Análisis Semántico y Transformación de Programas Lógico–Funcionales

Germán Vidal Oriola
Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia

Memoria presentada para optar al título de:
Doctor en Informática

Dirigida por:
María Alpuente Frasnedo

Tribunal de lectura:

Presidente:	**Isidro Ramos Salavert**	**U.P. Valencia**
Vocales:	**Moreno Falaschi**	**U. Udine**
	Mario Rodríguez Artalejo	**U.C. Madrid**
	Fernando Orejas Valdés	**U.P. Catalunya**
Secretario :	**Juan José Moreno Navarro**	**U.P. Madrid**

Septiembre de 1996

Resumen

El problema de la integración de la programación lógica y funcional está considerado como uno de los más importantes en el área de investigación sobre programación declarativa. Para que los lenguajes declarativos sean útiles y puedan utilizarse en aplicaciones reales, es necesario que el grado de eficiencia de su ejecución se aproxime al de los lenguajes imperativos, tal y como se ha conseguido con el lenguaje Prolog. Para ello, es imprescindible el desarrollo de herramientas potentes para el *análisis y transformación* de los programas, capaces de optimizar las implementaciones realizadas. En general, es deseable sustituir las aproximaciones *ad-hoc* por tratamientos más sistemáticos para los problemas de análisis y transformación de programas. Puesto que la semántica de los lenguajes lógico–funcionales ha sido objeto de numerosos estudios y está matemáticamente bien formalizada, surge el interés por el desarrollo de métodos y técnicas formales para la formulación de optimizaciones, basadas en la semántica, que preserven las propiedades computacionales del programa. Esta tesis se centra en el desarrollo de tales técnicas, adoptándose una aproximación formal basada en la semántica (operacional) del lenguaje para desarrollar y analizar, en un contexto unificado, las diferentes optimizaciones.

En la primera parte, desarrollamos un marco para el análisis estático de programas lógico–funcionales, basado en la idea de construir aproximaciones correctas de la semántica operacional del programa. Formalizamos un esquema de análisis simple, uniforme y flexible, que permite estudiar distintos tipos de propiedades (relacionadas con el conjunto de respuestas computadas por el programa) de manera correcta y fácilmente implementable. El esquema es independiente de la estrategia de *narrowing* usada en la formulación del mecanismo operacional del lenguaje, lo que contribuye a dar generalidad al mismo.

Las técnicas de evaluación parcial son, de entre la gran variedad de técnicas existentes para la transformación de programas, las que mayor interés han despertado en las dos últimas décadas. Su utilidad no reside únicamente en la posibilidad de especializar programas, sino que sus aplicaciones se extienden también a la generación automática de compiladores o a la optimización de código, por citar sólo las más importantes. En la segunda parte de esta tesis mostramos que, en el contexto

de los lenguajes lógico–funcionales, la especialización de programas se puede basar directamente en el mecanismo operacional de *narrowing* que, debido a la propagación bidireccional de parámetros realizada a través del procedimiento de unificación, es capaz de producir optimizaciones apreciables. Esta visión unificada de ejecución y especialización nos permite explotar las contribuciones de ambos campos, funcional y lógico, y desarrollar un esquema simple y potente para mejorar el programa original respecto a su capacidad para computar respuestas. También mostramos que, debido a la componente funcional, son posibles otras optimizaciones (como la inclusión de pasos de simplificación deterministas) con el beneficio añadido de que, en nuestro esquema, todas las optimizaciones quedan 'compiladas' en el programa transformado. Formalizamos los conceptos básicos para la evaluación parcial de programas lógico–funcionales y demostramos la corrección y completitud de la transformación. El esquema presentado en este trabajo constituye la primera aproximación totalmente automática, correcta y finita para la evaluación parcial de programas lógico–funcionales.

Agradecimientos

La realización de una tesis doctoral no puede considerarse, en modo alguno, fruto del esfuerzo de un solo individuo. Son muchas las personas y entidades a las que el autor desea expresar su sincero agradecimiento y, sin cuya colaboración, la realización de esta tesis no habría sido posible. Sin ánimo de ser exhaustivo, quiero expresar aquí mi gratitud a las siguientes personas:

- A los compañeros del grupo de Programación Lógica e Ingeniería del Software de la Universidad Politécnica de Valencia, que no sólo han colaborado con su trabajo, sino que han contribuído a hacer más grata mi vida en la universidad. De entre ellos, me gustaría destacar a los siguientes: Isidro Ramos, María José Ramírez, Javier Piris, María José Ramis, Salvador Lucas, Asunción Casanova, Vicente Gisbert y, especialmente, a Javier Oliver, mas que un compañero de trabajo, un amigo, y al que le debo en buena parte el haber conseguido finalizar esta tesis en las fechas previstas.

- A Moreno Falaschi, un buen amigo con el que siempre ha sido un placer trabajar, y al que le debo, no sólo la posibilidad de haber realizado varias estancias en distintas universidades italianas, sino el haber conseguido que me sintiese en ellas como en mi propia casa.

- Por último, y muy especialmente, a María Alpuente, sin cuyos conocimientos, apoyo y dedicación constante a lo largo de estos cuatro años, nada de esto habría sido posible. Sin lugar a dudas, suyo es el mérito de haberme lanzado a esta aventura llamada *tesis doctoral*, abriendo las puertas a una nueva etapa de mi vida que, si bien no ha sido especialmente fácil, ha sido una de las más interesantes.

Quiero agradecer también a aquellos organismos que, de una forma u otra, han colaborado en el desarrollo de los trabajos contenidos en esta tesis:

- La Universidad Politécnica de Valencia y, concretamente, el Departamento de Sistemas Informáticos y Computación, en el que el autor desarrolla habitualmente sus actividades tanto docentes como de investigación.

- Las universidades de Padova y Udine, así como el grupo de Programación Lógica de Pisa, liderado por Giorgio Levi, en los que el autor ha realizado distintas estancias de investigación.

- El Ministerio de Educación y Ciencia y la Comisión Interministerial de Ciencia y Tecnología, que han financiado en gran medida los trabajos del autor.

Finalmente, un agradecimiento muy especial para mis padres, hermanos y amigos, que me han soportado durante estos cuatro largos pero intensos años.

Índice General

Capítulo 1

Introducción

El problema de la integración de la programación lógica y funcional está considerado como uno de los más importantes en el área de investigación sobre programación declarativa. Muchas propuestas recientes para la integración usan sistemas de reescritura de términos condicionales como programas y (alguna variante del) estrechamiento (*narrowing*) [Fay79, Sla74] como mecanismo operacional, soportando así unificación y variables lógicas en un contexto funcional. El procedimiento de *narrowing* puede verse como una extensión propia de la reescritura y puede ser implementado eficientemente sustituyendo emparejamiento (ajuste o *pattern matching*) por unificación en el procedimiento de reducción. El uso de *narrowing* como mecanismo operacional para realizar las computaciones supone una extensión muy potente de los programas lógicos tradicionales, y el modelo resultante tiene buenas oportunidades para explotar el paralelismo implícito en el lenguaje.

