j$ $(\widehat{\underline{R_1} \wedge R_2})$ & $\hat{S}_j \vdash^{q'_j}_{\mathcal{P},RGSC(\mathcal{D})} \bigvee_{i\in I_j} \hat{S}_{j,i}$*, con* $q'_j \leq p_j$*.*

(2)$_j$ $\forall\, i \in I_j : (\widehat{R_1 \wedge \underline{R_2}})$ & $\hat{S}_{j,i} \vdash^{q'_{j,i}}_{\mathcal{P},RGSC(\mathcal{D})} D_{j,i}$*, con* $q'_{j,i} \leq p_j$ *y donde* $\bigvee_{i\in I_j} D_{j,i} = D_j$*.*

Entonces, podemos componer las siguientes computaciones:

(1) $(\widehat{\underline{R_1} \wedge R_2})$ & $\hat{S} \vdash_{\mathbf{RL}} \bigvee_{j\in J} (\widehat{\underline{R_1} \wedge R_2})$ & $\hat{S}_j \vdash^{q'_j}_{\mathcal{P},RGSC(\mathcal{D})} \bigvee_{j\in J} (\bigvee_{i\in I_j} \hat{S}_{j,i})$ *aplicando (1)*$_j$*, con tamaño* $1 + \sum_{j\in J} q'_j \leq 1 + \sum_{j\in J} p_j = p$*.*

(2) $\forall\, j \in J$ *y* $\forall\, i \in I_j : (\widehat{R_1 \wedge \underline{R_2}})$ & $\hat{S}_{j,i} \vdash^{q'_{j,i}}_{\mathcal{P},RGSC(\mathcal{D})} D_{j,i}$ *aplicando (2)*$_j$*, con tamaño* $q'_{j,i} \leq p_j < p$ *y* $\bigvee_{j\in J} (\bigvee_{i\in I_j} D_{j,i}) = \bigvee_{j\in J} D_j = D$*.*

□

Apéndice B

El sistema $\mathcal{TOY}(\mathcal{FD})$

B.1. Implementación de $CFLP(\mathcal{FD})$ en $\mathcal{TOY}(\mathcal{FD})$

Presentamos el sistema $\mathcal{TOY}(\mathcal{FD})$ como una implementación concreta de la instancia $CFLP(\mathcal{FD})$ de nuestro esquema genérico de programación lógico funcional con restricciones. Mediante $\mathcal{TOY}(\mathcal{FD})$ se consigue extender el sistema $\mathcal{TOY}$ [ALR07] a través del manejo y la resolución de restricciones de dominio finito $\mathcal{FD}$. Comenzamos describiendo las características más significativas del sistema, especialmente su sintaxis para el uso de restricciones $\mathcal{FD}$ y la corrección de su implementación. A continuación resaltamos sus principales ventajas mostrando la eficiencia de su ejecución a través de diversos resultados y comparativas con otros sistemas afines. El lector interesado puede ampliar la información contenida en este apéndice a través de la publicación [FHSV07], en la cual se describen tanto los fundamentos teóricos de $\mathcal{TOY}(\mathcal{FD})$, en la línea de los Capítulos 2 y 3 de esta memoria, como los detalles concretos de su implementación. También es posible obtener información más detallada del uso del sistema y de ejemplos concretos de programación con restricciones $\mathcal{FD}$ a través del manual de usuario de $\mathcal{TOY}(\mathcal{FD})$, el cuál se encuentra disponible en [FHS03c] y en la página web del sistema `http://toy.sourceforge.net`.

Como ya se ha indicado en la presentación de las herramientas de depuración presentadas en la segunda parte de esta memoria, tanto el desarrollo de las mismas como de la implementación de $\mathcal{TOY}(\mathcal{FD})$ en el sistema $\mathcal{TOY}$ ha sido el resultado de un intenso trabajo en equipo. En este sentido, la principal contribución del doctorando al desarrollo de $\mathcal{TOY}(\mathcal{FD})$ ha consistido, por una parte, en establecer las bases teóricas del lenguaje $CFLP(\mathcal{FD})$ como una instancia concreta del esquema presentado en la primera parte de esta tesis, y por otra, en clarificar aspectos de la implementación que ya existía de $\mathcal{TOY}(\mathcal{FD})$ antes de la realización de esta tesis mediante la incorporación de estrategias eficientes de cómputo basadas en estrechamiento perezoso con restricciones y árboles definicionales, siguiendo las pautas marcadas en los trabajos [Vad02, Vad03a, Vad03b, Vad05, EV05, Vad07].

B.1.1. Introducción

Comenzamos nuestra descripción del sistema $\mathcal{TOY}(\mathcal{FD})$ desde el punto de vista concreto de la programación describiendo en primer lugar, brevemente, tanto su sintaxis concreta como algunas de las restricciones $\mathcal{FD}$ predefinidas que ofrece el sistema.

Una visión general de $\mathcal{TOY}(\mathcal{FD})$

Un $CFLP(\mathcal{FD})$-programa en $\mathcal{TOY}(\mathcal{FD})$ está constituido por *declaraciones de tipos*, *tipos sinónimos*, definiciones de *operadores infijos* (los cuales juegan el papel de las funciones primitivas para el dominio $\mathcal{FD}$ presentadas en el Capítulo 2) y reglas de programa específicas para definir *funciones*. La sintaxis utilizada es muy similar a la que ofrece *Haskell* [Bir98, PHAB+02], con la excepción destacable de que tanto las variables de datos como las variables de tipos comienzan siempre con una letra en mayúscula, mientras que los símbolos de constructoras y los símbolos de tipos comienzan siempre con minúscula. En particular, las funciones están *curryficadas* y las convenciones usuales sobre la asociatividad de la aplicación son también consideradas en el lenguaje. Como es usual en programación funcional, los tipos declarados por el usuario se comprueban y los no declarados se infieren. Para ilustrar las definiciones de tipos de datos presentamos a continuación los siguientes ejemplos, que hacen uso de la sintaxis concreta de $\mathcal{TOY}(\mathcal{FD})$:

- `data nat = zero | suc nat`, para definir los números naturales.
- `data bool = false | true`, para definir el tipo predefinido de los booleanos.

Un programa $\mathcal{P}$ en $\mathcal{TOY}(\mathcal{FD})$ es, pues, un conjunto de *reglas de definición* para los símbolos de función en su signatura. Las reglas de definición para una función `f` tienen la forma sintáctica básica `f` $\mathtt{t}_1$ `...` $\mathtt{t}_n$ `= r <== C`, para las que, informalmente, su significado pretendido se corresponde con el que hemos dado en la Sección 2.6, salvo que no se hace uso de producciones en la parte condicional de la regla: una llamada a `f` puede ser reducida a `r` siempre que los parámetros actuales se ajusten a los patrones $\mathtt{t}_i$ y las restricciones en `C` se satisfagan. $\mathcal{TOY}(\mathcal{FD})$ también permite definir *predicados* (de manera similar a como se hace en programación lógica), los cuales pueden ser vistos como un caso particular de funciones booleanas con tipo `p ::` $\overline{\tau}_n$ `-> bool`. Como una facilidad sintáctica que ofrece el sistema, es posible usar *cláusulas* como una abreviatura particular para las reglas de definición cuyo lado derecho sea `true`. Esto permite escribir definiciones de predicados con un estilo muy semejante al que se utiliza en Prolog [SS86, CM87], de modo que una cláusula de la forma `p` $\mathtt{t}_1$ `...` $\mathtt{t}_n$ `:- C` permite abreviar una regla de definición de función de la forma `p` $\mathtt{t}_1$ `...` $\mathtt{t}_n$ `= true <== C`. Con este azúcar

```
% Elección indeterminista de dos valores
   infixr 40 //
   X // Y = X
   X // Y = Y
% Inserción indeterminista de un elemento en una lista
   insert X []     = [X]
   insert X [Y|Ys] = [X,Y|Ys] // [Y|insert X Ys]
% Generación indeterminista de permutaciones de una lista
   permut []     = []
   permut [X|Xs] = insert X (permut Xs)
% Determinar si una lista de números está ordenada
   sorted []       = true
   sorted [X]      = true
   sorted [X,Y|Ys] = sorted [Y|Ys] <== X <= Y
% Ordenación perezosa de una lista usando 'generate-and-test'
   sort Xs = check_list (permut Xs)
   check_list Xs = Xs <== sorted Xs == true
```

Objetivo	*Respuestas*
(a) `sort [4,2,5,1,3] == L`	`L == [1,2,3,4,5]; no more solutions`
(b) `sort [3,2,1] /= L`	`L /= [1,2,3] ; no more solutions`
(c) `F [2,1,3] == [1,2,3]`	`F == permut; F == sort; ...`

Cuadro B.1: Ejemplos básicos de programación en $\mathcal{TOY}(\mathcal{FD})$

sintáctico en mente, y algunos otros cambios obvios como la eliminación de la currificación, cualquier $CLP(\mathcal{FD})$-programa (puro) puede ser directamente traducido a un $CFLP(\mathcal{FD})$-programa equivalente.

Ejemplos sencillos de programación en $\mathcal{TOY}(\mathcal{FD})$

El Cuadro B.1 muestra algunos ejemplos sencillos de programación en el sistema $\mathcal{TOY}(\mathcal{FD})$ que no hacen uso de la extensión sobre $\mathcal{FD}$, junto con algunos objetivos y sus resultados [LS99b]. Ejemplos más complejos y de mayor interés práctico se mostrarán en la Sección B.3. El uso de operadores de restricciones infijos está permitido en $\mathcal{TOY}(\mathcal{FD})$, tales como `//` para construir la expresión `X // Y`, la cual es entendida como `// X Y`.

El objetivo (a) en el Cuadro B.1 permite ordenar una lista en una computación puramente funcional. La respuesta para el objetivo (b) involucra una desigualdad sintáctica. En el objetivo (c), `F` es una variable lógica de orden superior y los valores objetivos para esta variable son patrones de orden superior (`permut`, `sort`, etc).

```
RESTRICCIONES RELACIONALES
    (# =), (#\=) :: int → int → bool
    (#<), (#<=), (#>), (#>=) :: int → int → bool
RESTRICCIONES ARITMÉTICAS
    (#+), (#−), (#∗), (#/) :: int → int → int
RESTRICCIONES DE PERTENENCIA
    domain :: [int] → int → int → bool
RESTRICCIONES DE ENUMERACIÓN
    labeling :: [labelType] → [int] → bool
RESTRICCIONES COMBINATORIAS
    all_different :: [int] → bool
```

Cuadro B.2: Subconjunto de restricciones $\mathcal{FD}$ predefinidas en $\mathcal{TOY}(\mathcal{FD})$

Uso de restricciones $\mathcal{FD}$ en $\mathcal{TOY}(\mathcal{FD})$

El Cuadro B.2 muestra un pequeño subconjunto de las restricciones $\mathcal{FD}$ admitidas por $\mathcal{TOY}(\mathcal{FD})$, las cuales son instancias típicas que se pueden encontrar en cualquier otro sistema CP. En este cuadro, `int` representa el tipo básico de los enteros, `[`τ`]` es el tipo 'lista de τ', y el tipo de datos `labelType` es un tipo predefinido usado para definir varias estrategias de búsqueda para el etiquetado de variables de dominio finito [FHS03c].

Las primitivas usadas en restricciones relacionales son aplicadas a enteros y devuelven un valor booleano, mientras que las primitivas aritméticas son aplicadas a enteros y devuelven valores enteros. Estos pueden ser combinados con las primitivas de restricciones relacionales para construir (des)igualdades (no)lineales como restricciones. Más aún, las *restricciones reificadas* (véase el Capítulo 8 de [MS98]) pueden ser implementadas igualando una variable booleana a una restricción booleana para todas las restricciones construidas a partir de los operadores en este cuadro y la restricción `domain` (ver el Ejemplo 23). Debido a la componente funcional, podemos aplicar esta técnica para igualar también expresiones booleanas a restricciones booleanas. Ambas, las primitivas de restricciones relacionales y aritméticas, se distinguen sintácticamente añadiendo el prefijo # a las primitivas relacionales usuales con el fin de denotar su diferente comportamiento operacional.

La restricción de pertenencia `domain` restringe una lista de variables (su primer argumento) a tomar valores en un intervalo entero (definido por sus dos siguientes argumentos enteros) siempre que su resultado sea el valor `true`, mientras que restringe estas variables a tomar valores diferentes del intervalo cuando su resultado tiene el valor `false`. La restricción de enumeración `labeling` asigna valores a las variables en su lista de enteros de entrada de acuerdo a las opciones que se hayan especificado con el argumento de tipo lista `labelType`. En esta lista se especifican tanto las estrategias de búsqueda tales como *first-fail* (ver la Subsección B.2.1) como las

opciones de optimización para encontrar valores mínimos y máximos para funciones de coste. La restricción combinatoria `all_different` asegura valores diferentes para los elementos pertenecientes a su lista argumento y es un ejemplo del conjunto de restricciones globales soportadas por $\mathcal{TOY}(\mathcal{FD})$, para la cual un algoritmo de propagación eficiente sí ha sido desarrollado.

En este apéndice no mencionamos ni explicamos todas las restricciones predefinidas en detalle, pero sí se recomienda al lector interesado visitar el enlace propuesto en [FHS03c] para una explicación más detallada. Enfatizamos que todos los fragmentos de código que aparecen en este apéndice son ejecutables en $\mathcal{TOY}(\mathcal{FD})$, y que las respuestas para los objetivos de los ejemplos corresponden a ejecuciones reales de los programas.

Ejemplo 23 *A continuación mostramos la resolución, al nivel de la línea de comandos de $\mathcal{TOY}(\mathcal{FD})$, de un objetivo muy sencillo que no involucra etiquetado pero sí hace uso de otras restricciones $\mathcal{FD}$.*

```
TOY(FD)> domain [X, Y] 10 20, X #<= Y == L
         yes     L == true,  X in 10..20, Y in 10..20;
         yes     L == false, X in 11..20, Y in 10..19;
         no
```

Se observa que esta implementación de CFLP($\mathcal{FD}$) sólo proporciona una información limitada de salida, consistente de: (1) sustituciones de la forma `Variable == Patrón`*, (2) restricciones de desigualdad* `Variable /= Patrón`*, (3) disyunciones D de restricciones* `Variable in RangoEntero` *(estas restricciones denotan los posibles valores que una variable* podría *tomar, como es común en sistemas con restricciones; es decir, estas restricciones no determinan D sino la negación de D), y (4) información de éxito:* `yes` *y* `no` *denotan* éxito *y* fallo*, respectivamente. Finalmente, '*;*' separa las soluciones que han sido solicitadas explícitamente por el usuario. Las restricciones primitivas en el almacén de restricciones de $\mathcal{FD}$ no se muestran.*

Ejemplo 24 *Mostramos ahora un programa en $\mathcal{TOY}(\mathcal{FD})$ que sí involucra etiquetado para resolver el clásico problema de las N-reinas, cuyo objetivo es el de situar N reinas sobre un tablero de ajedrez de dimensiones $N \times N$ de modo que no se ataquen entre sí.*

```
include "misc.toy"
include "cflpfd.toy"

queens :: int -> [labelType] -> [int]
queens N Label = L <== length L == N, domain L 1 N,
                       constrain_all L, labeling Label L
```

```
constrain_all :: [int] ->  bool
constrain_all []     = true
constrain_all [X|Xs] = true <== constrain_between X Xs 1,
                                constrain_all Xs

constrain_between :: int -> [int] -> int -> bool
constrain_between X []     N = true
constrain_between X [Y|Ys] N = true <== no_threat X Y N,
                                        constrain_between X Ys (N+1)

no_threat:: int -> int -> int -> bool
no_threat X Y I = true <== X #\= Y, X #+ I #\= Y, X #- I #\= Y
```

El significado pretendido de las funciones debería ser claro a partir de sus nombres y definiciones, y teniendo en cuenta que `length L` *permite devolver la longitud de la lista* `L`*. Las primeras dos líneas son necesarias para poder incluir funciones predefinidas, como por ejemplo* `length` *y* `domain`*. Como ejemplo de resolución en la línea de comandos del sistema, consideramos el siguiente objetivo en el que* `ff` *representa la estrategia de enumeración 'first-fail' (ver la Subsección B.2.1).*

```
TOY(FD)> queens 15 [ff] == L
         yes    L ==  [1,3,5,14,11,4,10,7,13,15,2,8,6,9,12]
```

Ejemplo 25 *Presentamos un tercer programa en $\mathcal{TOY}(\mathcal{FD})$ que resuelve el conocido problema de CLP($\mathcal{FD}$) denominado* Send + More = Money*. En él se utiliza azúcar sintáctico para representar funciones con el estilo de predicados.*

```
smm :: int -> int -> int -> int -> int -> int -> int ->
       [labelType] -> bool

smm S E N D M O R Y Label :-
         domain [S,E,N,D,M,O,R,Y] 0 9, S #> 0, M #> 0,
         all_different [S,E,N,D,M,O,R,Y],
         add S E N D M O R Y,
         labeling Label [S,E,N,D,M,O,R,Y]

add :: int -> int -> int -> int -> int -> int -> int ->
       bool
```

```
add S E N D M O R Y :-  1000 #* S #+ 100 #* E #+ 10 #* N #+ D
                     #+ 1000 #* M #+ 100 #* O #+ 10 #* R #+ E
      #=  10000 #* M #+ 1000 #* O #+ 100 #* N #+ 10 #* E #+ Y
```

El Cuadro B.3 muestra ejemplos concretos de objetivos y de respuestas que pueden ser computadas mediante el sistema $\mathcal{TOY}(\mathcal{FD})$ para los ejemplos anteriores.