En general, el procedimiento de *narrowing* tiene un grado de indeterminismo muy alto, debido a la existencia de dos grados de libertad: uno en la elección del subtérmino a reducir y el otro en la elección de la regla. Pese a que el algoritmo de *narrowing* es un método correcto y completo para obtener las soluciones a un sistema de ecuaciones, su terminación no está asegurada. Se han diseñado muchas estrategias para reducir la talla del espacio de búsqueda que eliminan algunas derivaciones inútiles como, por ejemplo, *narrowing* "innermost" [Fri85], *narrowing* básico [Hul80, MH94], *narrowing* selección [BGM88], *narrowing* perezoso [Han94a, MR92, Red85], etc. Cada una de estas estrategias sigue manteniendo, bajo determinadas condiciones, la completitud del cálculo. En [Alp91, AN89, BL86, DP88b, Mor89, Red85] se puede encontrar una revisión, análisis y clasificación de éstas y otras propuestas para la integración. La colección en [DL86] constituye una referencia estándar en el campo. La panorámica más actualizada se presenta en [Han94b].

Para que los lenguajes declarativos sean útiles y puedan utilizarse en aplicaciones reales, es necesario que el grado de eficiencia de su ejecución se aproxime al de los lenguajes imperativos, tal y como se ha conseguido con el lenguaje Prolog. El perfeccionamiento de las técnicas de compilación y la explotación del paralelismo inherente a los programas, se presentan como las dos fuentes principales de mejora de rendimiento. Básicamente, existen dos aproximaciones para la implementación eficiente de un lenguaje lógico–funcional [Han94b]:

1. la compilación a otro lenguaje de alto nivel para el que ya existen técnicas eficientes de compilación [JMM93, vEY87], y

2. la compilación al lenguaje de una "máquina abstracta" (de bajo nivel) que es ejecutable de forma eficiente sobre el *hardware* convencional [KLMR90a, KLMR90b].

En ambas aproximaciones, es necesario el desarrollo de herramientas potentes de análisis y transformación de programas para los lenguajes lógico–funcionales. Por un lado, los métodos de análisis son imprescindibles para el desarrollo de técnicas de compilación avanzadas (que requieran un análisis global del programa). Por otro lado, las técnicas de transformación de programas son útiles, por ejemplo, para la optimización de programas, el desarrollo de técnicas de compilación "abstracta" y la generación automática de compiladores a partir de intérpretes.

La habilidad para analizar un programa, razonando acerca de sus propiedades, es una de las tareas más importantes en el área de síntesis y manipulación de programas. El análisis de flujo de datos, es decir, el proceso de recoger información sobre la forma en que el programa usa las variables y las estructuras de datos (sin necesidad de ejecutarlo), juega un papel fundamental en el diseño de procesadores de programas tales como depuradores, compiladores e intérpretes. En el caso de los lenguajes declarativos, la aproximación al análisis de programas que goza de mayor reconocimiento está basada en la teoría de la interpretación abstracta desarrollada por P. Cousot y R. Cousot [CC77, CC79, CC92]. La idea básica consiste en usar descripciones aproximadas y finitas de los objetos computacionales, haciendo así tratable el problema del análisis de flujo de datos. Todo esfuerzo de investigación en este campo queda justificado, por ejemplo, cuando se recapacita sobre la enorme proporción de código dedicada en la mayoría de compiladores modernos a la comprobación y optimización del código generado, puesto que cualquier mejora en el código intermedio contribuye a producir un código máquina correcto (con idéntica semántica) y más rápido.

La transformación automática de programas es un método para derivar programas correctos y eficientes. En la literatura se han propuesto una gran variedad de transformaciones para mejorar el código. Una de las mejor estudiadas, la evaluación parcial (EP) de programas, ofrece un marco unificado para la investigación acerca de procesadores de lenguajes, en particular, compiladores e intérpretes [Jon88, JGS93].

El objetivo de la EP se puede ver de la siguiente forma: dado un programa y alguna forma de restricción sobre su uso (i.e. información sobre algunos de los datos de entrada), la EP del programa consiste en la construcción de un nuevo programa "residual" más eficiente, pero que se comporta de forma equivalente al original cuando se usa de acuerdo con la restricción impuesta [Jon88]. La compilación y la generación automática de compiladores a partir de intérpretes son dos de sus principales aplicaciones [Fut71]. La EP es, por tanto, una técnica de transformación de programas que pone mayor énfasis en los métodos puramente automáticos que las técnicas de transformación de programas tradicionales. Una de las principales razones para el interés en la EP es el deseo de conseguir una compilación automática y eficiente por medio de programas generales y fácilmente parametrizables [JGS93].

La interacción entre ambas técnicas (análisis y transformación de programas) es bastante usual. Es posible usar técnicas de transformación de programas, como paso previo al análisis, con el objeto de generar un programa transformado semánticamente equivalente al original pero que permite obtener mayor información durante la fase de análisis (compilación abstracta). Asímismo, es posible utilizar técnicas de análisis semántico con el objeto de extraer información útil para optimizar el proceso de transformación del programa [JGS93].

1.1 Análisis Semántico

Una aproximación interesante para el análisis de los lenguajes de alto nivel consiste en considerar el análisis de un programa como un tipo de pseudo-evaluación, es decir, un proceso que imita la ejecución de un programa. La teoría de la Interpretación Abstracta [CC77, CC79] es una teoría de aproximación semántica, que se usa para proporcionar estáticamente (en tiempo finito) respuestas correctas a cierto tipo de cuestiones relevantes sobre el comportamiento de los programas en tiempo de ejecución. Los datos y operadores semánticos "concretos" son aproximados y reemplazados por los correspondientes datos y operadores "abstractos". En este contexto, un análisis se ve como una computación abstracta definida por los operadores abstractos sobre descripciones de datos en lugar de sobre los datos mismos. Diferentes estilos de definición semántica conducen a diferentes aproximaciones al análisis de programas. En el caso de los programas lógicos, existen dos aproximaciones principales: análisis descendente (*top down*) y análisis ascendente (*bottom up*) [CC92, MS89]. La aproximación descendente propaga la información en el mismo sentido que la regla de resolución, mientras que la aproximación ascendente propaga la información como en la computación del menor punto fijo del operador de consecuencias en un paso. Una diferencia importante entre ambas aproximaciones está en la independencia o no del análisis respecto al objetivo, siendo independiente en el caso ascendente y dependiente en el descendente. Los análisis ascendentes permiten estudiar propiedades del conjunto de éxitos del programa y son, generalmente, fáciles de implementar. Por otro lado, los análisis descendentes permiten estudiar propiedades tanto del conjunto de éxitos como de las respuestas parciales, pueden ser más precisos y su implementación suele ser más compleja.

Está demostrado que es posible implementar los lenguajes lógico–funcionales de manera eficiente, adaptando las técnicas de implementación de los lenguajes de programación lógica y funcional pura, si se toma como base una semántica operacional adecuada. Sin embargo, las técnicas avanzadas de compilación, que dependen del análisis global del programa, requieren el desarrollo de nuevos métodos de análisis para programas lógico–funcionales [Han94b]. Sin pretender ser exhaustivos, algunos de los principales trabajos en este campo se han centrado en los siguientes puntos: análisis de la insatisfacibilidad ecuacional [AFM95, BEØ93], análisis de modos [BPM93, Han94c, HZ94], análisis de la *groundness* [Boy93, HL96], análisis de la demanda [MKM$^+$93], análisis para la identificación del paralelismo implícito [SR92, SR93], análisis para la detección de computaciones deterministas [BPM93], etc.