Objetivo	*Respuestas*

`domain [A,B] 1 (1+2),`
`A #> B,`
`all_different [A,B],`
`labeling [ ] [A,B]`
	`A == 2, B == 1;`
	`A == 3, B == 1;`
	`A == 3, B == 2;`
	`no more solutions`

`domain [X,Y,Z] 1 10,` `2 #* X #+ 3 #* Y #+ 2 #< Z`	`X in 1..2, Y == 1, Z in 8..10;` `no more solutions`
`domain [X,Y,Z] 1 5, X #> Y,` `2 #* Y #> Z #+ 4, X #>= Z`	`X in 4..5,Y in 3..4,Z in 1..3;` `no more solutions`

`smm S E N D M O R Y [] == T`	`S == 9,E == 5,N == 6,D == 7,M == 1,`
	`O == 0,R == 8,Y == 2,T == true;`
	`no more solutions`

`queens 5 [] == [M,A,E,Y,B],`
`smm S E N D M O R Y []`
	`M == 1,A == 3,E == 5,Y == 2,B == 4,`
	`S == 9,N == 6,D == 7,O == 0,R == 8;`
	`no more solutions`

Cuadro B.3: Ejemplos de resolución de objetivos en $\mathcal{TOY}(\mathcal{FD})$

B.1.2. Comparativa entre $CFLP(\mathcal{FD})$ y $CLP(\mathcal{FD})$

Es bien conocido que $CLP(\mathcal{FD})$ es una aproximación declarativa a la programación con restricciones $\mathcal{FD}$ que goza de un gran éxito y popularidad; en esta subsección vamos a tratar de discutir brevemente las principales ventajas que conlleva adoptar $CFLP(\mathcal{FD})$ con respecto a $CLP(\mathcal{FD})$, centrando nuestra atención en la implementación realizada en $\mathcal{TOY}(\mathcal{FD})$. Para ello, vamos a comenzar viendo por qué la incorporación de características propias de la programación funcional permite enriquecer el contexto CLP. Cuando sea necesario ilustraremos las diferentes características de $CFLP(\mathcal{FD})$ mediante ejemplos concretos. Otros ejemplos de programación en programación lógico funcional pura y $CFLP(\mathcal{FD})$ pueden encontrarse, respectivamente, en [LS99b] y en [FHS03c].

$CFLP(\mathcal{FD}) \supset CLP(\mathcal{FD})$

Como ya se ha indicado en el Capítulo 1, $CFLP(\mathcal{FD})$ proporciona junto con otros aspectos las principales características de $CLP(\mathcal{FD})$, es decir, la resolución de restricciones $\mathcal{FD}$, el indeterminismo, y por supuesto, la forma relacional. Más aún, $CFLP(\mathcal{FD})$ proporciona azúcar sintáctico para representar predicados en programación lógica, y de este modo cualquier $CLP(\mathcal{FD})$-programa puro puede ser fácilmente traducido en un $CFLP(\mathcal{FD})$-programa equivalente. En este sentido, $CLP(\mathcal{FD})$ puede ser considerado como un subconjunto estricto de $CFLP(\mathcal{FD})$ con respecto a la formulación de problemas. Como consecuencia directa, nuestro lenguaje es capaz de tratar con un amplio rango de aplicaciones (al menos con todas aquellas aplicaciones que pueden ser formuladas en el lenguaje $CLP(\mathcal{FD})$). No insistiremos aquí sobre este aspecto, pues preferimos concentrarnos en las capacidades extras de $CFLP(\mathcal{FD})$ con respecto a $CLP(\mathcal{FD})$.

$CFLP(\mathcal{FD}) \setminus CLP(\mathcal{FD})$

Debido a su componente funcional, $CFLP(\mathcal{FD})$ añade una mayor expresividad a $CLP(\mathcal{FD})$ al permitir la declaración de funciones y la evaluación perezosa en el estilo más clásico de la programación funcional. En lo que sigue vamos a enumerar, y a discutir, otros aspectos que no están presentes (o que son inusuales) en el paradigma $CLP(\mathcal{FD})$.

Tipos. El lenguaje de programación que proporciona el sistema $\mathcal{TOY}(\mathcal{FD})$ es fuertemente tipado, y de este modo involucra todas sus ventajas, como el proceso de comprobación de tipos, el cual permite enriquecer el desarrollo de los programas y su posterior mantenimiento. Cada restricción $\mathcal{FD}$ tiene asociada una declaración de tipos, como cualquier función, lo que significa que un uso incorrecto puede ser detectado a través de un proceso de comprobación de tipos.

Notación Funcional. Es bien conocido que la notación funcional reduce el número de variables con respecto a la notación relacional, y de este modo $CFLP(\mathcal{FD})$ incrementa la expresividad de $CLP(\mathcal{FD})$ al combinar la notación relacional con la notación funcional. Por ejemplo, en $CLP(\mathcal{FD})$ la conjunción de restricciones `N = 2, X` $\in$ `[1,10-N]` no se puede expresar directamente y debe ser escrita añadiendo una tercera componente, como `N = 2, Max is 10-N, domain([X],1,Max)`, que utiliza una variable extra. Sin embargo, $\mathcal{TOY}(\mathcal{FD})$ permite expresar la restricción directamente como `N == 2, domain [X] 1 (10-N)`.

Currificación. De nuevo, debido a su componente funcional, $\mathcal{TOY}(\mathcal{FD})$ permite utilizar funciones currificadas (y de este modo también restricciones); por ejemplo,

el lector puede remitirse a la aplicación de la restricción $\mathcal{FD}$ currificada que aparece más adelante en el Ejemplo 26.

Orden Superior y Polimorfismo. En $\mathcal{TOY}(\mathcal{FD})$ las funciones son *ciudadanos de primera clase*, lo que significa que una función (y de este modo de nuevo también una restricción $\mathcal{FD}$) puede aparecer en cualquier lugar donde haya sido declarada. Como consecuencia directa, una restricción $\mathcal{FD}$ puede aparecer como un argumento (o incluso como un resultado) de otra función o restricción. Las funciones que manejan otras funciones se denominan *funciones de orden superior*, como ya se ha comentado. Asimismo, la utilización de argumentos *polimórficos* en $CFLP(\mathcal{FD})$ también está permitida.

Ejemplo 26 *Un ejemplo tradicional de una función polimórfica de orden superior es la que se describe a continuación:*

```
map :: (A -> B) -> [A] -> [B]
map F []     = []
map F [X|Xs] = [(F X) | (map F Xs)]
```

La función `map` *recibe como argumentos una función* `F` *y una lista, y produce otra lista resultante de la aplicación de la función a cada uno de los elementos en la lista inicial. Ahora supongamos que* `X` *e* `Y` *son variables de* $\mathcal{FD}$ *con rango en el dominio [0..100] (el cuál se expresa, por ejemplo, mediante la restricción* `domain [X,Y] 0 100`*). Entonces, el objetivo* `map (3 #<) [X,Y]` *permite devolver la lista de booleanos* `[true,true]` *resultante de la evaluación de la lista* `[3 #< X,3 #< Y]`*, y* `X` *e* `Y` *son también restringidas a tomar valores en el rango* `[4,100]` *puesto que las restricciones* `3 #< X` *y* `3 #< Y` *son enviadas al resolutor de restricciones. Obsérvese también el uso de la función currificada* `(3 #<)`*.*

Pereza. En contraste con los lenguajes lógicos, los lenguajes funcionales soportan *evaluación perezosa*, donde los argumentos de las funciones son evaluados hasta la extensión que sea requerida (el *call-by-value* usado en programación lógica vs. el *call-by-need* usado en programación funcional). Estrictamente hablando, la evaluación perezosa se corresponde con la noción de *only once evaluated* en adición al *only required extent* [Pey87]. $\mathcal{TOY}(\mathcal{FD})$ incrementa el poder de $CLP(\mathcal{FD})$ incorporando un mecanismo novedoso que permite combinar *evaluación perezosa* y resolución de restricciones $\mathcal{FD}$, de tal modo que sólo las restricciones que son demandadas son enviadas al resolutor. Éste es un poderoso mecanismo que abre nuevas posibilidades para la resolución de restricciones $\mathcal{FD}$. Por ejemplo, en contraste con $CLP(\mathcal{FD})$, es posible manejar estructuras de datos infinitas.

Ejemplo 27 *Consideramos las funciones recursivas* `take` *y* `generateFD` *dadas en el Ejemplo 9. Una evaluación impaciente del siguiente objetivo no termina cuando trata*

de evaluar completamente el segundo argumento, dando así lugar a una computación infinita. Sin embargo, una evaluación perezosa genera exactamente los 3 primeros elementos de la lista, como se muestra a continuación:

```
TOY(FD)> take 3 (generateFD 10) == List
         yes    List == [ _A, _B, _C ]     _A, _B, _C in 0..9
```

En general, el estrechamiento perezoso impide realizar cómputos que no sean demandados, y por tanto mejora el tiempo de computación. El Ejemplo 28 contiene una formulación del típico problema de *series* (o *secuencias*) *mágicas* que aparece en [vHen89]. Este ejemplo permite resaltar el poder expresivo de $\mathcal{TOY}(\mathcal{FD})$ resolviendo múltiples instancias del problema que pueden ser descritas y resueltas mediante la evaluación perezosa de listas infinitas.

Ejemplo 28 *Sea* $S = (s_0, s_1, \ldots, s_{N-1})$ *una serie finita no vacía de enteros no negativos. La serie* S *se dice que es* N-mágica *si y sólo si existen* s_i *apariciones de* i *en* S*, para todo* $i \in \{0, \ldots, N-1\}$*. A continuación proponemos un programa en* $\mathcal{TOY}(\mathcal{FD})$ *que permite calcular series mágicas, donde la función* `generateFD` *se define como en el Ejemplo 9.*

```
lazymagic :: int -> [int]
lazymagic N = L <== take N (generateFD N) == L,
                    constrain L L 0 Cs,
                    sum L (#=) N,
                    scalar_product Cs L (#=) N,
                    labeling [ff] L

constrain :: [int] -> [int] -> int -> [int] -> bool
constrain []     A B []     = true
constrain [X|Xs] L I [J|Js] = true <== I == J,
                                       count I L (#=) X,
                                       constrain Xs L (I+1) Js
```

`sum`*/3,* `scalar_product`*/4 y* `count`*/4 son restricciones predefinidas de orden superior [FHS03c] que aceptan un operador relacional de restricción* $\mathcal{FD}$ *con tipo asociado* `int` → `int` → `bool` *como argumento (por ejemplo, la restricción* `#=`*). Por otro lado,* `sum L C N` *significa que la suma de los elementos de la lista* `L` *está relacionada a través de* `C` *con el entero* `N` *(en el ejemplo, la suma está restringida a ser igual a* `N`*).* `scalar_product` *y* `count` *denotan el producto escalar y el conteo de elementos, respectivamente, bajo los mismos parámetros que* `sum`*.*

Un objetivo `lazymagic N`*, para algún natural* `N`*, devuelve las series* `N`*-mágicas, donde la condición* `take N (generateFD N)` *es evaluada perezosamente debido a*

que `(generateFD N)` *produce una lista infinita. Para este problema resulta más interesante devolver una lista de soluciones diferentes que comiencen a partir de* `N`*. Esto puede hacerse usando una definición recursiva para producir la lista infinita de series mágicas* `(from N)`*, como se muestra a continuación:*

```
magicfrom :: int -> [[int]]
magicfrom N = [lazymagic N | magicfrom (N+1)]
```

Ahora es posible generar una lista de series mágicas mediante evaluación perezosa. Por ejemplo, el siguiente objetivo genera una lista de tres elementos, los cuales contienen, respectivamente, la solución a los problemas de series 7-mágicas, 8-mágicas y 9-mágicas.

```
TOY(FD)> take 3 (magicfrom 7) == L
         yes     L == [ [ 3, 2, 1, 1, 0, 0, 0 ],
                        [ 4, 2, 1, 0, 1, 0, 0, 0 ],
                        [ 5, 2, 1, 0, 0, 1, 0, 0, 0 ] ]
```

Aún es posible obtener una mayor expresividad mezclando funciones currificadas, funciones de orden superior y composición de funciones (otra de las buenas características procedente de la componente funcional de $\mathcal{TOY}(\mathcal{FD})$). Por ejemplo, consideremos el código en $\mathcal{TOY}(\mathcal{FD})$ que se muestra a continuación:

```
from :: int -> [int]
from N = [N | from (N+1)]

(.) :: (B -> C) -> (A -> B) -> (A -> C)
(F . G) X = F (G X)

lazyseries :: int -> [[int]]
lazyseries = map lazymagic . from
```

donde `(.)`*/2 define la composición de funciones. Obsérvese que* `lazyseries` *currifica la composición* `(map lazymagic) . from`*. A partir de este código es fácil generar la lista de 3 elementos mostrada más arriba introduciendo el siguiente objetivo:*

```
TOY(FD)> take 3 (lazyseries 7) == L
```

El ejemplo anterior proporciona una idea clara de las principales características de $CFLP(\mathcal{FD})$ como la combinación de la resolución de restricciones $\mathcal{FD}$, el mantenimiento de listas infinitas y la evaluación perezosa, la notación currificada de funciones, polimorfismo, funciones de orden superior (y de este modo también restricciones de orden superior) y composición de funciones, que sin duda incrementan la potencia expresiva de la programación con respecto a $CLP(\mathcal{FD})$.

B.1.3. Algunas notas sobre la implementación de $\mathcal{TOY}(\mathcal{FD})$

La implementación del sistema $\mathcal{TOY}(\mathcal{FD})$ permite integrar, como lenguaje anfitrión, el lenguaje lógico funcional perezoso de orden superior $\mathcal{TOY}$, y como resolutor de restricciones, el eficiente resolutor de restricciones $\mathcal{FD}$ de SICStus Prolog [COC97]. Para poder evaluar internamente las restricciones $\mathcal{FD}$ se utilizan principalmente dos predicados: `hnf(E,H)`, mediante el cual se especifica que `H` es uno de los posibles resultados de la evaluación de la expresión `E` a su correspondiente *forma normal de cabeza* (en inglés, *head normal form*, `hnf`), y `solve`/1 mediante el que se realiza un *chequeo de la satisfactibilidad* de las restricciones que aparecen en reglas y objetivos antes de la evaluación de una regla dada. Este predicado se define, básicamente, como sigue [1]:

(1) $\texttt{solve}((\varphi, \varphi'))$	:−	$\texttt{solve}(\varphi), \texttt{solve}(\varphi')$.
(2) $\texttt{solve}(\texttt{L} == \texttt{R})$	:−	$\texttt{hnf}(\texttt{L}, \texttt{L}'), \texttt{hnf}(\texttt{R}, \texttt{R}'), \texttt{equal}(\texttt{L}', \texttt{R}')$.
(3) $\texttt{solve}(\texttt{L} /= \texttt{R})$	:−	$\texttt{hnf}(\texttt{L}, \texttt{L}'), \texttt{hnf}(\texttt{R}, \texttt{R}'), \texttt{notequal}(\texttt{L}', \texttt{R}')$.
(4) $\texttt{solve}(\texttt{L} \#\diamondsuit \texttt{R})$	:−	$\texttt{hnf}(\texttt{L}, \texttt{L}'), \texttt{hnf}(\texttt{R}, \texttt{R}'), \{\texttt{L}' \#\diamondsuit \texttt{R}'\}$. donde $\diamondsuit \in \{=, \backslash=, <, <=, >, >=\}$.
(5) $\texttt{solve}(\texttt{C}\ \texttt{A}_1 \ldots \texttt{A}_n)$	:−	$\texttt{hnf}(\texttt{A}_1, \texttt{A}'_1), \ldots, \texttt{hnf}(\texttt{A}_n, \texttt{A}'_n), \{\texttt{C}(\texttt{A}'_1, \ldots, \texttt{A}'_n)\}$. donde `C` es cualquier restricción que devuelva un booleano.

La interacción con el resolutor de restricciones de $\mathcal{TOY}(\mathcal{FD})$ (es decir, el resolutor de restricciones $\mathcal{FD}$ de SICStus en la actual versión del sistema) se refleja en las dos últimas cláusulas: cada vez que una restricción $\mathcal{FD}$ aparece, el resolutor es eventualmente invocado con un objetivo $\{G\}$, donde G es la traducción de la restricción $\mathcal{FD}$ desde $\mathcal{TOY}(\mathcal{FD})$ a SICStus Prolog. Las formas normales de cabeza son requeridas para todos aquellos argumentos que sean restricciones con el fin de permitir al resolutor resolver cada restricción.

De acuerdo con las suposiciones expuestas en el Capítulo 2 para los resolutores de restricciones en el esquema $CFLP(\mathcal{D})$, el resolutor de dominios finitos de SICStus Prolog satisface razonablemente las condiciones enunciadas en el Postulado 2. Por otra parte, puesto que los cálculos operacionales que hemos presentado en el Capítulo 4 son *fuertemente completos* en el sentido de que la elección de las reglas de transformación de objetivos es una elección *don't care*, en la implementación de $\mathcal{TOY}(\mathcal{FD})$ es posible seleccionar una estrategia dirigida por demanda que resulte adecuada en la práctica: el resolutor de restricciones se aplica sólo al final del proceso de resolución de objetivos, justo cuando ya se dispone del máximo posible de información necesaria para ello. En este sentido, hemos realizado un análisis minucioso de la implementación del sistema $\mathcal{TOY}(\mathcal{FD})$ y hemos comparado las derivaciones

[1] El código no corresponde exactamente a la implementación, pues es el resultado de varias transformaciones y optimizaciones.

producidas mediante el cálculo de estrechamiento con restricciones $CDNC(\mathcal{FD})$ con las trazas obtenidas a partir de $\mathcal{TOY}(\mathcal{FD})$. La conclusión a la que hemos llegado es que ambas son prácticamente idénticas, si se sigue la estrategia dirigida por demanda que hemos descrito para $CDNC(\mathcal{D})$ a través de la utilización de árboles definicionales. Esto nos afianza en la idea de que la semántica operacional que se ha descrito formalmente en el Capítulo 4 de esta tesis permite cubrir adecuadamente la implementación realizada en $\mathcal{TOY}(\mathcal{FD})$, tal y como se describe en nuestras publicaciones [FHSV07] y [EV05].

Con el fin de ilustrar más detalladamente esta idea, vamos a mostrar mediante un ejemplo concreto cómo la ejecución en $\mathcal{TOY}(\mathcal{FD})$ de un objetivo se ajusta a nuestra estrategia de estrechamiento perezoso con árboles definicionales descrita a través del cálculo $CDNC(\mathcal{FD})$. Consideramos el objetivo *check_list* $(from\ M) < 3$ para el $CFLP(\mathcal{FD})$-programa (y por tanto para un programa en $\mathcal{TOY}(\mathcal{FD})$) del Ejemplo 9. El sistema permite devolver las siguientes respuestas:

```
Toy(FD)> check_list (from M) < 3
         yes
         M in 1..2
         Elapsed time: 0 ms.

         more solutions (y/n/d) [y]?
         yes
         M in 3..4
         Elapsed time: 0 ms.

         more solutions (y/n/d) [y]?
         no.
         Elapsed time: 0 ms.
```

Las respuestas computadas se corresponden exactamente con las que han sido obtenidas en el Ejemplo 13 del Capítulo 4 utilizando un cálculo de resolución de objetivos. A continuación mostramos cómo cada paso de la derivación de estrechamiento con restricciones y árboles definicionales se ajusta a la traza correspondiente a este objetivo. Cada paso de la traza contiene el nombre del módulo del sistema que se encuentra activo en cada momento. Describimos brevemente antes los módulos que aparecen en los pasos de la traza.

- **initToy:** Contiene la interfaz con el usuario y reconoce, en la línea de comandos, objetivos, comandos y expresiones que han de ser evaluadas. Incluye los análisis léxico y sintáctico.
- **primitivCod:** Contiene el conjunto original de primitivas de $\mathcal{TOY}$.

- **plgenerated:** La compilación de un programa en $\mathcal{TOY}(\mathcal{FD})$ genera un programa en SICStus Prolog. Este programa queda definido en este módulo, el cual contiene definiciones predefinidas para tipos y funciones. El módulo es creado en el proceso de transformación. Tras la compilación, el fichero Prolog generado puede ser cargado, con lo que el usuario queda preparado para poder ejecutar objetivos en el sistema.

- **toycomm:** Este módulo es necesario para ejecutar programas en $\mathcal{TOY}(\mathcal{FD})$. El sistema lo carga automáticamente al comienzo. Contiene todos los predicados que son comunes a todos los programas en $\mathcal{TOY}$.

La derivación comienza planteando el objetivo $\square$ *check_list* $(from\ M) < 3 \square \square \varepsilon$. Inicialmente, $\mathcal{TOY}(\mathcal{FD})$ realiza un análisis léxico y sintáctico del objetivo. Si este proceso es correcto, entonces los tipos son chequeados, de forma que si los tipos también son correctos, entonces el objetivo es traducido a código Prolog. Cada restricción que forma parte del objetivo es entonces traducida y enviada al resolutor de dominios finitos de SICStus. El siguiente paso de la traza es análogo al primer paso dado en nuestra derivación de estrechamiento con restricciones:

```
Call: initToy: $<('$$susp'('$check_list',['$$susp'('$from',[_23825],
                                    _24738,_24736)],_24561,_24559),3,true,[],_20384)
```

En este paso de la traza se produce la llamada al predicado de desigualdad que se define a continuación mediante una cláusula Prolog. El predicado de desigualdad tiene un símbolo prefijo `$` con el fin de distinguir entre la primitiva de $\mathcal{TOY}(\mathcal{FD})$ y el predicado de desigualdad de Prolog:

```
$<(X,Y,H,Cin,Cout) :- hnf(X,HX,Cin,Cout1),
                      hnf(Y,HY,Cout1,Cout),
                     (number(HX),number(HY) ->
                                  (HX<HY,H=true;HX>=HY,H=false);errPrim).
```

Antes de poder invocar al resolutor de $\mathcal{FD}$ mediante la restricción `HX<HY`, tanto *check_list* $(from\ M)$ como 3 han de ser previamente transformados a *hnf*, de un modo similar a como se realiza la evaluación de los argumentos de la restricción siguiendo el cálculo de estrechamiento perezoso con restricciones. Puesto que 3 es un número, ya está en *hnf*. En el caso de *check_list* $(from\ M)$, puesto que es un argumento no primitivo, el sistema ha de computar su *hnf*. Una vez que los argumentos están en *hnf* el predicado de desigualdad verifica si ambos son números. Si es así, entonces son comparados mediante el operador relacional de Prolog (<) a través de la restricción `HX<HY`. En la derivación de estrechamiento, este paso se correspondería con la obtención del nuevo objetivo $\exists R.$ *check_list* $(from\ M) \rightarrow R \square$

$\square$ $R < 3$ $\square$ ε, mediante el cual se evalúa el argumento no primitivo de la restricción $\mathcal{FD}$ de manera similar a como se computa la *hnf* de *check_list* $(from\ M)$.

El siguiente paso en la traza es entonces la llamada al predicado que computa la *hnf* de *check_list* $(from\ M)$:

Call: primitivCod: **hnf**('$$susp'('$check_list',['$$susp'('$from',[_23825],_24738,_24736)], _24561,_24559), _28441,[ ],_28443)

Siguiendo con la derivación de estrechamiento, *check_list* $(from\ M)$ comienza con un símbolo de función definida en su raíz, mientras que R es una variable demandada. Por tanto, podemos despertar la suspensión *check_list* $(from\ M)$ introduciendo una nueva *producción demandada* con un árbol definicional asociado $< check\_list\ (from\ M), \mathcal{T}_{check\_list} > \rightarrow R$. Este paso da lugar a un nuevo objetivo $\exists R. < check\_list\ (from\ M), \mathcal{T}_{check\_list} > \rightarrow R\ \square\ \square\ R < 3\ \square\ \varepsilon$. El sistema $\mathcal{TOY}(\mathcal{FD})$ produce este efecto de despertar una suspensión por medio del predicado `hnf_susp`. Este predicado está contenido en el cuerpo del predicado `hnf`, cuyo comportamiento es el siguiente: si el argumento es una variable o una expresión con un símbolo pasivo en la raíz, entonces es una *hnf*. Si la expresión es una función definida, entonces esta aparece en *forma suspendida*. En este caso, `hnf` chequea que la expresión no haya sido evaluada (pues de otro modo, ésta debería ser devuelta como resultado). Es entonces cuando llama al predicado `hnf_susp`:

Call: toycomm:**hnf_susp**('$check_list',['$$susp' ('$from',[_23825],_24738,_24736)], _24561,[ ],_28443)

La definición de `hnf_susp` para `$check_list` es la siguiente:

hnf_susp('$check_list', '.'(_A, [ ]), _B, _C, _D) :- '$check_list'(_A, _B, _C, _D).

y la llamada correspondiente es:

Call: plgenerated: '$check_list'('$$susp'('$from',[_23825],_24738,_24736),_24561,[ ],_28443)

La definición del predicado `$check_list` sería como sigue:

'$check_list'(_A, _B, _C, _D) :- hnf(_A, _E, _C, _F),'$check_list_1'(_E, _B, _F, _D).

El predicado `$check_list` calcula primero la *hnf* de su argumento $from\ M$:

Call: plgenerated:hnf('$$susp'(**'$from'**,[**_23825**], _24738,_24736),_34510,[ ],_34512)

En la derivación, $\mathcal{T}_{check\_list}$ es un árbol definicional del tipo <u>*caso*</u>, y el símbolo que aparece en *check_list* $(from\ M)$ en la posición de distinción de casos es un símbolo de función definida $from$. En consecuencia, se introduce una nueva producción demandada en el objetivo, también con su correspondiente árbol definicional, con el fin de evaluar este argumento activo:

$$\exists R', R. < from\ M, \mathcal{T}_{from} > \rightarrow R', < check\_list\ R', \mathcal{T}_{check\_list} > \rightarrow R\ \square\ \square\ R < 3\ \square\ \varepsilon$$

Volviendo a la traza, como el argumento a evaluar (es decir, $from\ M$) no es ni una variable ni una expresión con un símbolo pasivo en la raíz, se invoca de nuevo al predicado `hnf_susp`:

Call: toycomm:**hnf_susp**('$**from**',[_**23825**],_24738, [ ],_34512)

La función `hnf_susp` para `$from` estaría definida del modo siguiente:

hnf_susp('$from', '.'(_A, [ ]), _B, _C, _D) :- '$from'(_A, _B, _C, _D).

y la correspondiente llamada sería

Call: plgenerated:'$from' (_23825,_24738,[ ],_34512)

La función `$from` se define entonces como

'$from'(_A, :(_A, '$$susp'('$from',['$$susp' ($+, [_A, 1], _B, _C)], _D, _E)), _F, _F).

Puesto que éste es un hecho Prolog, la computación de *hnf* para `$from` finaliza, lo que se corresponde con el siguiente paso de nuestra derivación:

$$\exists R', R.\ [M|from\ (M{+}1)] \rightarrow R', < check\_list\ R', \mathcal{T}_{check\_list} > \rightarrow R \square\ \square\ R < 3 \square\ \varepsilon \vdash_{\{R' \mapsto [M|As]\}}$$
$$\exists As, R.\ from\ (M{+}1) \rightarrow As, < check\_list\ [M|As], \mathcal{T}_{check\_list} > \rightarrow R\ \square\ \square\ R < 3\ \square\ \varepsilon$$

El cálculo de resolución de objetivos efectúa unificación sintáctica en el proceso de estrechamiento. En la traza se mostraría la salida del predicado `hnf` con la parte que se corresponde con $[M|from\ (M{+}1)]$:

Exit: plgenerated: hnf('$$susp'('$from',[_23825],_23825:
'$$susp'('$from',['$$susp'($+,[_23825,1],_40354,_40355)], _40347,_40348),hnf), _**23825:**
'$$susp'('**$from**',['$$susp' (**$+**,[_**23825**,**1**],_40354,_40355)],_40347,_40348),[ ],[ ])

Una vez concluida la computación de la *hnf* del argumento `$from` de `$check_list`, se produce una distinción de cuatro posibles alternativas a través de la definición del predicado `$check_list_1`, de acuerdo a la aplicación de las cuatro posibles reglas de la función *check_list* que figuran en el Ejemplo 9 y que sirven también para definir el correspondiente árbol definicional de esta función:

```
'$check_list_1'(_A, 0, _B, _C) :- unifyHnfs(_A, [ ], _B, _C).
'$check_list_1'(_A, 1, _B, _C) :- unifyHnfs(_A, :(_D, _E), _B, _F),
                                  '$domain'(:(_D, [ ]), 1, 2, true, _F, _C).
'$check_list_1'(_A, 2, _B, _C) :- unifyHnfs(_A, :(_D, _E), _B, _F),
                                  '$domain'(:(_D, [ ]), 3, 4, true, _F, _C).
'$check_list_1'(_A, 4, _B, _C):- unifyHnfs(_A, :(_D, _E), _B, _F),
                                  '$domain'( :(_D, [ ]), 5, 7, true, _F, _C).
```

La llamada al predicado `$check_list_1` es

Call: plgenerated:'**$check_list_1**'(_**23825**:'$$susp'('$from',['$$susp'($+,[_23825,1], _40354,_40355)], _40347,_40348),_24561,[ ],_28443)

Este predicado comienza tratando de unificar sus argumentos con la lista vacía a través de la primera cláusula:

Call: plgenerated: unifyHnfs(_23825:'$$susp' ('$from',['$$susp'($+,[_23825,1],_40354, _40355)],_40347,_40348),[ ],[ ],_28443)

y puesto que la unificación falla, continua con la segunda cláusula:

Call: plgenerated:unifyHnfs(_23825:'$$susp'('$from',['$$susp'($+,[_23825,1],_40354,_40355)],_40347,_40348),_**42798:_42799**,[ ],_42804)

Puesto que en este segundo caso la unificación ha tenido éxito, se continua con `$domain`:

Call: plgenerated: '**$domain**'(_23825:[ ],1,2,true, [ ],_28443)

La ejecución de `$domain` calcula la *hnf* de sus tres primeros argumentos (la lista $[M]$ y los números 1 y 2). A continuación, la restricción homónima de Prolog `domain` es ejecutada:

Call: plgenerated: domain([_23825],1,2)

con lo que la llamada a `domain` devuelve un éxito, y consecuentemente, la función `$check_list_1` también tiene éxito y devuelve 1 como la *hnf* de su primer argumento, es decir, de *check_list* $(from\ M)$. Esta parte de la traza se corresponde con el paso de la derivación:

$$\exists As, R.\ from\ (M+1) \to As,\ 1 \to R\ \square\ \square\ domain\ [M]\ 1\ 2,\ R < 3\ \square\ \varepsilon \vdash_{\{\mathbf{R} \mapsto \mathbf{1}\}}$$

Se obtiene así también un vínculo para el primer argumento de nuestra restricción:

$$\exists As.\ from\ (M+1) \to As\ \square\ \square\ domain\ [M]\ 1\ 2,\ 1 < 3\ \square\ \varepsilon$$

En la traza:

Exit: primitivCod: hnf('$$susp'('$check_list',['$$susp'('$from' ,[_58819],_58819: '$$susp'('$from',['$$susp'($+,[_58819|...],_40354,_40355)],_40347,_40348),hnf)],1,hnf),**1**,[],[])

Una vez que el cómputo de las *hnf*s en el predicado de desigualdad `($<)` ha concluido, el predicado Prolog $1 < 3$ es evaluado a `true`, y así la desigualdad `($<)` se hace también `true`, devolviendo la traza `Exit: 1<3`. En la derivación nos quedaría, analogamente:

$$\exists As.\ from\ (M+1) \rightarrow As\ \square\ \square\ domain\ [M]\ 1\ 2\ \square\ \varepsilon$$

y como As no aparece en el resto del objetivo, la suspensión $from\ (M+1) \rightarrow As$ es finalmente eliminada, dando lugar a la forma resuelta $\square\ \square\ domain\ [M]\ 1\ 2\ \square\ \varepsilon$. Se obtiene así la primera respuesta de la restricción $\mathcal{FD}$ proporcionada por el resolutor de SICStus, la cual sería mostrada por el sistema $\mathcal{TOY}(\mathcal{FD})$ finalmente en la forma `M in 1..2`.

En este punto, el sistema realiza una vuelta atrás con el objetivo de buscar respuestas alternativas:

Redo: initToy: $<('$$susp'('$check_list',['$$susp'('$from',[_58819],_58819:
'$$susp'('$from',['$$susp'($+,[_58819|...],_40354,_40355)],_40347,_40348),hnf)],1,hnf),3,[],[])

Siguiendo este proceso, las *hnf*s son ahora 2 and 3, respectivamente, con lo cual el predicado Prolog $2 < 3$ se evalúa a `true`. La desigualdad `($<)` se convierte también en `true` y la segunda respuesta es mostrada por $\mathcal{TOY}(\mathcal{FD})$: `M in 3..4`. De nuevo, el sistema hace "backtracking" con el fin de seguir buscando nuevas respuestas alternativas:

Redo: initToy: $<('$$susp'('$check_list',['$$susp'('$from' ,[_58819],_58819:
'$$susp'('$from',['$$susp'($+,[_58819|...],_40354,_40355)],_40347,_40348),hnf)],2,hnf),3,[],[])

Las *hnf*s son ahora 4 y 3. Puesto que 4 no es menor que 3, se buscan más *hnf*s de la expresión:

Redo: primitivCod: hnf('$$susp'('$check_list',['$$susp'('$from',[_58819],_58819:
'$$susp'('$from',['$$susp'($+,[_58819|...],_40354,_40355)],_40347,_40348),hnf)],4,hnf),4,[],[])

Puesto que no es posible encontrar más *hnf*s, no se buscan más alternativas, finalizando así la traza del objetivo, y de manera completamente análoga, la derivación mediante estrechamiento:

Fail: initToy: $<('$$susp'('$check_list',['$$susp'('$from',[_23825],
_24738,_24736)],_24561,_24559),3,true,[ ],_20384)

B.2. Resultados y comparativas de la eficiencia del sistema $\mathcal{TOY}(\mathcal{FD})$

Aunque $\mathcal{TOY}(\mathcal{FD})$ ha sido el primer sistema *CFLP* en integrar un sistema de restricciones $\mathcal{FD}$, destacamos también la existencia de una implementación en el lenguaje *Curry* [Han06] que soporta un conjunto limitado de restricciones $\mathcal{FD}$. Esta implementación, denominada *PAKCS* [Han07b], proporciona en concreto las siguientes restricciones:

(1) Un conjunto de primitivas {`*#,+#,-#,=#,/=#,<#,<=#,>#,>=#`},

(2) Una restricción de pertenencia `domain`/3,

(3) Algunas restricciones globales. Exactamente aquellas que se nombran en este apéndice, es decir, `all_different`/1, `count`/4, `scalar_product`/4 y `sum`/3.

(4) Una restricción de enumeración `labeling`/1 que también proporciona opciones de búsqueda.

En esta sección vamos a comparar la ejecución de nuestro sistema $\mathcal{TOY}(\mathcal{FD})$ con la del compilador de *Curry2Prolog*, que es la implementación de *Curry* utilizada dentro de *PAKCS* (en concreto, haremos uso en esta comparativa de la versión 1.7.1 de Diciembre de 2005, que es la que se utilizó en la publicación [FHSV07] para obtener los resultados experimentales que aquí se presentan). Además, para poder evaluar si $\mathcal{TOY}(\mathcal{FD})$ es competitivo con respecto a sistemas $CLP(\mathcal{FD})$ existentes, vamos a considerar también cuatro sistemas $CLP(\mathcal{FD})$ bien conocidos (de nuevo, haremos referencia a las versiones de estos sistemas que en su momento se utilizaron para obtener los resultados recogidos en [FHSV07]):

1) La versión 3.12.1 de Abril de 2005 del resolutor de restricciones $\mathcal{FD}$ de **SICStus Prolog** [COC97, SIC03]. Este resolutor ha sido incluido con el fin de poder medir la sobrecarga debida al mantenimiento de expresiones lógico funcionales, las cuales son compiladas a SICStus Prolog en $\mathcal{TOY}(\mathcal{FD})$, y por tanto incluyen toda la información extra necesaria para poder manejar características *FLP* tales como la pereza y las funciones de orden superior.

2) El sistema **GNU Prolog** (versión 1.2.16) [DC01, GNU05], un compilador libre de Prolog que incluye uno de los más eficientes resolutores de restricciones de dominio finito. Este resolutor está basado en el concepto de *indexical* [CD96] y se ha demostrado que su ejecución es comparable a sistemas comerciales.

3) **SWI-Prolog** (versión 5.4.x) [Wie03, SWI05], un sistema Prolog emergente y muy prometedor que proporciona un resolutor de restricciones de dominio de enteros implementado con variables atributo.

4) **Ciao Prolog** (versión 1.10#5 de Agosto de 2004), un entorno completo de programación multi-paradigma para el desarrollo de programas en el lenguaje Prolog, así como en diversos otros lenguajes que son extensiones y modificaciones de Prolog en diversas direcciones de interés. Ciao Prolog proporciona un paquete, basado en el concepto de *indexical*, para poder escribir y evaluar expresiones de programación con restricciones sobre dominios finitos en un programa Ciao.

B.2.1. Etiquetado (*labeling*)

La resolución de restricciones puede ser implementada mediante una combinación de dos procesos: la propagación de restricciones y el etiquetado o *labeling* (es decir, la búsqueda) [Dech03]. El proceso de etiquetado consiste en (1) elegir una variable (*ordenación de variables*) y (2) asignar a la variable elegida un valor que pertenezca a su dominio (*ordenación de valores*). La ordenación de variables y la ordenación de valores utilizadas para realizar el etiquetado puede influir considerablemente en la eficiencia de la resolución de restricciones cuando sólo una solución del problema es requerida (el efecto es pequeño cuando la búsqueda es la de todas las soluciones). En nuestro estudio vamos a considerar dos etiquetados, un etiquetado *básico* (o *naïve*) que elige la variable más a la izquierda de una lista de variables para a continuación seleccionar el valor más pequeño de su dominio, y el etiquetado *first-fail* que usa un famoso principio recogido en [HE80] que en inglés básicamente diría lo siguiente: *to succeed, try first where you are the most likely to fail*. Este principio recomienda la elección de la variable más restringida, lo que significa elegir (para el dominio finito) una variable con el dominio más pequeño. El etiquetado básico nos permite asegurar que ambos, la ordenación de variables y de valores, son los mismos para todos los sistemas, y en consecuencia (aunque menos eficiente) es mejor para comparar los diferentes sistemas cuando sólo una solución es requerida.