1.2 Transformación de Programas

La transformación automática de programas es un método para derivar programas correctos y eficientes. Por transformación de programas entendemos el problema de, dado un programa P, generar un programa P' que resuelve el mismo problema y es semánticamente equivalente a P, pero goza de un mejor comportamiento respecto a cierto criterio de evaluación [BC93]. La EP es una técnica de transformación de programas que consiste en la especialización de programas respecto a ciertos datos de entrada, conocidos en tiempo de compilación [Fut71]. El programa resultante puede ser ejecutado más eficientemente ya que, usando el conjunto de datos (parcialmente) conocidos, es posible evitar algunas computaciones en tiempo de ejecución que se realizarán, una única vez, durante el proceso de compilación. En general, las técnicas de EP deben incluir algún criterio de parada para garantizar la terminación del proceso [MG94].

La EP ha sido aplicada intensivamente en el campo de la programación funcional (ver [JGS93, SS88] para una lista extensa de referencias). Comenzando por el trabajo de Komorowski [Kom82], la EP también ha atraido un interés considerable en el campo de la programación lógica, donde se la conoce generalmente como deducción parcial. Las transformaciones de plegado y desplegado, definidas por Burstall y Darlington en [BD77] para programas funcionales, son explotadas en mayor o menor medida por las distintas técnicas de EP. El desplegado consiste en el reemplazamiento de una llamada a función por su respectiva definición (aplicando la correspondiente sustitución). El plegado es la transformación inversa, es decir, el reemplazamiento de cierto fragmento de código por la correspondiente llamada a función. Para programas funcionales, los pasos de plegado y desplegado sólo involucran emparejamiento. La aproximación basada en la transformación de plegado/desplegado fue adaptada a la programación lógica por Tamaki y Sato [TS84], reemplazando emparejamiento por unificación en las reglas de transformación. El desplegado de un programa lógico consiste, por tanto, en aplicar un paso de resolución a un subobjetivo en el cuerpo de una cláusula de todas las formas posibles. Gracias al mecanismo de unificación, la EP de programas lógicos es también capaz de propagar información sintáctica sobre los datos de entrada, tal como la estructura de los términos, y no sólo valores constantes, haciendo así la EP de los programas lógicos más potente y efectiva que la EP de los programas funcionales. En [LS91] se presentan los fundamentos de la deducción parcial en un marco formal. Para programas lógicos definidos, Lloyd y Shepherdson han demostrado que la deducción parcial es siempre correcta, pero no necesariamente completa. La completitud se puede establecer usando alguna condición que garantice que todas las llamadas que pueden ocurrir durante la ejecución del programa transformado están cubiertas por alguna cláusula del mismo.

No existen en la literatura muchos trabajos relacionados con la especialización de programas lógico–funcionales. Hemos encontrado, sin embargo, dos excepciones notables. En [LS75], Levi y Sirovich definen un procedimiento de EP para el lenguaje funcional TEL, usando un mecanismo de ejecución simbólica basado en unificación que puede llegar a entenderse como una forma de *narrowing* perezoso. En [DP88a], Darlington y Pull muestran cómo la unificación permite integrar los pasos de desplegado e instanciación (introducidos en el marco de transformación de programas definido en [BD77]), obteniendo así la habilidad de *narrowing* para tratar con variables lógicas. También se esboza un evaluador parcial para el lenguaje funcional HOPE (extendido con unificación). Sin embargo, en ninguno de estos trabajos se han abordado cuestiones de control, terminación o equivalencia semántica. Otro ejemplo del uso de unificación en la técnica de desplegado se puede encontrar en [DR93], donde se utiliza para formular una aproximación a la síntesis de programas funcionales.

1.3 Objetivo de la Tesis

Para que una aproximación al análisis y transformación de programas resulte simple y, a la vez, bien fundamentada, ésta debe basarse en la semántica formal del lenguaje. Dado que la semántica de los lenguajes lógico–funcionales ha sido ampliamente estudiada, surge el interés por el desarrollo de técnicas de análisis y transformación basadas en la semántica. Esta tesis se enmarca en este ámbito y su objetivo se centra en los siguientes puntos:

- Desarrollar un marco genérico para el análisis estático de programas lógico–funcionales. El esquema debe ser independiente de la estrategia de *narrowing*, con el fin de poderse utilizar para distintos lenguajes. Asímismo, el análisis debe ser de propósito general, para poder estudiar distintas características (e.g. insatisfacibilidad, *groundness*, respuestas parciales, enlaces de variables, etc.), y fácilmente implementable sobre los sistemas existentes.

- Establecer fundamentos para las técnicas de evaluación parcial de programas lógico–funcionales. Estas técnicas han demostrado su eficacia en la optimización de los lenguajes lógicos y funcionales puros, y pueden aportar también resultados interesantes en el campo de la integración de ambos paradigmas. Las técnicas de transformación de programas deben estar basadas en la semántica del lenguaje, preservar la semántica del programa original y asegurar la terminación del proceso sin la intervención del usuario.

1.4 Organización de la Memoria

Este documento se organiza como sigue.

En el Capítulo 2, hemos agrupado los principales conceptos técnicos que se utilizan en el resto de la memoria. Empezamos presentando el lenguaje de los sistemas de reescritura de términos condicionales. Se introduce también un cálculo de *narrowing* condicional genérico, que admite como parámetro distintas estrategias de *narrowing*. A continuación, describimos como instancias de dicho cálculo algunas de las relaciones de *narrowing* más representativas: *narrowing* condicional básico, *narrowing* condicional "innermost" y *narrowing* perezoso (además, obviamente, del propio cálculo de *narrowing* condicional original). Adelantamos aquí que muchos de los resultados en este trabajo se presentan para la relación de *narrowing* genérico con estrategia, con la idea de dar mayor generalidad a los resultados.

La composicionalidad es una propiedad deseable que se ha reconocido como fundamental en la semántica de los lenguajes de programación, al permitir explotar distintas extensiones y optimizaciones para el lenguaje [GL91]. Concretamente, la composicionalidad AND establece la posibilidad de computar el significado de un objetivo a partir del de los elementos individuales que lo integran. Esta propiedad es imprescindible, por tanto, para el desarrollo de técnicas de análisis y transformación eficientes basadas en la semántica del lenguaje. En el Capítulo 3, se presentan algunas propiedades de composicionalidad de la semántica basada en *narrowing*. Mostramos un primer resultado que sirve para cualquier estrategia, y se basa en una noción semántica de composición paralela de respuestas. A continuación demostramos que, para aquellas estrategias de *narrowing* que sean "independientes del entorno", se cumple un resultado de composicionalidad para la correspondiente semántica que hace uso de un operador meramente sintáctico de composición paralela. Intuitivamente, una estrategia de *narrowing* se considera independiente del entorno, cuando las sustituciones computadas en la resolución de una ecuación del objetivo, sólo pueden introducir nuevas ocurrencias explotables en la propia ecuación o en sus descendientes (nunca en el resto de ecuaciones). Las estrategias básica, "innermost" y perezosa satisfacen esta condición. También mostramos cómo obtener una versión composicional del cálculo de *narrowing* condicional que puede servir, por ejemplo, como base para una implementación paralela del lenguaje.