B.2.2. Comparativas de rendimiento y eficiencia (*benchmarks*)

Para realizar la comparativa de eficiencia se ha hecho uso de un amplio abanico de pruebas o *benchmarks* [2], y con el fin de beneficiar la comparativa, siempre que ha sido posible se ha usado exactamente la misma formulación de los problemas para todos los sistemas, así como también las mismas restricciones $\mathcal{FD}$. En concreto, los benchmarks que hemos utilizado son:

- **cars**: resuelve un *problema de secuenciación de caracteres* con 10 caracteres [DSvH88]. Este benchmark maneja 100 variables booleanas (es decir, variables de dominio finito que toman rango sobre [0,1]), 10 variables de dominio finito que toman rango sobre [1,6], 6 restricciones *atmost*, 50 restricciones *element* y 49 inecuaciones lineales.

- **equation 10**: un sistema de 10 ecuaciones lineales con 7 variables que toman rango sobre [0,10].

- **equation 20**: un sistema de 20 ecuaciones lineales con 7 variables con rango sobre [0,10].

[2] Todos los programas que se utilizan en la comparativa de este apéndice se encuentran disponibles en `http://toy.sourceforge.net`.

- **magic series** (N): calcula una serie de N números tales que cada uno de ellos es el número de apariciones en la serie de su posición en la serie [CD96].
- **optimal Golomb ruler** (N): encuentra un conjunto ordenado de n enteros no negativos distintos denominados *marcas*, $a_1 < \cdots < a_n$, tales que todas las diferencias $a_i - a_j$ $(i > j)$ son distintas y a_n es mínimo [She90].
- **queens** (N): sitúa N reinas sobre un tablero de ajedrez de dimensiones $N \times N$ de forma que ninguna reina ataca a ninguna otra [vHen89].
- **pythagoras**: calcula las proporciones de un triángulo usando el *Teorema de Pitágoras*. Este problema involucra 3 variables con rango sobre [1,1000] y 7 inecuaciones (no lineales).
- **sendmore**: un problema criptoaritmético con 8 variables con rango sobre [0,9], con una ecuación lineal, 2 inecuaciones y 28 restricciones de desigualdad (o alternativamente, una restricción *all_different* impuesta sobre el conjunto total de variables restringidas). El objetivo de este problema es el de resolver la ecuación $SEND + MORE = MONEY$.
- **suudoku**: el problema consiste en completar cuadrados parcialmente completos de 9×9 casillas tales que cada fila y cada columna sean permutaciones de [1,...,9] y de forma que cada cuadrado de 3×3 casillas, donde el módulo 3 de la columna más a la izquierda es 0, sea también una permutación de [1,...,9].

Los programas **equation 10**, **equation 20** y **sendmore** permiten testear la eficiencia de los sistemas para resolver problemas de ecuaciones lineales. Los programas **cars** y **suudoku** comprueban la eficiencia de restricciones especializadas, como por ejemplo, la restricción *all_different*. El problema **pythagoras** permite tratar ecuaciones no lineales.

Los programas **queens** y **magic series** son escalables, y por tanto útiles para hacer pruebas de cómo los sistemas trabajan para instancias más grandes del mismo problema. Obsérvese que ambos, el número de variables y el número de valores para cada variable, crecen linealmente con el parámetro N en los ejemplos. Es decir, dado un valor N, al menos N variables de $\mathcal{FD}$ deben ser declaradas con dominios con rango entre 0 (o bien 1) y N.

La búsqueda para **optimal Golomb ruler** explora un aspecto extremadamente difícil, puesto que se trata de un problema combinatorio cuyas cotas crecen geométricamente con respecto al tamaño de la solución [She90]. Este benchmark (también escalable) nos permite comprobar las capacidades de optimización del sistema.

B.2.3. Resultados

Todos los benchmarks han sido ejecutados sobre la misma máquina Linux (bajo el sistema Fedora Core system, 2.69-1667) con un procesador Intel(R) Pentium 4 a 2.40

Benchmark	$\mathcal{TOY}(\mathcal{FD})$	*PAKCS*	SICStus	SWI	GNU	Ciao
cars	5	N	5	N	1	N
equation10	20	50	10	590	2	-
equation20	35	60	10	1185	4	-
magic(64)	265	340	(430) 260	N (OGS)	134	N
magic(100)	910	980	(1520) 900	N (OGS)	901	N
magic(150)	2700	3180	(4770) 2560	N (OGS)	(SO) 4894	N
magic(200)	5970	6540	(10870) 5690	N (OGS)	(SO) 14570	N
magic(300)	18365	22750	(RE) 17780	N (OGS)	(SO) 68020	N
pythagoras	50	80	20	940	10	902
queens(8)	10	20	10	110	1	31
queens(16)	180	200	170	38720	11	6873
queens(20)	4030	4200	3930	1064130	216	190435
queens(24)	8330	8400	8120	??	460	576625
queens(30)	1141760	1141940	1069750	??	67745	??
sendmore	0	5	0	15	0	14
suudoku	10	20	10	60	1	51

Cuadro B.4: $\mathcal{TOY}(\mathcal{FD})$ vs. $C(F)LP$ sistemas: etiquetado naïve

GHz y con una memoria RAM de 512 Mb. Con el fin de no alargar la presentación, proporcionamos sólo los resultados obtenidos para la búsqueda de la primera solución.

El Cuadro B.4 muestra los resultados que se obtienen usando un etiquetado naïve. El significado para las columnas es el siguiente. La primera columna proporciona el nombre del benchmark usado en la comparativa, mientras que las siguientes seis columnas muestran el tiempo (transcurrido) de ejecución (medido en milisegundos) para encontrar la primera respuesta del benchmark en cada sistema.
El Cuadro B.5 muestra los resultados que aparecen en el Cuadro B.4 pero en términos de la *eficiencia* que $\mathcal{TOY}(\mathcal{FD})$ consigue con respecto al resto de sistemas (es decir, al resultado de dividir el tiempo de un sistema dado por el tiempo de $\mathcal{TOY}(\mathcal{FD})$).
El Cuadro B.6 muestra los resultados de resolver los mismos benchmarks pero ahora usando el etiquetado first-fail. Obsérvese que las versiones utilizadas de SWI-Prolog y Ciao Prolog no proporcionan etiquetado first-fail. Asimismo, el Cuadro B.7 muestra la eficiencia correspondiente a los resultados del Cuadro B.6, es decir, de la ejecución de $\mathcal{TOY}(\mathcal{FD})$ con respecto al resto de sistemas. El significado para las columnas es como en el Cuadro B.5, pero ahora se ha añadido una última columna con el fin de poder mostrar la eficiencia de $\mathcal{TOY}(\mathcal{FD})$ usando el etiquetado first-fail con respecto al mismo sistema pero con el etiquetado naïve. Los Cuadros B.8 y B.9 muestran los resultados correspondientes a la *optimización*. Particularmente, el Cuadro B.8 muestra el tiempo (transcurrido) medido en milisegundos para resolver el problema de optimización asociado al proceso de benchmarking, mientras que el Cuadro B.9 muestra la eficiencia de nuestro sistema con respecto al resto de sistemas. En to-

Benchmark	*PAKCS*	SICStus	SWI	GNU	Ciao
cars	∞	1 00	∞	0.20	∞
equation10	2.50	0.50	29.50	0.10	∞
equation20	1.71	0.28	33.85	0.11	∞
magic (64)	1.28	(1.62) 0.98	∞	0.50	∞
magic (100)	1.07	(1.67) 0.98	∞	0.99	∞
magic (150)	1.17	(1.76) 0.98	∞	(∞) 1.81	∞
magic (200)	1.09	(1.82) 0.99	∞	(∞) 2.44	∞
magic (300)	1.23	(∞) 0.96	∞	(∞) 3.70	∞
pythagoras	1.60	0.40	18.80	0.20	18.04
queens (8)	2.00	1.00	11.00	0.10	3.12
queens (16)	1.11	0.94	215.11	0.06	38.18
queens (20)	1.04	0.97	264.05	0.05	42.25
queens (24)	1.00	0.97	(?)	0.05	69.22
queens (30)	1.00	0.93	(?)	0.05	(?)
sendmore	$\geq$ 5.00	$\geq$ 1.00	$\geq$ 15.00	$\geq$ 1.00	$\geq$ 14.00
suudoku	2.00	1.00	6.00	0.10	5.10

Cuadro B.5: Eficiencia de $\mathcal{TOY}(\mathcal{FD})$ en relación al etiquetado naïve

dos estos cuadros, todos los números representan el promedio de diez ejecuciones. El símbolo ?? significa que no se obtuvo una solución para el benchmark en un tiempo razonable, mientras que (?) indica un valor indeterminado. El símbolo **N** en las columnas de *PAKCS*, SWI-Prolog y Ciao Prolog significa que no se ha podido formular el correspondiente benchmark, debido a la insuficiencia de recursos para poder expresar las restricciones concretas del problema en estos sistemas. Asimismo, la notación *OGS* en la columna de SWI-Prolog indica que se ha obtenido un error de tipo *Out Of Global Stack*, y consecuentemente, no se ha obtenido ninguna respuesta. En la columna de GNU Prolog, la notación *(SO) número* significa que en la primera ejecución del programa no se ha calculado ninguna respuesta debido a la aparición de un error de tipo *Stack Overflow*, y a que después de incrementar significativamente el correspondiente entorno de las variables, en otras ejecuciones se obtuvo una respuesta en el tiempo (promedio) indicado por *número*. La notación *RE* en la columna de SICStus Prolog indica también que no se ha obtenido una respuesta, debido a que se devolvió un error de tipo *Resource Error by Insufficient Memory*. La marca (-) en la columna de Ciao Prolog significa que se obtuvo una respuesta incorrecta para este benchmark (este hecho parece haber sido causado por un error existente en el paquete de restricciones $\mathcal{FD}$).

Como ya se ha indicado, siempre que nos ha sido posible hemos mantenido la misma formulación para todos los benchmarks en cada uno de los sistemas. Sin embargo, esto no ha sido posible en el caso del benchmark de las series mágicas.

Benchmark	$\mathcal{TOY}(\mathcal{FD})$	*PAKCS*	SICStus	SWI	GNU	Ciao
cars	0	N	0	N	0	N
equation10	20	50	10	N	3	N
equation20	30	55	15	N	4	N
magic (64)	90	150	(320) 80	N	18	N
magic (100)	220	310	(1090) 195	N	53	N
magic (150)	470	690	(3440) 465	N	(SO) 52	N
magic (200)	870	1480	(7950) 850	N	(SO) 125	N
magic (300)	1835	3610	(RE) 1820	N	(SO) 568	N
magic (400)	3420	10050	(RE) 3370	N	(SO) 1088	N
magic (500)	5510	13100	(RE) 5250	N	(SO) 1830	N
pythagoras	50	80	10	N	10	N
queens (8)	10	15	5	N	1	N
queens (16)	20	50	8	N	2	N
queens (20)	45	75	10	N	3	N
queens (24)	40	80	15	N	4	N
queens (30)	150	190	25	N	6	N
sendmore	0	5	0	N	0	N
suudoku	10	20	10	N	1	N

Cuadro B.6: $\mathcal{TOY}(\mathcal{FD})$ vs. $C(F)LP$ sistemas: etiquetado first-fail

En los sistemas $\mathcal{TOY}(\mathcal{FD})$, *PAKCS* y SICStus Prolog, este problema ha sido resuelto mediante la utilización de restricciones específicas, es decir, *count*/4, *sum*/3 y *scalar_product*/4 (véase la formulación del Ejemplo 28). Sin embargo, el sistema GNU Prolog carece de estas restricciones, y por tanto se ha usado una formulación clásica que requiere el uso de *restricciones reificadas* [CD96]. Esta formulación clásica es, en cierto sentido, diferente en $\mathcal{TOY}(\mathcal{FD})$ puesto que la reificación se aplica sólo a tipos booleanos, mientras que GNU Prolog, como en general ocurre en los lenguajes $CLP(\mathcal{FD})$, los valores booleanos $false$ y $true$ corresponden a los valores numéricos 0 y 1. Por otra parte, no ha sido posible realizar esta formulación en *PAKCS* mediante restricciones reificadas puesto que este tipo de restricciones no está disponible. Sin embargo, puesto que SICStus Prolog sí permite la utilización de restricciones reificadas, las dos formulaciones han sido consideradas en este sistema. En este caso, en la columna de SICStus y para la fila correspondiente al benchmark de las series mágicas, mostramos entre paréntesis el tiempo transcurrido usando la formulación basada en restricciones reificadas, seguido del tiempo transcurrido usando la formulación alternativa basada en el uso de restricciones específicas.

En las tablas de eficiencia, en todos aquellos casos en los que, para un sistema particular o bien para un problema concreto, no se ha conseguido obtener una forma equivalente de expresarlo (por ejemplo para *PAKCS*, SWI-Prolog o Ciao Prolog) o

Benchmark	*PAKCS*	SICStus	SWI	GNU	Ciao	$\mathcal{TOY}(\mathcal{FD})$
cars	∞	$\geq$ 1.00	∞	$\geq$ 1.00	∞	$\geq$ 5.00
equation10	2.50	0.50	∞	0.15	∞	1.00
equation20	1.83	0.50	∞	0.13	∞	1.16
magic (64)	1.66	(3.55) 0.88	∞	0.20	∞	2.94
magic (100)	1.40	(4.95) 0.88	∞	0.24	∞	4.13
magic (150)	1.46	(7.31) 0.98	∞	(∞) 0.11	∞	5.74
magic (200)	1.70	(9.13) 0.97	∞	(∞) 0.14	∞	6.86
magic (300)	1.96	(∞) 0.99	∞	(∞) 0.30	∞	10.00
magic (400)	2.93	(∞) 0.98	∞	(∞) 0.31	∞	(?)
magic (500)	2.37	(∞) 0.95	∞	(∞) 0.33	∞	(?)
pythagoras	1.60	0.20	∞	0.20	∞	1.00
queens (8)	1.50	0.50	∞	0.10	∞	1.00
queens (16)	2.50	0.40	∞	0.10	∞	9.00
queens (20)	1.66	0.22	∞	0.06	∞	89.55
queens (24)	2.00	0.37	∞	0.10	∞	208.25
queens (30)	1.26	0.16	∞	0.04	∞	7611.73
sendmore	$\geq$ 5.0	$\geq$ 1.00	∞	$\geq$ 1.00	∞	$\geq$ 1.00
suudoku	2.00	1.00	∞	0.10	∞	1.00

Cuadro B.7: Eficiencia de $\mathcal{TOY}(\mathcal{FD})$ en relación al etiquetado first-fail

Benchmark	$\mathcal{TOY}(\mathcal{FD})$	*PAKCS*	SICStus	SWI	GNU	Ciao
golomb(8)	360	350	280	N	86	N
golomb(10)	26230	27500	25730	N	8595	N
golomb(12)	5280170	5453220	5208760	N	2162863	N

Cuadro B.8: $\mathcal{TOY}(\mathcal{FD})$ vs. $C(F)LP$ sistemas: optimización de benchmarks

bien se ha devuelto un error que ha impedido la computación de una primera respuesta, o incluso se ha obtenido una respuesta pero esta es incorrecta, se ha usado el símbolo ∞. Mediante este símbolo indicamos que nuestro sistema es claramente superior a aquel sistema, puesto que $\mathcal{TOY}(\mathcal{FD})$ es capaz de proporcionar restricciones que permiten formular una solución para ese benchmark, y por tanto computar una respuesta. También, un resultado de la forma $\geq$ *x.00* indica que $\mathcal{TOY}(\mathcal{FD})$ ha computado una respuesta en 0.0 milisegundos; en estos casos, *x.00* indica que $\mathcal{TOY}(\mathcal{FD})$ es, al menos, x veces más rápido que el sistema comparado.

Benchmark	SICStus	*PAKCS*	SWI	GNU	Ciao
golomb(8)	0.77	0.97	∞	0.23	∞
golomb(10)	0.98	1.04	∞	0.32	∞
golomb(12)	0.98	1.03	∞	0.40	∞

Cuadro B.9: Eficiencia de $\mathcal{TOY}(\mathcal{FD})$ para la optimización de benchmarks

B.2.4. Análisis de los resultados

La tercera columna en los Cuadros B.5 y B.7 y la columna 2 en el Cuadro B.9 muestran que, en general, nuestra implementación se comporta de un modo muy parecido a la que se muestra para SICStus Prolog, tanto en restricciones de satisfacción como en restricciones de optimización (de hecho, esto no es sorprendente puesto que la versión actual de $\mathcal{TOY}(\mathcal{FD})$ utiliza el resolutor $\mathcal{FD}$ de SICStus Prolog) excepto para resolver ecuaciones lineales (en estos casos es entre dos y cuatro veces más lento). La razón parece estar en el proceso de transformación previo a la invocación del resolutor $\mathcal{FD}$. Las expresiones han de ser transformadas a *forma normal de cabeza*, lo que significa que sus argumentos son también transformados a *hnf* (ver Sección B.1.3). De este modo, parece haber una sobrecarga cuando las expresiones (como por ejemplo aquellas que representan ecuaciones lineales) involucran un alto número de argumentos y subexpresiones. Esta podría ser también la razón que justificaría que $\mathcal{TOY}(\mathcal{FD})$ sea más lento en la resolución del benchmark de las reinas usando el etiquetado first-fail (aunque esta lentitud no se evidencia con el etiqueta naïve).

PAKCS es entre una y tres veces más lento que nuestra implementación. Esto es bastante interesante puesto que la implementación de *PAKCS* es bastante eficiente y está también basada en la librería $\mathcal{FD}$ de SICStus Prolog. Quizás la razón de esta lentitud con respecto a $\mathcal{TOY}(\mathcal{FD})$ es que *PAKCS* implementa un modelo operacional alternativo que también soporta concurrencia, y este modelo introduce algún tipo de sobrecarga en la resolución de objetivos.

$\mathcal{TOY}(\mathcal{FD})$ también ejecuta los benchmarks razonablemente bien en comparación con otros sistemas $CLP(\mathcal{FD})$. Claramente supera tanto al resolutor de restricciones de Ciao Prolog como de SWI-Prolog, los cuales están lejos, en las versiones utilizadas, de la eficiencia de $\mathcal{TOY}(\mathcal{FD})$ en la resolución de problemas de satisfacción de restricciones. Desde un punto de vista más equitativo hemos de decir que estos resultados no pueden extrapolarse a todo el sistema Ciao Prolog y SWI-Prolog, los cuales son bastante eficientes; de hecho, el resolutor de cotas enteras de SWI-Prolog parece ser un resolutor de restricciones enteras simple no optimizado que probablemente será largamente mejorado en futuras versiones. Este mismo argumento puede ser aplicado al paquete de resolución de restricciones en dominios finitos que actualmente se encuentra integrado en el sistema Ciao Prolog, el cual sin embargo parece no estar suficientemente maduro todavía.