En el Capítulo 4, desarrollamos un marco para el análisis de programas que hace uso de una relación de *narrowing* aproximado, formalizada en el marco de la teoría de la interpretación abstracta. La relación de *narrowing* abstracto permite obtener una aproximación, correcta y finita, del conjunto de éxitos (respuestas computadas) de un programa lógico-funcional. El esquema se parametriza mediante la estrategia de *narrowing* usada y la función de aproximación que construye los programas abstractos a partir de los concretos. Demostramos la corrección y terminación del análisis independientemente de la estrategia de *narrowing* empleada, siempre que ésta sea independiente del entorno y el programa abstracto no permita computaciones infinitas. Presentamos un método general para aproximar el programa que garantiza la terminación del análisis, y se presentan algunas instancias de dicho método. Por otro lado, y siguiendo unas líneas similares a las del Capítulo 3, demostramos la composicionalidad del cálculo abstracto (para el mismo tipo de estrategias) y presentamos una caracterización composicional del mismo.

En el Capítulo 5, investigamos varias aplicaciones de la técnica de análisis formalizada en el capítulo anterior. Concretamente, mostramos cómo el conjunto de éxitos abstracto recogido por el análisis permite guiar las computaciones de un *narrower* concreto. De esta forma, se obtiene un espacio de derivaciones de *narrowing* que terminan más a menudo que las del cálculo original, sin pérdida de completitud. A continuación, mostramos cómo incorporar, en el marco de un lenguaje con restricciones ecuacionales, distintas optimizaciones en el mecanismo de comprobación de las restricciones. Por último, formalizamos un análisis simple que nos permite, a partir de la semántica abstracta de un objetivo, extraer información precisa acerca de los enlaces de sus variables en las derivaciones de éxito.

En el Capítulo 6, revisamos algunas de las propuestas para la evaluación parcial de programas lógicos y funcionales, y formulamos los conceptos básicos de una propuesta integrada. Así, definimos una extensión apropiada (usando *narrowing*) de los conceptos de "resultante", "evaluación parcial", "cierre" e "independencia", usados en la formalización de la evaluación (deducción) parcial de los programas lógicos tradicionales. Establecemos una serie de resultados técnicos sobre la transformación definida y demostramos, bajo las condiciones de cierre e independencia introducidas, la corrección y completitud de nuestra propuesta.

En el Capítulo 7, se aborda la construcción de un método automático para la evaluación parcial de programas lógico–funcionales, basado en los conceptos introducidos en el capítulo anterior. El esquema resultante es paramétrico respecto a la estrategia de *narrowing* usada, la regla de desplegado adoptada y un operador de abstracción. Presentamos a continuación una instancia del método, en la que fijamos una regla de desplegado y un operador de abstracción que garantizan la terminación del proceso de manera automática. Ilustramos la efectividad del método definido mediante algunos ejemplos representativos. Finalmente, comentamos las principales líneas de trabajo futuro.

Resumen de Aportaciones

Terminamos esta introducción con un breve resumen de las principales aportaciones de esta tesis. Indicamos también las publicaciones más relevantes a las que ha dado lugar cada parte del trabajo.

- Composicionalidad de la semántica. En primer lugar, se muestra que, bajo la condición de independencia del entorno, es posible establecer la propiedad de composicionalidad AND de una semántica basada en *narrowing* condicional. Este resultado establece, en particular, la composicionalidad de las estrategias de *narrowing* condicional básico, *narrowing* condicional "innermost" y *narrowing* perezoso. En segundo lugar, se demuestra la propiedad de completitud fuerte (*strong completeness*) de dichas estrategias, como una consecuencia de la composicionalidad. Por último, y para aquellas relaciones con la propiedad de composicionalidad AND, se muestra cómo se puede definir un cálculo de *narrowing* composicional, semánticamente equivalente al original, que puede servir como base para una implementación paralela del lenguaje. (El núcleo de estos trabajos ha sido publicado en [AFV94, AFV96a].)

- Análisis estático. Se define un análisis estático de programas lógico-funcionales que permite obtener una aproximación finita del comportamiento operacional del programa, entendido en términos de un *observable* basado en el conjunto de las respuestas computadas. El análisis se define de manera paramétrica respecto a la estrategia de *narrowing* y una versión "abstracta" del programa (que debe garantizar la terminación del análisis). Se presenta un método general para obtener una aproximación correcta del programa, basado en el uso del concepto de "grafo de prevención de bucles", cuya información permite eliminar del programa aquellos términos que pueden provocar derivaciones infinitas. Este método asegura, además, la terminación del análisis. También presentamos algunas posibles instancias del mismo. Por último, se muestra la corrección del análisis resultante para las estrategias de *narrowing* independientes del entorno, e.g. *narrowing* condicional básico, *narrowing* condicional "innermost" y *narrowing* perezoso. (Parte de este trabajo ha sido publicado en [AFV94, AFV96a].)

- Optimizaciones del narrowing. Se formaliza una relación de *narrowing* refinado que permite eliminar, en base a la información del análisis, algunas derivaciones redundantes. De esta forma, se consigue asegurar la terminación en un mayor número de casos. (Parte de este trabajo fue presentado en [AFRV93a, AFRV93b].)

- Resolución perezosa. Extendemos los resultados de composicionalidad a la semántica abstracta, lo que nos permite definir análisis que incorporan a la vez composicionalidad e incrementalidad. Estos resultados se aplican a la construcción de un mecanismo de resolución perezosa incremental para lenguajes lógicos con restricciones ecuacionales. (Parte de este trabajo fue presentado en [AFV93].)

- Transformación de programas. Se definen los fundamentos de las técnicas de evaluación parcial en el marco de los lenguajes lógico–funcionales y se establece, bajo las condiciones de cierre e independencia, la corrección y completitud de la transformación. Asímismo, formulamos un método genérico y automático para la especialización de programas, y presentamos una instancia del mismo que garantiza la terminación del proceso. (Parte de este trabajo ha sido publicado en [AFV95, AFV96b, AFJV96a].)

Capítulo 2

Preliminares

En este capítulo revisamos los conceptos fundamentales de la programación lógico-funcional que se necesitan en la tesis. Tras introducir una serie de nociones preliminares en la Sección 2.1, se introduce, en la Sección 2.2, el algoritmo de *narrowing* condicional, el mecanismo de ejecución estándar básico para programas lógico-funcionales. En la Sección 2.3, se introduce un procedimiento de *narrowing* genérico, que se parametriza usando distintas estrategias de reducción, y se presentan varios refinamientos del *narrowing* formalizados como instancias del mismo. Las cuestiones más específicas, referentes al análisis semántico o a la transformación de programas serán introducidas, posteriormente, al principio del capítulo correspondiente. Para un mayor detalle sobre los conceptos introducidos en este capítulo, se puede consultar [DJ90, Höl89, Klo92].

2.1 Conceptos Básicos

Términos

Denotamos por V un conjunto infinito (numerable) de variables y por Σ un conjunto de símbolos de función f/n, cada uno con una aridad n asociada. $\mathcal{T} \equiv \tau(\Sigma \cup V)$ y $\tau(\Sigma)$ denotan, respectivamente, el conjunto de términos y términos *básicos* (sin variables) construidos sobre $\Sigma \cup V$ y Σ, respectivamente. Asumimos que el alfabeto Σ puede contener algunos símbolos primitivos, incluyendo al menos el constructor $true$ y un símbolo de función binario $=$ para representar la igualdad, escrito en notación infija, que nos permite interpretar las ecuaciones $s = t$ como términos, con $s, t \in \tau(\Sigma \cup V)$. El término $true$ se considera también una ecuación.