Con respecto al resolutor de restricciones de GNU Prolog, nuestro sistema se comporta aceptablemente bien si tenemos en cuenta que este resolutor ha mostrado una eficiencia comparable a sistemas comerciales. Excepto para el benchmark de las N-reinas (que parece comportarse de una manera particularmente optimizada para el resolutor GNU) nuestro sistema está en el mismo orden de eficiencia. Más aún, éste se comporta mejor sobre problemas escalables cuando el tamaño del problema se incrementa (por ejemplo, en el problema de las series mágicas con el etiquetado naïve). En este sentido, de nuevo con la excepción del problema de las N-reinas, siempre que la instancia del problema se incrementa, la ejecución de $\mathcal{TOY}(\mathcal{FD})$ llega a ser más cercana a la de GNU Prolog (este resultado queda confirmado tanto para restricciones de satisfacción como para restricciones de optimización).

Más aún, con respecto a la comparación con el otro sistema $CFLP(\mathcal{FD})$ que es más cercano a $\mathcal{TOY}(\mathcal{FD})$, podemos decir que *PAKCS* proporciona un pequeño conjunto de restricciones globales (exactamente cuatro) como se mencionaba al principio de esta sección, mientras que $\mathcal{TOY}(\mathcal{FD})$ también da soporte a restricciones especializadas para problemas particulares, como por ejemplo, *problemas de scheduling*. Más aún, *PAKCS* no proporciona restricciones $\mathcal{FD}$ que ayuden a los usuarios a recopilar estadísticas del proceso de resolución de restricciones como $\mathcal{TOY}(\mathcal{FD})$ hace (por ejemplo, el número de reducciones o *podas* del dominio, *entailments* detectados por una restricción, *backtracks* debidos a inconsistencias, *constraint resumptions*, etc.) los cuales son muy útiles en la práctica. Desde una postura de nuevo más equilibrada, mencionamos que *PAKCS* soporta también la *evaluación concurrente de restricciones*, la cual es también muy conveniente en la práctica y no está contemplada en $\mathcal{TOY}(\mathcal{FD})$.

Basándonos en los resultados mostrados en esta sección podemos asegurar que $\mathcal{TOY}(\mathcal{FD})$ es el primer sistema $CFLP(\mathcal{FD})$ puro que proporciona un amplio conjunto de restricciones $\mathcal{FD}$ que lo convierten en realmente competitivo comparado con otros sistemas $CLP(\mathcal{FD})$ existentes. Estos resultados nos alientan a continuar trabajando con el fin de poder mejorar los resultados en un futuro cercano mediante la introducción de otras posibles optimizaciones.

B.3. Ejemplos de otros dominios de restricciones en el sistema $\mathcal{TOY}$

Terminamos este apéndice incluyendo una breve selección de ejemplos de programas escritos en sintaxis $\mathcal{TOY}$ que hacen uso de las facilidades del sistema para programar restricciones en otros dominios del esquema genérico $CFLP(\mathcal{D})$ distintos del de dominio finito $\mathcal{FD}$, y que también han sido presentados en el Capítulo 2 de esta tesis. El propósito de esta última sección es el de resaltar la comodidad que el programador consigue al tener integrado $\mathcal{TOY}(\mathcal{FD})$ dentro del propio sistema $\mathcal{TOY}$,

de forma que sea posible trabajar fácilmente sobre otros dominios de restricciones ya implementados como el dominio de Herbrand $\mathcal{H}$ o el dominio de los números reales $\mathcal{R}$, sin tener que cambiar necesariamente ni de entorno ni de sistema. Al igual que en las secciones precedentes, el código de los programas puede ser ejecutado directamente en el sistema mediante la carga y descarga de las librerías oportunas (los comandos `/cflpr` y `/nocflpr`, respectivamente, para el caso de los números reales, de manera análoga a como se cargan y descargan directamente las librerías de restricciones de dominio finito mediante los comandos `/cflpfd` y `/nocflpfd`). De manera adicional, en algunos de los programas que se presentan a continuación se hace uso de la construcción especial `where` para la declaración de *definiciones locales*, muy usuales en muchos de los lenguajes de programación funcional.

B.3.1. Ejemplos en $\mathcal{TOY}$ de $CFLP(\mathcal{H})$-programación

El siguiente programa permite ordenar una lista de enteros mediante el método *generate & test* de permutaciones, generando a través de una función perezosa e indeterminista las permutaciones de una lista inicial de enteros y comprobando su ordenación. Por supuesto, este programa no trata de ser eficiente sino tan solo de ilustrar una técnica de programación bien conocida dentro de nuestro sistema *CFLP* usando igualdad estricta. No obstante, destacamos que el uso de una función indeterminista como generador perezoso proporciona mejores resultados que programas similares escritos en *Haskell* o en Prolog, los cuales tardan en ejecutarse más de una hora para el mismo objetivo que aparece descrito en este ejemplo.

Ordenación por permutaciones en $CFLP(\mathcal{H})$ usando igualdad estricta

```
% 'Generate & test' perezoso como un esquema de orden superior

  findSol :: (Input -> Solution) -> (Solution -> bool) -> Input ->
             Solution
  findSol Generate Test Input = Candidate <== Test Candidate
                                where Candidate = Generate Input

% Objetivo: findSol generate test input == Sol

  permSort :: [int] -> [int]
  permSort = findSol permute isSorted

% El generador computa permutaciones de una lista:

  permute []     = []
  permute [X|Xs] = [Y|permute Ys] where (Y,Ys) = split_one [X|Xs]
```

```
  split_one [X|Xs] = (X,Xs)
  split_one [X|Xs] = (Y,[X|Ys]) where (Y,Ys) = split_one Xs

% Comprobación de listas ordenadas:

  isSorted :: [int] -> bool
  isSorted []       = true
  isSorted [X]      = true
  isSorted [X,Y|Zs] = (X <= Y) /\ (isSorted [Y|Zs])

% /\ se comporta como una conjunción secuencial:

  infixr 40 /\

  (/\) :: bool -> bool -> bool
  false /\ Y = false
  true  /\ Y = Y

% Objetivo: permSort [3,11,8,10,1,14,12,5,6,9,2,7,15,13,4] == Xs
% Solución: Xs = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]
% Elapsed time in the system Toy: 8513 ms.
```

Diferencia de listas en $CFLP(\mathcal{H})$ usando restricciones de desigualdad

El siguiente $CFLP(\mathcal{H})$-programa permite computar la lista diferencia entre dos listas `Xs` e `Ys` dadas como argumentos. Para ello, se elimina de `Xs` la primera aparición de cada uno de los miembros de `Ys` y se falla el cómputo en el caso en el que algún miembro de `Ys` no aparezca en `Xs` con la misma multiplicidad (es decir, computamos la diferencia entre las dos listas interpretando ambas como multiconjuntos).

```
  infixl 50 --

  (--) :: [A] -> [A] -> [A]
  Xs -- []     = Xs
  Xs -- [Y|Ys] = (delete Y Xs) -- Ys

  delete :: A -> [A] -> [A]
  delete Y [X|Xs] = if Y == X then Xs else [X|delete Y Xs]

% 'delete Y Xs' falla si Y no aparece en Xs. La regla de 'delete' es
% código TOY para representar la CFLP(H)-regla de programa:
```

```
% delete Y [X|Xs] -> if R then Xs else [X|delete Y Xs]
%                                                  <== X == Y ->! R
%
%
% La primitiva '==' involucra restricciones de desigualdad.
% Una definición equivalente, pero menos eficiente, sería:
%
%
% delete Y [X|Xs] -> Xs              <== Y == X
% delete Y [X|Xs] -> [X|delete Y Xs] <== Y /= X
%
%
% Esta versión permite explicitar las restricciones de desigualdad.


% Objetivo:   [1,2,3,2,4] -- [2,4] == Xs
% Solución:   Xs = [1,3,2]


% Objetivo:   ("angle" -- Xs) ++ Xs == "angel"
% Soluciones: Xs = "l"; Xs = "el"; Xs = "gel"; etc.
```

Como aplicación, es posible computar permutaciones de una forma alternativa a como se ha hecho en el ejemplo anterior mediante la función `permute`.

```
  permutation :: [A] -> [A]
  permutation Xs = Ys <== Ys -- Xs == []

% Objetivo:   permutation [1,2,3] == Xs
% Soluciones: Xs == [1,2,3] ;
%             Xs == [1,3,2] ;
%             Xs == [2,1,3] ;
%             Xs == [2,3,1] ;
%             Xs == [3,1,2] ;
%             Xs == [3,2,1] ;
%             no
```

Esta alternativa no sería, sin embargo, una buena elección si se utiliza en cooperación con `permSort`, debido a que `permutation` no es un generador perezoso.

B.3.2. Ejemplo en $\mathcal{TOY}(\mathcal{R})$ de $CFLP(\mathcal{R})$-programación

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                                               %
%                          Programación in CFLP(R)                              %
%                                                                               %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Cálculo de una hipoteca (adaptado de K. Marriott y P.J. Stuckey [MS98])

% Declaración de algunos tipos sinónimos de utilidad para este problema:

type principal   = real      % el capital
type time        = real      % el plazo de devolución del préstamo,
                             % es decir, el número de pagos periódicos
type interest    = real      % el tipo de interés
type repayment   = real      % interés por un pago periódico
type balance     = real      % el balance pendiente
% Cálculo del nuevo capital NP después de un pago R:

% NP = P + P*I - R

mortgage :: (principal,time,interest,repayment) -> balance
mortgage (P, T, I, R) = P                                   <== T == 0
mortgage (P, T, I, R) = mortgage (P + P*I - R, T-1, I, R) <== T >= 1

% Esta formulación admite varios modos de uso. Algunos ejemplos serían:

% ¿Cuál es el balance correspondiente al préstamo de 1000 Euros en 10
% años a un interés del 10% y con un pago de 150 Euros por año?
%
% Objetivo: mortgage (1000, 10, 10/100, 150) == B
% Solución: B == 203.12876995000016

% ¿Cuál es el préstamo en 10 años al 10% con un pago anual de 150 Euros?
%
% Objetivo: mortgage (P, 10, 10/100, 150) == 0
% Solución: P == 921.6850658557024