Los términos se pueden ver como árboles etiquetados de la forma habitual. Usamos la función estándar $depth$ para denotar la máxima profundidad de un término. Así, $depth(t) = 1$ si t es una constante o una variable, y $depth(t) = 1 +$

$max(\{depth(t_1), \ldots, depth(t_n)\})$ si t es de la forma $f(t_1, \ldots, t_n)$, $n > 0$. Las ocurrencias de un término t se representan por secuencias (posiblemente vacías) de números naturales que sirven para indicar los subtérminos de t. Las ocurrencias están ordenadas por el orden *prefijo*: $u \leq v$ si existe w tal que $uw = v$. Denotamos por Λ la secuencia vacía. $O(t)$ denota el conjunto de ocurrencias de un término t. $\bar{O}(t)$ denota el conjunto de ocurrencias *no variables* de un término t, mientras que $O_V(t)$ denota el conjunto de ocurrencias variables de t (i.e. $O_V(t) = O(t) - \bar{O}(t)$). $t_{|u}$ denota el subtérmino a la ocurrencia u de t, mientras que $t[u]$ denota el símbolo de función que etiqueta el subtérmino $t_{|u}$. $t[s]_u$ denota el término t cuyo subtérmino a la ocurrencia u ha sido reemplazado por s. Estas nociones se pueden extender a secuencias de ecuaciones de forma natural. Por ejemplo, el conjunto de ocurrencias de una secuencia de ecuaciones $g \equiv (e_1, \ldots, e_n)$ se define como: $O(g) = \{i.u \mid u \in O(e_i),\ i = 1, \ldots, n\}$.

La identidad entre objetos sintácticos se denota usando el símbolo $\equiv$. $Var(o)$ denota el conjunto de variables (distintas) que aparecen en el objeto sintáctico o.

Ecuaciones, Sustituciones y Unificadores Sintácticos

Usamos la aproximación de [Mah90] para describir el retículo de los conjuntos (finitos) de ecuaciones. Denotamos por Eqn el dominio formado por los conjuntos finitos de ecuaciones sobre términos, posiblemente cuantificados existencialmente. Los elementos de Eqn se interpretan como conjunciones (cuantificadas) de ecuaciones y se tratan módulo equivalencia lógica. Denotamos por $fail$ cualquier conjunto insatisfacible de ecuaciones, es decir, $fail$ implica a cualquier otro conjunto de ecuaciones. De forma análoga, el conjunto vacío de ecuaciones, denotado por $true$, viene implicado por todos los elementos de Eqn. Denotamos el retículo de ecuaciones por $(Eqn, \leq)$, donde:

$$E \leq E' \Leftrightarrow E' \Rightarrow E$$

es decir, E es una generalización de E' sii E' implica lógicamente a E.

De esta forma, Eqn es un retículo ordenado por $\leq$ cuyo menor elemento es $true$ y cuyo mayor elemento es $fail$. Nótese que, al considerar los elementos de Eqn módulo equivalencia lógica, el preorden $\leq$ se extiende como un orden parcial sobre el retículo.

Es importante destacar que, cuando tratamos con la semántica operacional concreta, no necesitamos usar variables cuantificadas existencialmente, es decir, todas las ecuaciones se pueden considerar como no cuantificadas. En la semántica abstracta, nuestro algoritmo reemplaza algunos de los términos del programa por un símbolo especial que, desde el punto de vista lógico, equivale a una variable cuantificada existencialmente. Por tanto, será sólo cuando discutamos la relación entre la semántica operacional concreta y su contrapartida abstracta cuando será necesario considerar conjuntos de ecuaciones cuantificadas.

Decimos que un conjunto de ecuaciones está en *forma resuelta* si es $fail$ o si tiene la forma $\exists y_1 \ldots \exists y_m.\ \{x_1 = t_1, \ldots, x_n = t_n\}$, donde cada x_i es una variable distinta que no ocurre en ninguno de los términos t_i, y cada y_i ocurre en alguno de los términos t_j. Todo conjunto de ecuaciones tiene al menos una forma resuelta equivalente (lógicamente). Denotamos por $solve(E)$ una función no determinista que obtiene una de tales formas resueltas de E. Por ejemplo, una forma resuelta de $\exists x.\ \{x = f(y), z = y\}$ es $\{z = y\}$, mientras que $\exists y.\ \{x = f(y), z = y\}$ está en forma resuelta.

Restringimos nuestro interés a las sustituciones idempotentes sobre $\tau(\Sigma \cup V)$, y denotamos dicho dominio por Sub. Existe un isomorfismo natural entre sustituciones y ecuaciones no cuantificadas. La representación ecuacional de una sustitución $\theta \equiv \{x_1/t_1, \ldots, x_n/t_n\}$ es el conjunto de ecuaciones $\widehat{\theta} = \{x_1 = t_1, \ldots, x_n = t_n\}$. De forma análoga, la sustitución asociada a un conjunto de ecuaciones en forma resuelta $E \equiv \{x_1 = t_1, \ldots, x_n = t_n\}$ se denota por $sub(E) = \{x_1/t_1, \ldots, x_n/t_n\}$. La función identidad sobre V se conoce como *sustitución vacía* y la denotamos por ϵ.

Dada una sustitución θ y un conjunto de variables $W \subseteq V$, denotamos por $\theta\restriction_W$ la sustitución obtenida de θ restringiendo su dominio $Dom(\theta)$ a las variables de W. Denotamos por $Ran(\theta)$ las variables que aparecen en el rango de θ, i.e. si $\theta = \{x_1/t_1, \ldots, x_n/t_n\}$, entonces $Ran(\theta) = Var(t_1) \cup \ldots \cup Var(t_n)$. Consideramos el preorden $\leq$ habitual entre sustituciones: $\theta \leq \sigma$ sii $\exists \gamma.\ \sigma \equiv \theta\gamma$. Es importante destacar la equivalencia entre este preorden y la implicación lógica: $\theta \leq \sigma$ sii $\widehat{\sigma} \Rightarrow \widehat{\theta}$ (i.e. $\widehat{\theta} \leq \widehat{\sigma}$) [Pal90]. De la misma forma que en el caso anterior, podemos considerar el retículo $(Sub, \leq)$, en el que los elementos de Sub se consideran módulo equivalencia lógica y la relación $\leq$ es, por tanto, un orden parcial sobre las clases de equivalencia de sustituciones.

Una sustitución $\{x_1/t_1, \ldots, x_n/t_n\}$ se denomina un *unificador* del conjunto de ecuaciones E sii $\{x_1 = t_1, \ldots, x_n = t_n\} \Rightarrow E$. Usando el orden entre ecuaciones, θ es un unificador de E sii $E \leq \widehat{\theta}$. Denotamos el conjunto de unificadores de E por $unif(E)$, mientras que $mgu(E)$ denota el *unificador más general* del conjunto de ecuaciones no cuantificado E. Abusando de la notación, usaremos $fail$ para denotar el fallo en la computación del $mgu(E)$ (i.e. el hecho de que E es insatisfacible). Mientras que para cualquier conjunto de ecuaciones no cuantificado existe un unificador más general [LMM88], esto no es cierto en general para el caso de conjuntos de ecuaciones cuantificados [Mah90]. Escribimos $s \stackrel{?}{=} t$ para denotar la propiedad de que los términos s y t unifican sintácticamente (i.e. $mgu(\{s = t\}) \not\equiv fail$).