% ¿Cuál es la relación que debe darse entre el capital inicial, el pago y
% el balance en un préstamo de 10 años al 10%?
%
% Objetivo: mortgage(P, 10, 10/100, R) == B
% Solución: B == 2.5937424601*P-15.937424601000002*R
%
% Es decir, una restricción lineal que permite relacionar P, R y B.
```

Apéndice C

Referencias de las publicaciones en las que se basa la tesis

El contenido de esta tesis es, en buena medida, una compilación de diversas publicaciones que relacionamos a continuación.

- [EFHR+08] S. Estévez Martín, A.J. Fernández, M.T. Hortalá González, M. Rodríguez Artalejo, F. Sáenz Pérez y R. del Vado Vírseda. *Cooperation of Constraint Domains in the $\mathcal{TOY}$ System*. En Proceedings of the *10th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming* (PPDP'08), ACM Press, páginas 258-268, 2008.

- [CRV08] R. Caballero, M. Rodríguez Artalejo y R. del Vado Vírseda. *Declarative Diagnosis of Missing Answers in Constraint Functional-Logic Programming*. En Proceedings of the *Ninth International Symposium on Functional and Logic Programming* (FLOPS'08), Springer LNCS, volumen 4989, páginas 305-321, 2008.

- [Vad07] R. del Vado Vírseda. *A Higher-Order Demand-driven Narrowing Calculus with Definitional Trees*. En Proceedings of the *4th International Colloquium on Theoretical Aspects of Computing* (ICTAC'07), Springer LNCS, volumen 4711, páginas 169-184, 2007.

- [CRV07] R. Caballero, M. Rodríguez Artalejo y R. del Vado Vírseda. *Declarative Debugging of Missing Answers in Constraint Functional-Logic Programming (poster)*. En Proceedings of the *23th International Conference on Logic Programming* (ICLP'07), Springer LNCS, volumen 4670, páginas 425-427, 2007.

- [EFHR+07b] S. Estévez Martín, A.J. Fernández, M.T. Hortalá González, M. Rodríguez Artalejo y R. del Vado Vírseda. *A Fully Sound Goal Solving Calculus for the Cooperation of Solvers in the CFLP Scheme*. En Proceedings of the *15th International Workshop on Functional and (Constraint) Logic Programming* (WFLP'06), Elsevier ENTCS, volumen 177, páginas 235-252, 2007.

- [EFHR+07a] S. Estévez Martín, A.J. Fernández, M.T. Hortalá González, M. Rodríguez Artalejo, F. Sáenz Pérez y R. del Vado Vírseda. *A Proposal for the Cooperation of Solvers in Constraint Functional Logic Programming*. En Proceedings of the *6th Spanish Conference on Programming and Computer Languages* (PROLE'06), Elsevier ENTCS, volumen 188, páginas 37-51, 2007.

- [FHSV07] A.J. Fernández, M.T. Hortalá González, F. Sáenz Pérez y R. del Vado Vírseda. *Constraint Functional Logic Programming over Finite Domains*. Journal of *Theory and Practice of Logic Programming* (TPLP), Cambridge University Press, volumen 7 (5), páginas 537-582, 2007. Disponible en `http://arXiv.org/abs/cs/0601071`.

- [LRV07] F.J. López Fraguas, M. Rodríguez Artalejo y R. del Vado Vírseda. *A New Generic Scheme for Functional Logic Programming with Constraints*. Journal of *Higher-Order and Symbolic Computation* (HOSC), volumen 20 (1/2), páginas 73-122, 2007.

- [CRV06b] R. Caballero, M. Rodríguez Artalejo y R. del Vado Vírseda. *Algorithmic Debugging of Wrong Answers in Constraint Functional-Logic Programming (Technical Report)*. Informe Técnico DSIC 03-06 del Departamento de Sistemas Informáticos y Computación de la Universidad Complutense de Madrid, 2006.

- [CRV06a] R. Caballero, M. Rodríguez Artalejo y R. del Vado Vírseda. *Declarative Diagnosis of Wrong Answers in Constraint Functional-Logic Programming (poster)*. En Proceedings of the *22th International Conference on Logic Programming* (ICLP'06), Springer LNCS, volumen 4079, páginas 421-422, 2006.

- [EV05] S. Estévez Martín and R. del Vado Vírseda. *Designing an Efficient Computation Strategy in CFLP($\mathcal{FD}$) Using Definitional Trees*. En Proceedings of the ACM SIGPLAN *International Workshop on Curry and Functional Logic Programming* (WCFLP'05), ACM Press, páginas 23-31, 2005.

- [Vad05] R. del Vado Vírseda. *Declarative Constraint Programming with Definitional Trees*. En Proceedings of the *5th International Workshop on Frontiers of Combining Systems* (FroCoS'05), Springer LNCS/LNAI, volumen 3717, páginas 184-199, 2005.

- [LRV04b] F.J. López Fraguas, M. Rodríguez Artalejo y R. del Vado Vírseda. *A Lazy Narrowing Calculus for Declarative Constraint Programming*. En Proceedings of the ACM SIGPLAN *Conference on Principles and Practice of Declarative Programming* (PPDP'04), ACM Press, páginas 43-54, 2004.

- [LRV04a] F.J. López Fraguas, M. Rodríguez Artalejo y R. del Vado Vírseda. *Constraint Functional Logic Programming Revisited*. En Proceedings of the *5th International Workshop on Rewriting Logic and its Applications* (WRLA'04), Elsevier ENTCS, volumen 117, páginas 5-50, 2005.

- [Vad03b] R. del Vado Vírseda. *A Demand-driven Narrowing Calculus with Overlapping Definitional Trees*. En Proceedings of the ACM SIGPLAN *Conference on Principles and Practice of Declarative Programming* (PPDP'03), ACM Press, páginas 213-227, 2003.

- [Vad03a] R. del Vado Vírseda. *A Demand Narrowing Calculus with Overlapping Definitional Trees*. En Proceedings of the *12th International Workshop on Functional and (Constraint) Logic Programming* (WFLP'03), Germán Vidal (Ed.), Informe Técnico DSIC-II/13/03 del Departamento de Sistemas Informáticos y Computación de la Universidad Politécnica de Valencia, ISBN 84-96221-02-4, páginas 184-197, 2003. **(Best Newcomer Award)**

- [Vad02] R. del Vado Vírseda. *Estrategias de Estrechamiento Perezoso*. Trabajo de Investigación de Tercer Ciclo, Departamento de Sistemas Informáticos y Programación, Universidad Complutense de Madrid, Septiembre de 2002.

Bibliografía

Bibliografía

[AN86] H. Aït-kaci y R. Nasr. *LOGIN: A logic programming language with built-in inheritance*. Journal of Logic Programming, 3(3): 185–215, 1986.

[AN89] H. Aït-kaci y R. Nasr. *Integrating Logic and Functional Programming*. Lisp and Symbolic Computation, 2, pp. 51–89, 1989.

[AP93] H. Aït-kaci y A. Podelski. *Towards a meaning of LIFE*. Journal of Logic Programming, 16(3): 195–234, 1993.

[AP94] H. Aït-Kaci y A. Podelski. *A feature constraint system for logic programming with entailment*. Theoretical Computer Science 122, pp. 263–283, 1994.

[ALN87] H. Aït-kaci, P. Lincoln y R. Nasr. *Le Fun: logic, equations and functions*. En 1987 Symposium on Logic Programming (SLP'87). IEEE-CS, pp. 17–23, San Francisco, California, 1987.

[ACF02] M. Alpuente, F.J. Correa y M. Falaschi. *A Debugging Scheme for Functional Logic Programs*. En Proc. of International Workshop on Functional and (Constraint) Logic Programming (WFLP'01), Selected Papers, Elsevier ENTCS 64, pp. 18–55, 2002.

[ABCF03] M. Alpuente, D. Ballis, F.J. Correa y M. Falaschi. *Automated Correction of Functional Logic Programs*. En Proc. of ESOP'2003, Springer LNCS 2618, pp. 54–68, 2003.

[Ant92] S. Antoy. *Definitional trees*. En Proc. Int. Conf. on Algebraic and Logic Programming (ALP'92), Springer LNCS 632, pp. 143–157, 1992.

[Ant97] S. Antoy. *Optimal non-deterministic functional logic computations*. En Proc. of ALP'97, Springer LNCS 1298, pp. 16–30, 1997.

[Ant01] S. Antoy. *Constructor-based conditional narrowing*. En Proc. PPDP'01, ACM Press, pp. 199–206, 2001.

[AB07] S. Antoy y B. Brassel. *Computing with Subspaces*. En Proc. of the 9th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming (PPDP'07). ACM Press, pp. 121–130, 2007.

[AEH94] S. Antoy, R. Echahed y M. Hanus. *A Needed Narrowing Strategy*. En Proc. ACM Symp. on Principles of Programming Languages (POPL'94), Portland, ACM Press, pp. 268–279, 1994.

[AEH00] S. Antoy, R. Echahed y M. Hanus. *A Needed Narrowing Strategy*. Journal of the ACM 74 (4), pp. 776–822, 2000.

[Apt90] K.R. Apt. *Introduction to Logic Programming*. En J. van Leeuwen (ed.), *Handbook of Theoretical Computer Science*, Vol. B, Capítulo 10, Elsevier y The MIT Press, pp. 493–574, 1990.

[Apt03] K.R. Apt. *Principles of constraint programming*. Cambridge University Press, 2003.

[AG94] K.R. Apt y M. Gabbrielli. *Declarative Interpretations Reconsidered*. En Proc. Int. Conf. on Logic Programming (ICLP'94), Santa Margherita Ligure, the MIT Press, pp. 74–89, 1994.

[AGL94] P. Arenas, A. Gil y F.J. López-Fraguas. *Combining Lazy Narrowing with Disequality Constraints*. En Proc. Int. Symp. on Programming Language Implementation and Logic Programming (PLILP'94), Springer LNCS 844, pp. 385–399, 1994.

[AHLU96] P. Arenas, M.T. Hortalá, F.J. López-Fraguas y E. Ullán. *Real constraints within a functional logic language*. En P. Lucio, M. Martelli y M. Navarro, editores, *Joint Conference on Declarative Programming (APPIA-GULP-PRODE'96)*, Donostia-San Sebastián, España, Julio 1996.

[ALR98] P. Arenas, F.J. López-Fraguas y M. Rodríguez-Artalejo. *Embedding Multiset Constraints into a Lazy Functional Logic Language*. En Proc. Int. Symp. on Programming Language Implementation and Logic Programming (PLILP'98), situado conjuntamente con 6th Int. Conf. on Algebraic and Logic Programming (ALP'98), Pisa, Springer LNCS 1490, pp. 429–444, 1998.

[ALR99] P. Arenas, F.J. López-Fraguas y M. Rodríguez-Artalejo. *Functional plus Logic Programming with Built-in and Symbolic Constraints*. En Proc. Int. Conf. on Principles and Practice of Declarative Programming (PPDP'99), París, Springer LNCS 1702, pp. 152–169, 1999.

[ALR07] P. Arenas, F.J. López-Fraguas y M. Rodríguez-Artalejo. *TOY. A Multiparadigm Declarative Language. Versión* 2,3,1, 15 de Octubre, 2007. R. Caballero y J. Sánchez (Eds.), Disponible en `http://toy.sourceforge.net`.

[AR97a] P. Arenas y M. Rodríguez-Artalejo. *A Semantic Framework for Functional Logic Programming with Algebraic Polymorphic Types*. En Proc. Int. Joint Conference on Theory and Practice of Software Development (TAPSOFT'97), Springer LNCS 1214, pp. 453–464, 1997.

[AR97b] P. Arenas y M. Rodríguez-Artalejo. *A Lazy Narrowing Calculus for Functional Logic Programming with Algebraic Polymorphic Types*. En Proc. Int. Symp. on Logic Programming (ILPS'97), The MIT Press, pp. 53–68, 1997.

[AR01] P. Arenas y M. Rodríguez-Artalejo. *A general framework for lazy functional logic programming with algebraic polymorphic types*. Theory and Practice of Logic Programming 1(2), pp. 185–245, 2001.

[BN98] F. Baader y T. Nipkow. *Term Rewriting and All That*. Cambridge University Press, 1998.

[Bac95] R. Backofen. *A Complete Axiomatization of a Theory with Feature and Arity Constraints*. Journal of Logic Programming 24(1&2), pp. 37–71, 1995.

[BS95] R. Backofen y G. Smolka. *A complete and recursive feature theory*. Theoretical Computer Science 146, pp. 243–268, 1995.

[Bar03] F. Barber. *Constraint satisfaction problems*. Inteligencia Artificial. Revista Iberoamericana de IA, (20), 2003.

[Bar84] H.P. Barendregt. *The lambda calculus - Its syntax and semantics*. Studies in Logic and the Foundations of Mathematics, 103. North-Holland, Netherlands, 1984. Second Revised Edition.

[Bar90] H.P. Barendregt. *Functional Programming and Lambda Calculus*. En J. van Leeuwen (ed.), *Handbook of Theoretical Computer Science*, Vol. B, Capítulo 7, Elsevier y The MIT Press, pp. 321–363, 1990.

[BL86] M. Bellia y G. Levi. *The Relation between Logic and Functional Languages*. Journal of Logic Programming, vol. 3, pp. 217–236, 1986.

[BE95] D. Bert y R. Echahed. *On the Operational Semantics of the Algebraic and Logic Programming Language LPG*. En Recent Trends in Data Type Specifications, Springer LNCS 906, pp. 132–152, 1995.

[Bir98] R. Bird. *Introduction to Functional Programming using Haskell*, 2ª edición, Prentice Hall Europe, 1998.

[BGM88] P. Bosco, E. Giovannetti y C. Moiso. *Narrowing vs. SLD-resolution*. Theoretical Computer Science, 59, pp. 3–23, 1988.

[BGLM94] A. Bossi, M. Gabbrielli, G. Levi y M. Martelli. *The s-Semantics Approach: Theory and Applications*. Journal of Logic Programming 19&20, pp. 149–197, 1994.

[Bou93] A. Boudet. *Combining Unification Algorithms*. Journal of Symbolic Computation, 16, pp. 597–626, 1993.

[BDM97] J. Boye, W. Drabent y J. Maluszyński. *Declarative Diagnosis of Constraint Programs: an Assertion-based Approach*. En Proc. of the 3rd. Int'l Workshop on Automated Debugging-AADEBUG'97, U. of Linkoping Press, pp. 123–141, Linkoping, Suecia, Mayo 1997. DiSCiPl Delieverable D.WP2.2.M1.1-2, Septiembre 1997.

[BFHH+07] B. Brassel, S. Fischer, M. Hanus, F. Huch y G. Vidal. *Lazy call-by-value evaluation*. En Proc. of the International Conference on Functional Programming (ICFP'07). ACM Press, pp. 265–276, 2007.

[BHH04] B. Brassel, M. Hanus y F. Huch. *Encapsulating non-determinism in functional logic computations*. Journal of Functional and Logic Programing, 2004.

[BHHV04] B. Brassel, M. Hanus, F. Huch y G. Vidal. *A semantics for tracing declarative multi-paradigm programs*. En Proc. of the 6th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP'04). ACM Press, pp. 179–190, 2004.

[BB94] W.L. Buntine y H.J. Bürckert. *On solving equations and disequations*. Journal of the ACM 41(4), 591–629, 1994.

[Cab04] R. Caballero. *Técnicas de diagnóstico y depuración declarativa para lenguajes lógico-funcionales*. PhD Tesis, Departamento de Sistemas Informáticos y Programación, Universidad Complutense de Madrid, 2004.

[Cab05] R. Caballero. *A Declarative Debugger of Incorrect Answers for Constraint Functional-Logic Programs*. En Proc. WCFLP'05, ACM SIGPLAN, pp. 8–13, 2005.

[CGS07] R. Caballero, Y. García-Ruiz y F. Sáenz-Pérez. *A new proposal for debugging datalog programs*. En Proc. of International Workshop on Functional and (Constraint) Logic Programming (WFLP'07), Elsevier ENTCS 216, pp. 79–92, 2008.

[CLR01] R. Caballero, F.J. López-Fraguas y M. Rodríguez-Artalejo. *Theoretical Foundations for the Declarative Debugging of Lazy Functional Logic Programs*. En Proc. of the 5th International Symposium on Functional and Logic Programming (FLOPS'01), Springer LNCS 2024, pp. 170–184, 2001.

[CR02] R. Caballero y M. Rodríguez-Artalejo. *A Declarative Debugging System for Lazy Functional Logic Programs*. Electronic Notes in Theoretical Computer Science 64, 63 páginas, 2002.

[CR04] R. Caballero y M. Rodríguez-Artalejo. *$\mathcal{DDT}$: A Declarative Debugging Tool for Functional Logic Languages*. En Proc. of the 7th International Symposium on Functional and Logic Programming (FLOPS'2004), Springer LNCS 2988, pp. 70–84, 2004.

[CRV06a] R. Caballero, M. Rodríguez-Artalejo y R. del Vado-Vírseda. *Declarative diagnosis of wrong answers in constraint functional-logic programming (poster)*. En Proc. of the Twenty Second International Conference on Logic Programming (ICLP 2006), Springer LNCS 4079, pp. 421–422, 2006.

[CRV06b] R. Caballero, M. Rodríguez-Artalejo y R. del Vado-Vírseda. *Algorithmic Debugging of Wrong Answers in Constraint Functional-Logic Programming (Technical Report)*. Informe Técnico 03-06. Departamento de Sistemas Informáticos y Programación, Universidad Complutense de Madrid, 2006.

[CRV07] R. Caballero, M. Rodríguez-Artalejo y R. del Vado-Vírseda. *Declarative debugging of missing answers in constraint functional-logic programming (poster)*. En Proc. of 23th International Conference on Logic Programming (ICLP 2007), Springer LNCS 4670, pp. 425–427, 2007.

[CRV08] R. Caballero, M. Rodríguez-Artalejo y R. del Vado-Vírseda. *Declarative diagnosis of missing answers in constraint functional-logic programming*. En proceedings of FLOPS'08, Springer LNCS 4989, 2008.

[CZ92] R. Caferra y N. Zabel. *A method for simoultaneous search for refutations and models by equational constraint solving*. Journal of Symbolic Computation, 13(6), pp. 613-642, 1992.

[COC97] M. Carlsson, G. Ottosson y B. Carlson. *An open-ended finite domain constraint solver*. En Proc. of the 9th International Symposium on Programming Languages: Implementations, Logics and Programs (PLILP'97). Springer LNCS 1292, pp. 191–206, 1997.

[CD08] O. Chitil y T. Davie. *Comprehending Finite Maps for Algorithmic Debugging of Higher-Order Functional Programs*. En Proceedings of the *10th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming* (PPDP'08), ACM Press, pp. 205–216, 2008.

[CL07] O. Chitil y Y. Luo. *Structure and properties of traces for functional programas*. En Proc. of the 3rd International Workshop on Term Graph Rewriting (Termgraph 2006), Elsevier ENTCS 176(1), pp. 39–63, 2007.

[Cla79] K.L. Clark. *Predicate logic as a computational formalism*. Research Report DOC 79/59, Imperial College, Department of Computing, London, 1979.

[CLL04] J.M. Cleva, J. Leach y F.J. López-Fraguas. *A logic programming approach to the verification of functional-logic programs*. En Proc. Sixth ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP'04). ACM Press, pp. 9–19, 2004.

[CLIP97] The CLIP Group. *Program Assertions. The CIAO System Documentation Series*. Technical Report CLIP4/97.1, Facultad de Informática, UPM, Agosto, 1997.

[CM87] W.F. Clocksin y C.S. Mellish. *Programación en Prolog*. Edición española: Editorial Gustavo Gili S.A. Barcelona, 1987.

[CD96] P. Codognet y D. Diaz. *Compiling constraints in clp(FD)*. Journal of Logic Programming, vol. 27 (3), pp. 185–226, 1996.

[Col82] A. Colmerauer. *Prolog and Infinite Trees*. En K.L. Clark y S.A. Tärnlud (eds.) Logic Programming, Academic Press, pp. 153–172, 1982.

[Col84] A. Colmerauer. *Equations and Inequations on Finite and Infinite Trees*. En Proc. of the 2nd International Conference on Fifth Generation Computer Systems, pp. 85–89, 1984.

[Col90] A. Colmerauer. *An introduction to PROLOG III*. Communications of the ACM (CACM) 33,7, pp. 69–90, 1990.

[CR96] A. Colmerauer y P. Roussel. *The birth of Prolog*. En *History of Programming Languages*, edited by Thomas J. Bergin y Richard G. Gibson, ACM Press/Addison-Wesley, 1996.

[CLMV99] M. Comini, G. Levi, M.C. Meo y G. Vitiello. *Abstract diagnosis*. Journal of Logic Programming 39 (1–3): 43–93, 1999.

[Com91] H. Comon. *Disunification: A survey*. En *Computational Logic - Essays in Honor of Alan Robinson*. MIT Press, pp. 322–359, 1991.

[CL89] H. Comon y P. Lescanne. *Equational Problems and Disunification*. Journal of Symbolic Computation, volume 7, pp. 371–425, 1989.