Decimos que el par $\langle t, \{\theta_1, \ldots, \theta_n\}\rangle$ es una generalización de un conjunto no vacío de términos $\{t_1, \ldots, t_n\}$ si, para todo $i = 1, \ldots, n$, se cumple $t = t_i\theta_i$. Decimos que el par $\langle t, \Theta\rangle$ es la *generalización más específica* (msg, "most specific generalization" [LMM88]) de un conjunto de términos S, y lo denotamos $\langle t, \Theta\rangle = msg(S)$ si 1) $\langle t, \Theta\rangle$

es una generalización de S, y 2) para toda generalización $\langle t', \Theta' \rangle$ de S se cumple que t' es más general que t.

Programas y Objetivos

Definimos ahora el concepto de *programa* y *objetivo* lógico-funcional. Informalmente, presentamos un programa lógico-funcional mediante un conjunto de cláusulas de Horn ecuacionales o, de forma equivalente, mediante un sistema de reescritura de términos condicional.

Una teoría de Horn ecuacional $\mathcal{E}$ consiste de un conjunto finito de cláusulas de la forma $e \Leftarrow e_1, \ldots, e_n$, $n \geq 0$, donde $e, e_i, i = 1, \ldots, n$, son ecuaciones. Un *objetivo ecuacional* es una cláusula de Horn ecuacional sin cabeza o, equivalentemente, una secuencia de ecuaciones. Denotamos por $Goal$ el conjunto de objetivos ecuacionales $\Leftarrow g$ que, en lo que sigue, denotamos simplemente por g.

Es importante clarificar la relación existente entre los dominios Eqn y $Goal$. La diferencia está en que los elementos de Eqn son *conjuntos* de ecuaciones (posiblemente cuantificados existencialmente) mientras que los elementos de $Goal$ son *secuencias* de ecuaciones (sin cuantificar). Esta distinción facilita la manipulación sintáctica de los objetivos, pero no la tendremos en cuenta cuando no haya necesidad. Denotamos por $(Goal, \leq)$ el dominio de los objetivos ecuacionales ordenado por el orden implicativo $\leq$.

Un *sistema de reescritura de términos condicional* (SRTC en lo sucesivo) es un par $(\Sigma, \mathcal{R})$, donde $\mathcal{R}$ es un conjunto finito de (esquemas de) reglas de reescritura (o reglas de reducción) de la forma $(\lambda \to \rho \Leftarrow C)$, donde $\lambda, \rho \in \tau(\Sigma \cup V)$, $\lambda \notin V$ y $Var(\rho) \cup Var(C) \subseteq Var(\lambda)$[1]. La condición C es una secuencia (posiblemente vacía) de ecuaciones $e_1, \ldots, e_n$, $n \geq 0$. Cuando existen variables en C que no aparecen en λ, éstas reciben el nombre de *variables extra*[2]. Cuando una regla de reescritura no posee condición, escribimos simplemente $(\lambda \to \rho)$. Cuando las condiciones de todas las reglas de $\mathcal{R}$ son vacías, decimos que $(\Sigma, \mathcal{R})$ es un sistema de reescritura de términos incondicional (SRT en lo sucesivo). Escribiremos a menudo simplemente $\mathcal{R}$ en lugar de $(\Sigma, \mathcal{R})$ para denotar un SRTC. Denotamos por $(Srtc, \subseteq)$ el dominio de los sistemas de reescritura condicionales, ordenado por la relación de inclusión de conjuntos.

Un término s se *reescribe* (condicionalmente) a un término t, y lo denotamos por $s \to_{\mathcal{R}} t$, si existe una regla $(\lambda \to \rho \Leftarrow s_1 = t_1, \ldots, s_n = t_n) \in \mathcal{R}$, una ocurrencia $u \in O(s)$, una sustitución σ tal que $s_{|u} = \lambda\sigma$, $t = s[\rho\sigma]_u$ y, para todo $i = 1, \ldots, n$, existe un término q_i tal que $s_i\sigma \to^*_{\mathcal{R}} q_i$ y $t_i\sigma \to^*_{\mathcal{R}} q_i$, donde $\to^*_{\mathcal{R}}$ denota el cierre reflexivo y transitivo de la relación $\to_{\mathcal{R}}$. Cuando no haya lugar a confusión, omitiremos el subíndice $\mathcal{R}$ de la relación de reescritura $\to_{\mathcal{R}}$.

[1] Es decir, trabajamos con SRTC's de tipo 1 en la terminología de [MH94].

[2] Si no se indica lo contrario, suponemos que los programas no contienen variables extra.

Un término s está en *forma normal* si no existe ningún término t tal que $s \rightarrow_{\mathcal{R}} t$. Denotamos por $s\downarrow$ la forma normal de s. Una sustitución σ se dice *normalizada* si $x\sigma$ está en forma normal para todo $x \in Dom(\sigma)$. Un SRTC $\mathcal{R}$ se dice *noetheriano* cuando no existe una secuencia infinita de términos de la forma $s_0 \rightarrow_{\mathcal{R}} s_1 \rightarrow_{\mathcal{R}} \ldots$; $\mathcal{R}$ se dice *confluente* si para todo término s tal que $s \rightarrow^*_{\mathcal{R}} t_1$ y $s \rightarrow^*_{\mathcal{R}} t_2$, entonces $t_1 \downarrow t_2$ (i.e. existe un término t tal que $t_1 \rightarrow^*_{\mathcal{R}} t$ y $t_2 \rightarrow^*_{\mathcal{R}} t$). Un SRTC $\mathcal{R}$ noetheriano y confluente se denomina *canónico* o completo.

Una teoría de Horn ecuacional $\mathcal{E}$ se puede ver como un SRTC $\mathcal{R}$ cuando, para cada una de sus cláusulas $s = t \Leftarrow e_1, \ldots, e_n$, se cumplen las siguientes condiciones: $s \notin V$ y $Var(t) \cup Var(e_1) \cup \ldots \cup Var(e_n) \subseteq Var(s)$. En este caso, las reglas son las cabezas de las cláusulas (implícitamente orientadas de izquierda a derecha) y las condiciones son los respectivos cuerpos. Asumimos que estas condiciones se cumplen para todas las teorías ecuacionales que se consideran en este trabajo y, por tanto, hablaremos indistintamente de una teoría de Horn ecuacional $\mathcal{E}$ o de su presentación equivalente como un SRTC $\mathcal{R}$.

Dado un SRTC $\mathcal{R}$, un símbolo de función $f \in \Sigma$ se dice *irreducible* sii no existe ninguna regla $(\lambda \rightarrow \rho \Leftarrow C) \in \mathcal{R}$ tal que f ocurra como símbolo más externo de λ. En caso contrario, f se denomina un símbolo de función *definido*. En los sistemas en los que se considera la anterior distinción, la signatura Σ se encuentra dividida de la forma $\Sigma = \mathcal{C} \biguplus \mathcal{F}$, donde $\mathcal{C}$ es el conjunto de símbolos de función irreducibles (a veces llamados *constructores*) y $\mathcal{F}$ es el conjunto de símbolos de función definidos.