[CC77] P. Cousot y R. Cousot. *Abstract Interpretation: a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints*. En Fourth ACM Symposium on Principles of Programming Languages, pp. 238–252, 1977.

[CC92] P. Cousot y R. Cousot. *Abstract Interpretation and Applications to Logic Programs*. Journal of Logic Programming 13 (2&3): 103–179, 1992.

[CMW93] J.N. Crossley, L. Mandel y M. Wirsing. *Untyped Constrained Lambda Calculus*. Technical Report 9318, Institut fur Informatic, Ludwig-Maximilians-Univ., Munchen, Septiembre 1993.

[CMW96] J.N. Crossley, L. Mandel y M. Wirsing. *First-order constrained lambda calculus*. En F. Baader y K.U. Schulz, editores, 1st International Workshop on Frontiers of Combining Systems (FroCoS'96), volumen 3 of *Applied Logic Series*, pp. 339–356, Munich, Alemania, Marzo 1996. Kluwer Academic Publishers.

[DM82] L. Damas y R. Milner. *Principal type-schemes for functional programs*. En POPL'82. ACM Press, pp. 207–212, 1982.

[DG89] J. Darlington y Y.K. Guo. *Constraint Functional Programming*. Informe técnico, Imperial College, Noviembre 1989.

[DG91] J. Darlington y Y.K. Guo. *Constraint Equational Deduction*. En Proc. of 2nd Int. Workshop on Conditional and Typed Rewriting Systems (CTRS'90), Springer LNCS 516, pp. 11–14, 1991.

[DGP92a] J. Darlington, Y.K. Guo y H. Pull. *Introducing Constraint Functional Logic Programming*. En PHOENIX Seminar and Workshop on Declarative Programming (DP'91), Springer Workshops in Computing, pp. 20–34, 1992.

[DGP92b] J. Darlington, Y.K. Guo y H. Pull. *A New Perspective on the Integration of Functional and Logic Languages*. En Proc. of the Int. Conf. on Fifth Generation Computer Systems (FGCS'92), IOS Press, pp. 682–693, 1992.

[Dech03] R. Dechter. *Constraint processing*. Morgan Kaufmann, 2003.

[DL86] D. DeGroot y G. Lindstrom. *Logic Programming: Functions, Relations and Equations*. Prentice-Hall, Englewood Cliffs, 1986.

[DHM00] P. Deransart, M. Hermenegildo y J. Maluszynski. *Analysis and Visualization tools for Constraint Programming: Constraint Debugging*. Springer LNCS 1870, Capítulo 5, pp. 151–174, 2000.

[DJ90] N. Dershowitz y J.P. Jouannaud. *Rewrite Systems*. En J. van Leeuwen (ed.), Handbook of Theoretical Computer Science, Vol. B, Capítulo 6, Elsevier y The MIT Press, pp. 243–320, 1990.

[DO90] N. Dershowitz y M. Okada. *A Rationale for Conditional Equational Programming*. Theoretical Computer Science 75, pp. 111–138, 1990.

[DP88] N. Dershowitz y A. Plaisted. *Equational Programming*. Machine Intelligence, 11, pp. 21–56, 1988.

[DC01] D. Diaz y P. Codognet. *Design and Implementation of the GNU Prolog System*. Journal of Functional and Logic Programming, vol. 6, 2001.

[DSvH88] M. Dincbas, H. Simonis y P. Van Hentenryck. *Solving the Car-Sequencing Problem in Constraint Logic Programming*. En Proc. of the 8th European Conference on Artificial Intelligence (ECAI'88). Yves Kodratoff (ed.), Pitmann Publishing, pp. 290–295, 1988.

[DNM89] W. Drabent, S. Nadjm-Tehrani y J. Maluszyński. *Algorithmic Debugging with Assertions*. En Harvey Abramson y M.H. Rogers, editores, Metaprogramming in Logic Programming, pp. 383–398. The MIT Press, Cambridge, 1989.

[EFHR+07a] S. Estévez-Martín, A.J. Fernández, M.T. Hortalá-González, M. Rodríguez-Artalejo, F. Sáenz-Pérez y R. del Vado-Vírseda. *A Proposal for the Cooperation of Solvers in Constraint Functional Logic Programming*. En Proceedings of the 6th Spanish Conference on Programming and Computer Languages (PROLE'06), Elsevier ENTCS 188, pp. 37–51, 2007.

[EFHR+07b] S. Estévez-Martín, A.J. Fernández, M.T. Hortalá-González, M. Rodríguez-Artalejo y R. del Vado-Vírseda. *A Fully Sound Goal Solving Calculus for the Cooperation of Solvers in the CFLP Scheme*. En Proceedings of the 15th International Workshop on Functional and (Constraint) Logic Programming (WFLP'06), Elsevier ENTCS 177, pp. 235–252, 2007.

[EFHR+08] S. Estévez-Martín, A.J. Fernández, M.T. Hortalá-González, M. Rodríguez-Artalejo, F. Sáenz-Pérez y R. del Vado-Vírseda. *Cooperation of Constraint Domains in the $\mathcal{TOY}$ System*. En Proceedings of the *10th International ACM SIGPLAN Symposium on Principles and Practice of Declarative Programming* (PPDP'08), ACM Press, pp. 258–268, 2008.

[EV05] S. Estévez-Martín y R. del Vado-Vírseda. *Designing an Efficient Computation Strategy in CFLP($\mathcal{FD}$) Using Definitional Trees*. En Proc. of the International Workshop on Curry and Functional Logic Programming (WCFLP 2005), pp. 23-31, 2005.

[FLMP89] M. Falaschi, G. Levi, M. Martelli y C. Palamidessi. *Declarative Modeling of the Operational Behavior of Logic Languages*. Theoretical Computer Science 69(3). pp. 289–318, 1989.

[FLMP93] M. Falaschi, G. Levi, M. Martelli y C. Palamidessi. *A Model-theoretic Reconstruction of the Operational Semantics of Logic Programs*. Information and Computation 102(1). pp. 86–113, 1993.

[Fay79] M.J. Fay. *First-Order Unification in an Equational Theory*. En Proc. Workshop on Automated Deduction (CADE'79), Academic Press, pp. 161–177, 1979.

[Fer92] M. Fernandez. *Narrowing based procedures for equational disunification*. Revista *Applicable Algebra in Engineering, Communication and Computing*, volumen 3(1), pp. 1–26, 1992.

[FHS02] A.J. Fernández, M. T. Hortalá-González y F. Sáenz-Pérez. *A Constraint Functional Logic Language for Solving Combinatorial Problems*. En *Research and Development in Intelligent Systems XIX*, Springer BCS Conference Series, pp. 337–350, 2002.

[FHS03a] A.J. Fernández, M.T. Hortalá-González y F. Sáenz Pérez. *Solving Combinatorial Problems with a Constraint Functional Logic Language*. En Proc. 5th International Symposium on Principles and Practical Aspects of Declarative Languages (PADL'2003), Springer LNCS 2562, pp. 320–338, 2003.

[FHS03b] A.J. Fernández, M.T. Hortalá-González y F. Sáenz Pérez. $\mathcal{TOY}(\mathcal{FD})$*: Sketch of Operational Semantics*. En Proc. 9th International Conference on Principles and Practice of Constraint Programming (CP'03), Springer LNCS 2833, pp. 827–831, 2003.

[FHS03c] A.J. Fernández, M.T. Hortalá-González y F. Sáenz-Pérez. $\mathcal{TOY}(\mathcal{FD})$*: Version 0.8 User Manual*, 27 de Octubre, 2003. Sistema y documentación disponible en `http://toy.sourceforge.net`.

[FHSV07] A.J. Fernández, M.T. Hortalá-González, F. Sáenz-Pérez y R. del Vado-Vírseda. *Constraint Functional Logic Programming over Finite Domains*. Revista *Theory and Practice of Logic Programming* (TPLP), Cambridge University Press, vol. 7 (5), pp. pp. 537–582, 2007. Disponible en `http://arXiv.org/abs/cs/0601071`.

[Fer87] G. Ferrand. *Error Diagnosis in Logic Programming, an Adaptation of E.Y. Shapiro's Method*. Journal of Logic Programming 4(3), pp. 177–198, 1987.

[FLT03] G. Ferrand, W. Lesaint y A. Tessier. *Towards declarative diagnosis of constraint programs over finite domains*. ArXiv Computer Science e-prints, 2003.

[FH87] A.J. Field y P.G. Harrison. *Functional Programming*. Addison-Wesley, Reading, MA, 1987.

[Fri85] L. Fribourg. *SLOG: a logic programming language interpreter based on clausal superposition and rewriting*. En Proc. of Second IEEE Int'l Symp. on Logic Programming, pp. 172–185, 1985.

[FSKG92] P. Fritzson, N. Shahmehri, M. Kamkar y T. Gyimothy. *Generalized algorithmic debugging and testing* ACM Letters on Programming Languages and Systems (LOPLAS) archive Volumen 1 , Issue 4 (Diciembre 1992), pp. 303–322, 1992.

[Fru98] T. Frühwirth. *Theory and Practice of Constraint Handling Rules*. Journal of Logic Programming, Special Issue on Constraint Logic Programming, Volumen 37, número 1-3, pp. 95–138, 1998.

[GL91] M. Gabbrielli y G. Levi. *Modeling Answer Constraints in Constraint Logic Programs*. En Proc. of the Eigth Int. Conf. on Logic Programming (ICLP'91), The MIT Press, pp. 238–252, 1991.

[GDL95] M. Gabbrielli, G.M. Dore y G. Levi. *Observable Semantics for Constraint Logic Programs*. Journal of Logic and Computation 5 (2), pp. 133–171, 1995.

[GS89] J. Gallier y W. Snyder. *Complete Sets of Transformations for General E-unification*. Theoretical Computer Science 67, pp. 203–260, 1989.

[GLMP91] E. Giovannetti, G. Levi, C. Moiso y C. Palamidessi. *Kernel-LEAF: A Logic plus Functional Language*. Journal of Computer and System Sciences 42(2), pp. 139–185, 1991.

[GNU05] GNU Prolog. `http://pauillac.inria.fr/~diaz/gnu-prolog/`, 2005.

[GHLR96] J.C. González-Moreno, M.T. Hortalá-González, F.J. López-Fraguas y M. Rodríguez-Artalejo. *A Rewriting Logic for Declarative Programming*. En Proc. European Symp. on Programming (ESOP'96), Springer LNCS 1058, pp. 156–172, 1996.

[GHLR99] J.C. González-Moreno, M.T. Hortalá-González, F.J. López-Fraguas y M. Rodríguez-Artalejo. *An Approach to Declarative Programming Based on a Rewriting Logic*. Journal of Logic Programming 40(1), pp. 47–87, 1999.

[GHR97] J.C. González-Moreno, M.T. Hortalá-González y M. Rodríguez-Artalejo. *A Higher-Order Rewriting Logic for Functional Logic Programming*. En Proc. Int. Conf. on Logic Programming, The MIT Press, pp. 153–167, 1997.

[GHR01] J.C. González-Moreno, M.T. Hortalá-González y M. Rodríguez-Artalejo. *Polymorphic Types in Functional Logic Programming*. En FLOPS'99 special issue of the Journal of Functional and Logic Programming, 2001.

[GMB01] L. Granvilliers, E. Monfroy y F. Benhamou. *Cooperative solvers in constraint programming: a short introduction*. En Workshop on Cooperative Solvers in Constraint Programming, ALP Newsletter 14 (2), 2001.

[GS90] C.A. Gunter y D. Scott. *Semantic Domains*. En J. van Leeuwen (ed.), *Handbook of Theoretical Computer Science*, Elsevier y The MIT Press, Vol. B, Capítulo 6, pp. 633–674, 1990.

[GL90] Y.K. Guo y H. Lock. *A Classification Scheme for Declarative Programming Languages*. GMD-Studien 182, 1990.

[Ham96] M. Hamana. *Algebraic Semantics For Higher-Order Functional-Logic Programming*. En Proc. of 2nd Fuji International Workshop on Functional and Logic Progamming. World Scientific, 1996.

[Han90] M. Hanus. *Compiling Logic Programs with Equality*. En Proc. of 2nd Int'l Workshop on Programming Language Implementation and Logic Programming, Springer LNCS 456, pp. 387–401, 1990.

[Han94a] M. Hanus. *Combining Lazy Narrowing with Simplification*. En Proc. of 6th Int'l Symp. on Programming Language Implementation and Logic Programming (PLILP'94), Springer LNCS 844, pp. 370–384, 1994.

[Han94b] M. Hanus. *The Integration of Functions into Logic Programming: From Theory to Practice*. Journal of Logic Programming 19&20, pp. 583–628, 1994.

[Han97] M. Hanus. *A Unified Computation Model for Functional and Logic Programming*. En Proc. 24st ACM Symposium on Principles of Programming Languages (POPL'97), ACM Press, pp. 80–93, 1997.

[Han06] M. Hanus. *Curry: an Integrated Functional Logic Language, Version 0.8.2*, 28 de Marzo, 2006. `http://www-i2.informatik.uni-kiel.de/~curry/`.

[Han07a] M. Hanus. *Multi-paradigm Declarative Languages*. En Proc. of the International Conference on Logic Programming (ICLP'07), Springer LNCS 4670, pp. 45–75, 2007.

[Han07b] M. Hanus. *PAKCS 1.9.1, User Manual (Version of May 7, 2008)*. The Portland Aachen Kiel Curry System. Disponible en la dirección `http://www.informatik.uni-kiel.de/~pakcs/`.

[HK01] M. Hanus y J. Koj. *An Integrated Development Environment for Declarative Multi-Paradigm Programming*. A. Kusalik Ed., proceedings of the Eleventh Workshop on Logic Programming Environments, 2001.

[HKM97] M. Hanus, H. Kuchen y J.J. Moreno-Navarro. *Curry: A Truly Functional Logic Language*. En ILPS'95 Workshop on Visions for the Future of Logic Programming. Proc. 24st ACM Symposium on Principles of Programming Languages (POPL'97), ACM Press, pp. 80–93, 1997.

[HP99] M. Hanus y C. Prehofer. *Higher-order narrowing with definitional trees*. Journal of Functional Programming, 9(1): 33–75, 1999.

[HE80] R. Haralick y G. Elliot. *Increasing tree search efficiency for constraint satisfaction problems*. Journal of Artificial Intelligence, vol. 14, pp. 263–313, 1980.

[HSW95] M. Henz, G. Smolka y J. Würtz. *Object-oriented concurrent constraint programming in Oz*. En V. Saraswat y P.V. Hentenryck (eds.), *Principles and Practice of Constraint Programming*, The MIT Press, Capítulo 2, pp. 27–48, 1995.

[HPB98] M. Hermenegildo, G. Puebla y F. Bueno. *Using Global Analysis, Partial Specifications, and an Extensible Assertion Language for Program Validation and Debugging*. Technical Report CLIP8/98.0. Facultad de Informática. Universidad Politécnica de Madrid, 1998.

[HPBL02] M. Hermenegildo, G. Puebla, F. Bueno y P. López-García. *Abstract Verification and Debugging of Constraint Logic Programs*. En Proc. Int. Workshop on Constraint Solving and Constraint Logic Programming 2002, pp. 1–14, 2002.

[Hin69] R. Hindley. *The principal type-scheme of an object in combinatory logic*. Trans. Amer. Math. Soc 146, pp. 29–60, 1969.

[Hon92] H. Hong. *CLP(CF): Constraint Logic Programming over Complex Functions*. Technical Report 94-09, RISC-Linz, Castle of Hagenberg, Austria, 1992.

[Hon94] H. Hong. *Confluency of Cooperative Constraint Solvers*. Technical Report 94-08, RISC-Linz, Castle of Hagenberg, Austria, 1994.

[HS88] M. Höhfeld y G. Smolka. *Definite Relations over Constraint Languages*. LILOG-Report 53, IBM Germany, 1988.

[Hud89] P. Hudak. *Conception, Evolution and Application of Functional Programming Languages*. ACM Computing Surveys, 21, pp. 359–411, 1989.

[Hul80] J.M. Hullot. *Canonical Forms and Unification*. En Proc. Conf. on Automated Deduction (CADE'80), Springer LNCS 87, pp. 318–334, 1980.

[Hus88] H. Hussmann. *Nichtdeterministische Algebraische Spezifikationen*. PhD Tesis, University of Passau, 1988.

[Hus92] H. Hussmann. *Nondeterministic Algebraic Specifications and Nonconfluent Term Rewriting*. Journal of Logic Programming 12, pp. 237–255, 1992.

[Hus93] H. Hussmann. *Non-determinism in Algebraic Specifications and Algebraic Programs*. Birkhäuser Verlag, 1993.

[IMS01] T. Ida, M. Marin y T. Suzuki. *Higher-Order Lazy Narrowing Calculus: A Solver for Higher-Order Equations*. En Proc. of EUROCAST'01, Springer LNCS 2178, pp. 479–493, 2001.

[JL87] J. Jaffar y J.L. Lassez. *Constraint Logic Programming*. En Proc. ACM Symp. on Principles of Programming Languages (POPL'87), ACM Press, pp. 111–119, 1987.

[JM94] J. Jaffar y M.J. Maher. *Constraint Logic Programming: A Survey*. Journal of Logic Programming 19&20, pp. 503–581, 1994.

[JMMS98] J. Jaffar, M.J. Maher, K. Marriott y P. Stuckey. *Semantics of constraints logic programs*. Journal of Logic Programming 37, 1-3, pp. 1–46, 1998.

[JMSY92] J. Jaffar, S. Michaylov, P.J. Stuckey y R.H.C. Yap. *The $CLP(\mathcal{R})$ Language and System*. ACM Transactions on Programming Languages and Systems, 14 (3) pp. 339–395, 1992.

[JK91] J.P. Jouannaud y C. Kirchner. *Solving Equations in Abstract Algebras: A Rule-Based Survey of Unification*. En J.L. Lassez y G. Plotkin (eds.) Computational Logic, Essays in Honor of Alan Robinson, The MIT Press, pp. 357-321, 1991.

[KR94] H. Kirchner y C. Ringeissen. *Constraint Solving in Combined Algebraic Domains*. En Proceedings of the 11th International Conference on Logic Programming (ICLP'94), MIT Press, pp. 617-631, 1994.

[KKR90] C. Kirchner, H. Kirchner y M. Rusinowitch. *Deduction with Symbolic Constraints*. Revue Française d'Intelligence Artificielle, 4 (3) pp. 9–52, 1990.

[Klo92] J.W. Klop. *Term Rewriting Systems*. En S. Abramsky, D.M. Gabbay y T.S.E. Maibaum (eds.) *Handbook of Logic in Computer Science*, Vol. 2, pp. 2–116, Oxford University Press, 1992.

[KMI01] N. Kobayashi, M. Marin y T. Ida. *Collaborative Constraint Functional Logic Programming in an Open Environment*. En Proc. of The Second Asian Workshop on Programming Languages and Systems (APLAS'01), pp. 49-59, 2001.

[KMI03] N. Kobayashi, M. Marin y T. Ida. *Collaborative Constraint Functional Logic Programming System in an Open Environment*. IEICE Transactions on Information and Systems E86-D, 1, pp. 63-70, 2003.

[KMIC02] N. Kobayashi, M. Marin, T. Ida y Z. Che. *Open CFLP: An open system for collaborative constraint functional logic programming*. En Proc. of the 11th International Workshop on Functional and (Constraint) Logic Programming (WFLP'02). Grado, Italia, pp. 229-232, 2002.

[Kow74] R.A. Kowalski. *Predicate Logic as a Programming Language*. In Information Processing 74, North-Holland, pp. 569–574, 1974.

[KLMR92] H. Kuchen, F.J. López-Fraguas, J.J. Moreno-Navarro y M. Rodríguez-Artalejo. *Implementing a Lazy Functional Logic Language with Disequality Constraints*. En Proc. Joint Int. Conf. and Symposium on Logic Programming (JICSLP'92), The MIT Press, pp. 207–221, 1992.

[Lai00] C. Laï. *Assertions with Constraints for CLP Debugging*. En P. Derasant, M. Hermenegildo y J. Maluszynski (eds.) Analysis and Visualization Tools for Constraint Programming, Capítulo 3, pp. 109–120. Springer LNCS 1870, 2000.

[Lan75] D.S. Lankford. *Canonical inference*. Technical Report ATP-32, Department of Mathematics and Computer Science, University of Texas at Austin, 1975.

[LMM88] J.L. Lassez, M. Maher y K. Marriot. *Unification Revisited*. Foundations of logic and functional programming, Springer LNCS 306, pp. 67–113, 1988.

[Llo87a] J.W. Lloyd. *Foundations of Logic Programming*. 2nd. ed., Springer Verlag, Berlin, Heidelberg, 1987.

[Llo87b] J.W. Lloyd. *Declarative error diagnosis*. New Generation Computing 5(29, pp. 133–154, 1987.

[LLR93] R. Loogen, F.J. López-Fraguas y M. Rodríguez-Artalejo, *A Demand Driven Computation Strategy for Lazy Narrowing*, En Proc. Int. Symp. on Programming Language Implementation and Logic Programming (PLILP'93), Springer LNCS 714, pp. 184–200, 1993.

[Lop92] F.J. López-Fraguas. *A General Scheme for Constraint Functional Logic Programming*. En Proc. Int. Conf. on Algebraic and Logic Programming (ALP'92), Springer LNCS 632, pp. 213–227, 1992.

[Lop94] F.J. López-Fraguas. *Programación Funcional y Lógica con Restricciones*. PhD Tesis, Universidad Complutense de Madrid, 1994.

[LRV04a] F.J. López-Fraguas, M. Rodríguez-Artalejo y R. del Vado-Vírseda. *Constraint Functional Logic Programming Revisited*. En Proc. of the 5th International Workshop on Rewriting Logic and its Applications (WRLA'2004), Elsevier ENTCS 117, pp. 5–50, 2005.