Un tipo importante de teorías de Horn ecuacionales son aquéllas que pueden ser representadas mediante un SRTC *canónico por niveles* [GM86, MH94]. Sea $\rightarrow_{\mathcal{R}}$ la relación de reescritura condicional asociada a la teoría de Horn ecuacional $\mathcal{E}$. Entonces $\rightarrow_{\mathcal{R}}$ es equivalente a $\cup_{i \geq 0}\{\rightarrow_{\mathcal{R}_i}\}$, donde:

1. $\rightarrow_{\mathcal{R}_0} = \emptyset$; y

2. $t \rightarrow_{\mathcal{R}_{n+1}} t'$ sii existe una regla $(\lambda \rightarrow \rho \Leftarrow C) \in \mathcal{R}$, una ocurrencia $u \in O(t)$ y una sustitución σ tal que $t_{|u} = \lambda\sigma$, $t' = t[\rho\sigma]_u$ y $s\sigma\downarrow_{\mathcal{R}_n} s'\sigma$ para todo $(s = s') \in C$.

La relación $\rightarrow_{\mathcal{R}}$ se dice *confluente por niveles* [GM86] si, para todo $n \geq 0$, la relación $\rightarrow_{\mathcal{R}_n}$ es confluente. Existen condiciones sintácticas para asegurar la propiedad de confluencia por niveles [GM86]. Obviamente, la propiedad de confluencia por niveles implica confluencia. Decimos que $\mathcal{R}$ es canónico por niveles si $\rightarrow_{\mathcal{R}}$ es confluente por niveles y noetheriano. Esto es equivalente a decir que $\rightarrow_{\mathcal{R}_n}$ es canónico, para todo $n \geq 0$.

Otra clase importante de sistemas de reescritura son los llamados *decrecientes*. Decimos que un SRTC es decreciente si existe una extensión bien fundada $\succ$ de la relación de reescritura $\rightarrow_{\mathcal{R}}$ con las siguientes propiedades:

1. $\succ$ cumple la *propiedad de subtérmino*, i.e. $t \succ t_{|u}$ para todo $u \in O(t) - \{\Lambda\}$, y

2. si $(\lambda \to \rho \Leftarrow C) \in \mathcal{R}$ y σ es una sustitución, entonces $\lambda\sigma \succ \rho\sigma$ y, para todo $s = t \in C$, $\lambda\sigma \succ s\sigma$ y $\lambda\sigma \succ t\sigma$.

Unificadores Semánticos ($\mathcal{E}$-unificadores)

Cerramos esta primera sección de conceptos básicos introduciendo qué consideramos como las *soluciones* de un objetivo con respecto a un programa.

Cada teoría de Horn ecuacional $\mathcal{E}$ genera una relación de congruencia más pequeña $=_\mathcal{E}$, llamada *$\mathcal{E}$-igualdad*, sobre el conjunto de términos $\tau(\Sigma \cup V)$ (la menor teoría ecuacional que contiene todas las consecuencias lógicas de $\mathcal{E}$ bajo la relación $\models$ y obedece los axiomas de la igualdad para $\mathcal{E}$). $\mathcal{E}$ se puede ver como una presentación o axiomatización de $=_\mathcal{E}$. Abusando de la notación, en ocasiones hablaremos de la teoría ecuacional $\mathcal{E}$ para denotar la teoría $=_\mathcal{E}$ axiomatizada por $\mathcal{E}$. Dado un conjunto de ecuaciones E, decimos que es *$\mathcal{E}$-unificable* sii existe una sustitución σ tal que, para toda ecuación $(s = t) \in E$, se cumple $s\sigma =_\mathcal{E} t\sigma$, i.e. $\mathcal{E} \models (s\sigma = t\sigma)$. La sustitución σ se denomina un $\mathcal{E}$-unificador de E (por abuso de notación, a menudo se denomina simplemente *solución*). El proceso de $\mathcal{E}$-unificación es sólo semidecidible.

Dado un conjunto de variables $W \subseteq V$, la $\mathcal{E}$-igualdad se extiende a sustituciones de la forma estándar: $\sigma =_\mathcal{E} \theta\, [W]$ sii $x\sigma =_\mathcal{E} x\theta$, $\forall x \in W$. W se omitirá cuando sea igual a V. Decimos que σ es una *$\mathcal{E}$-instancia* de σ' (y, por tanto, que σ' es más general que σ) bajo W, y lo denotamos por $\sigma' \leq_\mathcal{E} \sigma\, [W]$, sii $\exists\gamma.\ \sigma =_\mathcal{E} \sigma'\gamma\, [W]$.

Un conjunto de ecuaciones E junto con una teoría ecuacional forman un *problema de $\mathcal{E}$-unificación*. El conjunto $\mathcal{U}_\mathcal{E}(E)$ de todos los $\mathcal{E}$-unificadores de E es recursivamente enumerable [GR89, Höl89, Sie89]. Para problemas de $\mathcal{E}$-unificación, la noción de unificador más general se extiende a conjuntos completos y minimales de $\mathcal{E}$-unificadores. Un conjunto de $\mathcal{E}$-unificadores S de un conjunto de ecuaciones E se dice *completo* sii todo $\mathcal{E}$-unificador σ de E se puede factorizar como $\sigma =_\mathcal{E} \theta\gamma[Var(E)]$, donde $\theta \in S$. Un conjunto completo de $\mathcal{E}$-unificadores de un sistema de ecuaciones puede ser infinito. No siempre existe un conjunto completo y minimal $\mu\mathcal{U}_\mathcal{E}(E)$ de $\mathcal{E}$-unificadores de E. Un procedimiento de $\mathcal{E}$-unificación se considera completo si genera un conjunto completo de $\mathcal{E}$-unificadores para todo sistema de entrada. Más aún, la $\mathcal{E}$-unificación es un problema computacionalmente complejo, y los métodos generales demasiado ineficientes para ser considerados como base del mecanismo computacional de un lenguaje de programación. Por ello, la mayor parte de las aproximaciones adoptan alguna restricción sobre los programas, de modo que se pueda utilizar algún procedimiento eficiente de $\mathcal{E}$-unificación. Este procedimiento se basa, en la mayor parte de los casos, en (alguna variante de) la relación de *narrowing* [Fay79, Lan75, Sla74].

2.2 Narrowing Condicional

En el punto anterior hemos definido las nociones de programa, objetivo y solución. Veamos ahora cuál es el mecanismo de computación asociado al lenguaje que nos permite obtener las soluciones para un programa y un objetivo dados.

Consideramos inicialmente un programa no condicional $\mathcal{R}$. Si queremos evaluar una función cuyos argumentos son básicos, basta emparejar la llamada a función $f(t_1, \ldots, t_n)$ con la parte izquierda λ de alguna regla $(\lambda \rightarrow \rho)$ del programa, realizando una secuencia de pasos de reescritura. Por otro lado, si tenemos una llamada a función cuyos argumentos contienen variables, entonces generalmente es necesario instanciar estas variables a los términos apropiados para poder aplicar un paso de reescritura. Esto se puede llevar a cabo usando unificación en lugar de emparejamiento en el paso de reescritura, y recibe el nombre de *narrowing* [Fay79, Lan75, Sla74]. Informalmente, reducir por *narrowing* una expresión consiste en aplicarle la menor sustitución tal que la expresión resultante se pueda reducir, y entonces reducirla [Hul80]. El procedimiento de *narrowing* no sólo subsume la reescritura, sino también la resolución SLD de los programas lógicos. Formalmente, decimos que un término t se reduce por *narrowing* a un término t' si:

1. u es una ocurrencia no variable de t (i.e. $u \in \bar{O}(t)$);
2. $(\lambda \rightarrow \rho)$ es una variante nueva (i.e. renombrada o estandarizada aparte) de una regla de $\mathcal{R}$;
3. la sustitución σ es el mgu de $t_{|u}$ y λ; y
4. $t' = (t[\rho]_u)\sigma$.