[LRV04b] F.J. López-Fraguas, M. Rodríguez-Artalejo y R. del Vado-Vírseda. *A Lazy Narrowing Calculus for Declarative Constraint Programming*. En Proc. ACM SIGPLAN Conf. on Principles and Practice of Declarative Programming (PPDP'04), ACM Press, pp. 43–54, 2004.

[LRV07] F.J. López-Fraguas, M. Rodríguez-Artalejo y R. del Vado-Vírseda. *A New Generic Scheme for Functional Logic Programming with Constraints*. Revista *Higher-Order and Symbolic Computation 20*, 1/2, pp. 73–122, 2007.

[LS99a] F.J. López-Fraguas y J. Sánchez-Hernández. *Disequalities May Help to Narrow*. En Proc. APPIA-GULP-PRODE'99, pp. 89–104, 1999.

[LS99b] F.J. López-Fraguas y J. Sánchez-Hernández. *TOY: A Multiparadigm Declarative System*. En Proc. RTA'99, Springer LNCS 1631, pp. 244–247, 1999.

[LS00] F.J. López-Fraguas y J. Sánchez-Hernández. *Proving Failure in Functional Logic Programs*. En Proc. Int. Conf. on Computational Logic (CL'2000), Springer LNCS 1861, pp. 179–193, 2000.

[LS01] F.J. López-Fraguas y J. Sánchez-Hernández. *Functional Logic Programming with Failure: A Set-Oriented View*. En Proc. Int. Conf. on Logic Programming and Automated Reasoning (LPAR'2001), Springer LNCS 2250, pp. 455–469, 2001.

[LS02] F.J. López-Fraguas y J. Sánchez-Hernández. *Narrowing Failure in Functional Logic Programming*. En Proc. 6th Int. Symp. on Functional and Logic Programming (FLOPS'2002), Springer LNCS 2441, pp. 212–227, 2002.

[LS03] F.J. López-Fraguas y J. Sánchez-Hernández. *Failure and equality in functional logic programming*. En Proc. 12th International Workshop on Functional and (Constraint) Logic Programming (WFLP'03), Elsevier ENTCS 86(3), pp. 123–143, 2003.

[LS04] F.J. López-Fraguas y J. Sánchez-Hernández. *A Proof Theoretic Approach to Failure in Functional Logic Programming*. Theory and Practice of Logic Programming 4(1), pp. 41–74, 2004.

[LC07] Y. Luo y O. Chitil. *Proving the correctness of algorithmic debugging for functional programs*. En Trends in Functional Programming, volumen 7, pp. 19–34. Intellect Books, 2007.

[Lux01] W. Lux. *Adding linear constraints over real numbers to Curry*. En A. Middeldorp, H. Kuchen, y K. Ueda, editores, *5th International Symposium on Functional and Logic Programming (FLOPS'2001)*, Springer LNCS 2024, pp. 185–200, 2001.

[Mah88] M.J. Maher. *Complete Axiomatization of the Algebras of Finite, Rational and Infinite Trees*. En Proc. of the Third Annual Symposium of Logic in Computer Science (LICS'88), IEEE Computer Society Press, pp. 348–357, 1988.

[Man93] L. Mandel. *The Semantics of the Untyped Constrained Lambda Calculus*. Technical Report 9319, Institut fur Informatic, Ludwig-Maximilians-Univ., Munchen, Octubre 1993.

[Man95] L. Mandel. *Constrained lambda calculus*. Aachen, Verlag Shaker, 1995.

[Mar00] M. Marin. *Functional Logic Programming with Distributed Constraint Solving*. PhD Tesis, Johannes Kepler Universität Linz, 2000.

[MIS99] M. Marin, T. Ida y W. Schreiner. *CFLP: a Mathematica Implementation of a Distributed Constraint Solving System*. En Third International Mathematical Symposium (IMS'99), Hagenberg, Austria, 23–25 de Agosto, 10 páginas, 1999.

[MIS00] M. Marin, T. Ida y T. Suzuki. *Cooperative Constraint Functional Logic Programming*. En International Symposium on Principles of Software Evolution (IPSE'2000), pp. 223–230, 1–2 de Noviembre, 2000.

[MS98] K. Marriott y P.J. Stuckey. *Programming with Constraints, An Introduction*. The MIT Press, Crambidge, Massachusetts, 1998.

[MM02] N. Martí-Oliet y J. Meseguer. *Rewriting logic: roadmap and bibliography*. Theoretical Computer Science 285(2), pp. 121–154, 2002.

[Mes92] J. Meseguer. *Conditional Rewriting Logic as a Unified Model of Concurrency*. Theoretical Computer Science 96, pp. 73–155, 1992.

[MH94] A. Middeldorp y E. Hamoen. *Completeness Results for Basic Narrowing*. Applicable Algebra in Engineering, Communications and Computing 5, pp. 213–253, 1994.

[MO98] A. Middeldorp y S. Okui. *A deterministic lazy narrowing calculus*. Journal of Symbolic Computation, 25 (6), pp. 733–757, 1998.

[MOI96] A. Middeldorp, S. Okui y T. Ida. *Lazy narrowing: strong completeness and eager variable elimination*. Theoretical Computer Science 167, pp. 95–130, 1996.

[Mil78] R. Milner. *A theory of type polymorphism in programming*. Journal of Computer and Systems Sciences 17, pp. 348–375, 1978.

[Mor89] J.J. Moreno-Navarro. *Diseño, semántica e implementación de BABEL: Un lenguaje que integra la programación funcional y lógica*. Tesis doctoral, UPM, Madrid, 1989.

[MR92] J.J. Moreno-Navarro y M. Rodríguez-Artalejo. *Logic programming with functions and predicates: the language BABEL*. Journal of Logic Programming 12, pp. 191–224, 1992.

[Mol85] B. Möller. *On the Algebraic Specification of Infinite Objects - Ordered and Continuous Models of Algebraic Types*. Acta Informática 22, pp. 537–578, 1985.

[MS94] A. Mück y T. Streicher. *A Tiny Constrain Functional Logic Language and Its Continuation Semantics*. En Proc. European Symp. on Programming (ESOP'94), Springer LNCS 788, pp. 439–453, 1994.

[Nai91] L. Naish. *Adding equations to NU-Prolog*. En Proceedings of the Third International Symposium on Programming Language Implementation and Logic Programming (PLILP'91), Springer LNCS 528, pp. 15–26, 1991.

[Nai92] L. Naish. *Declarative Debugging of Lazy Functional Programs*. Technical Report 92/6. Department of Computer Science, University of Melbourne, 1992.

[Nai97] L. Naish. *A Declarative Debugging Scheme*. Journal of Functional and Logic Programming 3, 1997.

[NB95] L. Naish y T. Barbour. *A Declarative Debugger for a logical-functional language*. En Graham Forsyth y Moonis Ali (eds.), Eight International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert System - Invited and Additional Papers, Volume 2:91-99, Melbourne, June 1995. DSTO General Document 51.

[NB96] L. Naish y T. Barbour. *Towards a Portable Lazy Functional Declarative Debugger*. Australian Computer Science Communications, 18(1):401-408, 1996.

[Nil01a] H. Nilsson. *Freja: A small non-strict, purely functional language*. MSc dissertation, Departament of Computer Science and Applied Mathematics, Aston University, Birmingham, U.K., 1991.

[Nil01b] H. Nilsson. *How to look busy while being as lazy as ever: the Implementation of a lazy functional debugger*. Journal of Functional Programming, 11(6): 629–671, 2001.

[NF92] H. Nilsson y P. Fritzson. *Algorithmic Debugging for Lazy Funcional Languages*. En Proceedings of the Fourth International Symposium on Programming Language Implementation and Logic Programming (PLILP'92), Springer LNCS 631 pp. 385-399, 1992.

[NF94] H. Nilsson y P. Fritzson. *Algorithmic Debugging for Lazy Funcional Languages*. Journal of Functional Programming, 4(3): 337–370, 1994.

[NS97] H. Nilsson y J. Sparud. *The Evaluation Dependence Tree as a basis for Lazy Functional Debugging*. Automated Software Engineering, 4(2): 121–150, 1997.

[oDon77] M.J. O'Donnell. *Computing in Systems Described by Equations*. Springer LNCS 58, 1977.

[oDon85] M.J. O'Donnell. *Equational Logic as a Programming Language*. The MIT Press, Cambridge, MA, 1985.

[oDon94] M.J. O'Donnell. *Equational Logic Programming*. En D. Gabbay (ed.), Handbook of Logic in Artificial Intelligence and Logic Programming, volumen 5 on Logic Programming, Capítulo 2, Oxford Science Publications, 1994.

[Pal02] M. Palomino. *Comparing Meseguer's Rewriting Logic with the Logic CRWL*. En Proc. of the International Workshop on Functional and (Constraint) Logic Programming (WFLP'01), Elsevier ENTCS 64, pp. 255–276, 2002.

[Pal07] M. Palomino. *A Comparison between two logical formalism for rewriting*. En M. Falaschi y M. Maher (eds.) *Multiparadigm Languages and Constraint Programming (special issue)*. Journal of Theory and Practice of Logic Programming, volumen 7 (1-2), pp. 183–213, 2007.

[Pey87] S.P. Peyton Jones. *The implementation of functional programming languages*. Prentice Hall, Englewood Cliffs, N.J., 1987.

[PHAB+02] S.L. Peyton Jones (ed.), J. Hughes (ed.), L. Augustsson, D. Barton, B. Boutel, W. Burton, J. Fasel, K. Hammond, R. Hinze, P. Hudak, T. Johnsson, M.P. Jones, J. Launchbury, E. Meijer, J. Peterson, A. Reid, C. Runciman y P. Wadler. *Report on the programming language Haskell 98: a non-strict, purely functional language*. Disponible en `http://www.haskell.org/onlinereport/`, 2002.

[PvE93] R. Plasmeijer y M. van Eekelen. *Functional Programming and Parallel Graph Rewriting*. Addison-Wesley, 1993.

[Pop98] B. Pope. *Buddha. A Declarative Debugger for Haskell*. Honours Thesis, Department of Computer Science, University of Melbourne, Australia, Junio 1998.

[Pop07] B. Pope. *A declarative debugger for Haskell*. PhD Tesis, Computer Science and Software Engineering, University of Melbourne, 2007.

[PN03a] B. Pope y L. Naish. *Practical aspects of declarative debugging in Haskell 98*. En Proc. PPDP'2003, ACM Press, pp. 230–240, 2003.

[PN03b] B. Pope y L. Naish. *A Program Transformation for Debugging Haskell 98*. En Proc. ACSC'2003, Australian Computer Society CRPIT, vol. 16, pp. 227–236, 2003.

[Pre98] C. Prehofer. *Solving Higher-Order Equations. From Logic to Programming*. Birkhäuser, Series *Progress in Theoretical Computer Science*, 1998.

[PBH00a] G. Puebla, F. Bueno y M. Hermenegildo. *An Assertion Language for Constraint Logic Programs*. En P. Derasant, M. Hermenegildo y J. Maluszynski (eds.) *Analysis and Visualization Tools for Constraint Programming*, Capítulo 1, Springer LNCS 1870, pp. 23–61, 2000.

[PBH00b] G. Puebla, F. Bueno y M. Hermenegildo. *A Generic Preprocessor for Program Validation and Debugging*. En P. Derasant, M. Hermenegildo y J. Maluszynski (eds.) *Analysis and Visualization Tools for Constraint Programming*, Capítulo 2, Springer LNCS 1870, pp. 63–107, 2000.

[Rea93] C. Reade. *Elements of Functional Programming*. Addison-Wesley, 1993.

[Red85] U.S. Reddy. *Narrowing as the Operational Semantics of Functional Languages*. En Proc. of Second IEEE Int'l Symp. on Logic Programming, pp. 138–151, 1985.

[Ret87] P. Réty. *Improving basic narrowing techniques*. En Proc. of the Conf. on Rewriting Techniques and Applications (RTA'87), Springer LNCS 256, pp. 228–241, 1987.

[RS82] J.A. Robinson y E.E. Sibert. *LOGLISP: Motivation, Design and Implementation*. En K.L. Clark y S.A. Tärnlund (eds.) *Logic Programming*, Academic Press, pp. 299–313, 1982.

[Rod01] M. Rodríguez-Artalejo. *Functional and Constraint Logic Programming*. En H. Comon, C. Marché y R. Treinen (eds.), *Constraints in Computational Logics, Theory and Applications*, Revised Lectures of the International Summer School CCL'99, Springer LNCS 2002, Capítulo 5, pp. 202–270, 2001.

[Rub94] A. Rubio. *Automated Deduction with Constrained Clauses*. Tesis doctoral, UPC, Barcelona, 1994.

[Rut98] Z. Ruttkay. *Constraint satisfaction: a survey*. CWI Quaterly, 11(2-3): 163–214, 1998.

[Sar92] V. Saraswat. *The category of constraint systems is cartesian closed*. En Proc. of the 7th Annual IEEE Symposium on Logic in Computer Science. IEEE Press, pp. 341–345, 1992.

[Sar93] V. Saraswat. *Concurrent Constraint Programming Languages*. PhD Tesis, Carnegie Mellon University, 1989. En ACM distinguished dissertation series. The MIT press, 1993.

[SR90] V. Saraswat y M. Rinard. *Concurrent Constraint Programming*. En Proc. of the 17th Annual Symposium on Principles of Programming Languages (POPL'90), ACM Computer Society Press, pp. 232–245, 1990.

[SRP91] V. Saraswat, M. Rinard y P. Panangaden. *Semantic Foundations of Concurrent Constraint Programming*. En Proc. of the 18th Annual Symposium on Principles of Programming Languages (POPL'91), ACM Computer Society Press, pp. 333–352, 1991.

[Sco70] D.S. Scott. *Outline of a Mathematical Theory of Computation*. Technical Monograph PRG-2, Oxford University Computing Laboratory, Noviembre 1970.

[Sco82] D.S. Scott. *Domains for Denotational Semantics*. En Proc. ICALP'82, Springer LNCS 140, pp. 577–613, 1982.

[SF89] N. Shahmehri y P. Fritzson. *Algorithmic debugging for imperative programs with side-effects*. Res. Rep. LiTH-IDA-R-89-49. Dept. of Computer and Information Science, Linköping Univ, Suecia, 1989.

[Sha82] E.Y. Shapiro. *Algorithmic Program Debugging*. The MIT Press, Cambridge, Mass., 1982.

[She90] J.B. Shearer. *Some New Optimum Golomb Rulers*. IEEE Transactions on Information Theory, vol. 36, pp. 183–184, 1990.

[ST90] O. Shmueli y S. Tsur. *Logical Diagnosis of LDL programs*. En Proc. of the International Conference on Logic Programming, ICLP'90. The MIT Press, 1990.

[SIC03] SICStus Prolog user's manual, release 3,11,0, Octubre 2003. Swedish Institute of Computer Science, Sweden. Sistema disponible en la dirección `http://www.sics.se/isl/sicstus`.

[Sie89] J. Siekmann. *Unification Theory*. Journal of Symbolic Computation, 7, pp. 207–274, 1989.

[Sil06] J. Silva. *A comparative study of algorithmic debugging strategies*. En proceedings of LOPSTR'06, Springer LNCS 4407, pp. 143–159, 2006.

[Sil07] J. Silva. *Debugging Techniques for Declarative Languages: Profiling, Program Slicing, and Algorithmic Debugging*. PhD Tesis, Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, 2007.

[Sla74] J.R. Slagle. *Automated Theorem-Proving for Theories with Simplifiers, Commutativity and Associativity*. Journal of the ACM 21(4), pp. 622–642, 1974.

[ST94] G. Smolka y R. Treinen. *Records for Logic Programming*. Journal of Logic Programming 18, pp. 229–258, 1994.

[Spa99] J. Sparud. *Tracing and Debugging Lazy Functional Computations*. PhD Thesis. Department of Computer Science, Chalmers University of Technology. Göteborg, Suecia, 1999.

[SN95] J. Sparud y H.Ñilsson. *The Architectura of a Debugger for Lazy Functional Languages*. En Miraille Ducassé Ed., Proceedings of AADEBUG'95, Saint Malo, Francia, 1995.

[SS86] L. Sterling y E.Y. Shapiro. *The Art of Prolog*. The MIT Press, 1986.

[Sto77] J.E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*. The MIT Press, Cambridge, MA, 1977.

[SY84] P.A. Subrahmanyam y J.H. You. *FUNLOG = Functions + Logic: a Computational Model Integrating Functional and Logic Programming*. En Proc. of First IEEE Int'l Symp. on Logic Programming, pp. 144–153, 1984.

[SWI05] SWI-Prolog. `http://www.swi-prolog.org/`, 2005.

[Tar55] A. Tarski. *A lattice-theoretical fixpoint theorem and its applications*. Pacific Journal of Mathematics 5, pp. 285–309, 1955.

[Ter03] Terese. *Term Rewriting Systems*. Cambridge Tracts in Theoretical Computer Science, 55. Cambridge University Press, 2003.

[TF00] A. Tessier y G. Ferrand. *Declarative Diagnosis in the CLP Scheme*. En P. Deransart, M. Hermenegildo y J. Małuszynski (eds.) *Analysis and Visualization Tools for Constraint Programming*, Capítulo 5. Springer LNCS 1870, pp. 151–174, 2000.

[TA95] A. Tolmach y A.W. Appel. *A Debugger for Standard ML*. Journal of Functional Programming 5(2), pp. 155-200, 1995.

[Tsa93] E. Tsang. *Foundations of constraint satisfaction*. Academic Press, London and San Diego, 1993.

[Tur85] A.D. Turner. *Miranda: A non-strict functional language with polymorphic types*. En Proccedings IFIP International Conference on Functional Programming Languages and Computer Architecture (FPCA'85), Springer LNCS 201, 1985.

[Vad02] R. del Vado-Vírseda. *Estrategias de Estrechamiento Perezoso*. Trabajo de Investigación, Departamento de Sistemas Informáticos y Programación, Universidad Complutense de Madrid, 2002.

[Vad03a] R. del Vado-Vírseda. *A Demand Narrowing Calculus with Overlapping Definitional Trees*. En Proc. 12th International Workshop on Functional and (Constraint) Logic Programming (WFLP'03), Germán Vidal (Ed.). Technical Report DSIC-II/13/03, pp. 184–197, 2003.

[Vad03b] R. del Vado-Vírseda. *A Demand-driven Narrowing Calculus with Overlapping Definitional Trees*. En Proc. ACM SIGPLAN Conf. on Principles and Practice of Declarative Programming (PPDP'03), ACM Press, pp. 213–227, 2003.

[Vad05] R. del Vado-Vírseda. *Declarative Constraint Programming with Definitional Trees*. En Proc. 5th International Workshop on Frontiers of Combining Systems (FroCoS'05), Springer LNAI 3717 pp. 184–199, 2005.

[Vad07] R. del Vado-Vírseda. *A Higher-Order Demand-driven Narrowing Calculus with Definitional Trees*. En Proceedings of the 4th International Colloquium on Theoretical Aspects of Computing (ICTAC'07), Macao SAR, China, September 26-28, 2007. Springer LNCS 4711, pp. 169–184, 2007.

[Vad08] R. del Vado-Vírseda. *A Higher-Order Rewriting Logic on Lambda Abstractions for Declarative Programming*. Informe Técnico 09-08. Departamento de Sistemas Informáticos y Programación, Universidad Complutense de Madrid, Octubre, 2008.

[vHen89] P. van Hentenryck. *Constraint Satisfaction in Logic Programming*. Logic Programming Series, The MIT Press, 1989.

[vHen91] P. van Hentenryck. *Constraint logic programming*. The Knowledge Engineering Review, Vol. 6:3, pp. 151–194, 1991.

[vHSD98] P. van Hentenryck, V. Saraswat y Y. Deville. *Design, implementation and evaluation of the constraint language* `cc(FD)`. Journal of Logic Programming 37, pp. 139–164, 1998.

[vHSD94] P. van Hentenryck, H. Simonis y M. Dincbas. *Constraint satisfaction using constraint logic programming*. Artificial Intelligence 58, pp. 113–159, 1994.

[vRBDH+03] P. van Roy, P. Brand, D. Duchier, S. Haridi, M. Henz y C. Schulte. *Logic programming in the context of multiparadigm programming: the Oz experience*. Theory and Practice of Logic Programming 3 (6), pp. 717–763, 2003.

[vRH04] P. van Roy y S. Haridi. *Concepts, techniques and models of computer programming*. The MIT Press, Cambridge, MA, 2004.

[Wad98] P. Wadler. *Why no one uses Functional Languages*. SIGPLAN Notices 33(8), pp. 23–27, 1998.

[Wie03] J. Wielemaker. *An overview of the SWI-Prolog Programming Environment*. En Proccedings of the 13th International Workshop on Logic Programming Environments. Fred Mesnard and Alexander Serebenik (eds.), Katholieke Universiteit Leuven, Heverlee, Belgium, CW 371, pp. 1–16, 2003.

[Win85] G. Winskel. *On Powerdomains and Modality*. Theoretical Computer Science 36, pp. 127–137, 1985.

[Wol96] S. Wolfram. *The Mathematica Book*. Third Edition. Wolfram Media and Cambridge University Press, 1996.