En este caso, escribimos $t \leadsto_{[u,\lambda\rightarrow\rho,\sigma]} t'$, $t \leadsto_{[u,\sigma]} t'$ o simplemente $t \leadsto_\sigma t'$, si queda claro por el contexto. Veamos un ejemplo.

Ejemplo 1 *Dado el programa no condicional* $\mathcal{R}$*:*

$$\begin{array}{rcl} inc(x) & \rightarrow & s(x) \\ dec(s(y)) & \rightarrow & y \end{array}$$

se puede construir la siguiente secuencia de pasos de narrowing para el término $inc(dec(z))$[3]*:*

$$\underline{inc(dec(z))} \leadsto_{\{x/dec(z)\}} s(\underline{dec(z)}) \leadsto_{\{z/s(y)\}} s(y)$$

con sustitución asociada $\{z/s(y)\}$*. Nótese que esta secuencia de reducciones no puede obtenerse usando reescritura, ya que la expresión* $dec(z)$ *no empareja con (no es instancia de) la parte izquierda de ninguna regla.*

[3] En este ejemplo, y en el resto del trabajo, se utilizará la convención de subrayar los subtérminos considerados para realizar el correspondiente paso de *narrowing*.

Para el caso general de programas condicionales y objetivos ecuacionales, podemos formalizar el procedimiento de *narrowing* condicional usando un sistema de transición etiquetado [Plo81] $(Goal, Sub, \leadsto)$ cuya relación de transición $\leadsto \subseteq Goal \times Sub \times Goal$ formaliza los pasos de computación.

Definición 2.2.1 (narrowing condicional $\leadsto$) *Dado un SRTC $\mathcal{R}$, definimos la relación de narrowing condicional* $\leadsto$[4] *como la menor relación que satisface:*

$$\frac{u \in \bar{O}(g) \ \wedge \ r \equiv (\lambda \ \rightarrow \ \rho \ \Leftarrow \ C) \ll \mathcal{R} \ \wedge \ \sigma = mgu(\{g_{|u} = \lambda\})}{g \ \leadsto_\sigma \ (C, g[\rho]_u)\sigma}$$

donde $r \ll \mathcal{R}$ denota que r es una variante nueva de una regla de $\mathcal{R}$, tal que r no contiene variables usadas previamente en la computación (estandarizada aparte).

Un término t se denomina un *redex* (*ex*presión *red*ucible) de *narrowing*, si existe una variante nueva $(\lambda \rightarrow \rho \Leftarrow C)$ de una regla de reescritura en $\mathcal{R}$ y una sustitución σ tales que $t\sigma \equiv \lambda\sigma$. Una *derivación de narrowing* se define como: $g \leadsto_\theta^* g'$ sii $\exists\theta_1, \ldots, \exists\theta_n.\ g \leadsto_{\theta_1} \ldots \leadsto_{\theta_n} g'$ y $\theta = \theta_1 \ldots \theta_n$. Decimos que dicha derivación tiene longitud n. Si $n = 0$, entonces $\theta \equiv \epsilon$. Para tratar la unificación sintáctica como un paso de *narrowing*, añadimos al SRTC la regla $(x = x \ \rightarrow \ true)$, $x \in V$. Así, $(s = t) \leadsto_\sigma true$ sii $\sigma = mgu(\{s = t\})$. Denotamos por $\mathcal{R}_+$ la extensión de un SRTC $\mathcal{R}$ con la regla $(x = x \ \rightarrow \ true)$. Usaremos $\top$ como notación genérica para una secuencia de la forma $true, \ldots, true$. Una *derivación de éxito* para g en $\mathcal{R}$ es una derivación de la forma $g \leadsto_\theta^* \top$ que usa las reglas de $\mathcal{R}_+$. En la siguiente definición formalizamos la semántica operacional (*conjunto de éxitos*) de un objetivo ecuacional g con respecto a un programa $\mathcal{R}$.

Definición 2.2.2 (conjunto de éxitos $\mathcal{O}_\mathcal{R}$) *Dado un programa $\mathcal{R}$ y un objetivo ecuacional g, definimos la semántica operacional (o conjunto de éxitos) de g con respecto a $\mathcal{R}$ como sigue*[5]*:*

$$\mathcal{O}_\mathcal{R}(g) = \{\theta_{\lceil Var(g)} \mid g \leadsto_\theta^* \top\},$$

donde las derivaciones $g \leadsto_\theta^* \top$ se realizan usando las reglas del programa extendido $\mathcal{R}_+$. Las sustituciones $\theta_{\lceil Var(g)}$ se denominan sustituciones de respuesta computada *para g en $\mathcal{R}$.*

Un algoritmo de *narrowing* se dice *completo* si genera un conjunto completo de $\mathcal{E}$-unificadores para cualquier sistema de ecuaciones de entrada. Formalmente, (un

[4] Escribiremos $\leadsto_{[u,r,\sigma]}$ cuando sea necesario distinguir la ocurrencia y la regla usadas.

[5] Omitiremos el subíndice $\mathcal{R}$ cuando el programa se pueda determinar por el contexto.

procedimiento de) *narrowing* es completo para una clase de programas si se cumple la siguiente condición:

$$\text{si } \mathcal{E} \models g\sigma \quad \text{entonces} \quad \begin{array}{l} \text{existe una derivación } g \leadsto_\theta^* \top \\ \text{tal que } \theta \leq_\mathcal{E} \sigma[Var(g)]. \end{array}$$

El subíndice $\mathcal{E}$ puede eliminarse de la expresión $\theta \leq_\mathcal{E} \sigma[Var(g)]$ cuando sólo consideremos completitud con respecto a sustituciones normalizadas [MH94]. Se ha demostrado que el procedimiento de *narrowing* condicional es un algoritmo de $\mathcal{E}$-unificación completo para teorías ecuacionales que satisfacen diferentes restricciones [Han94b, Höl89, MH94]. Por ejemplo, lo es para programas canónicos y también para programas confluentes, si nos restringimos sólo a sustituciones normalizadas.

Ya que el procedimiento de *narrowing* ordinario genera un enorme espacio de búsqueda, se han desarrollado varias estrategias para controlar la selección de los *redexes*, mejorando así la eficiencia de *narrowing* al eliminar derivaciones innecesarias, pero sin perder la completitud del cálculo. En la próxima sección presentamos algunos de los refinamientos que cobran especial interés en este trabajo.

2.3 Estrategias de Narrowing

La implementación eficiente del *narrowing* es una tarea ciertamente difícil pero muy importante, en especial para áreas estrechamente relacionadas como son la programación algebráica, la programación lógica con restricciones (CLP), o la demostración automática de teoremas. Afortunadamente, algunos avances recientes han demostrado que, con restricciones y refinamientos adecuados, *narrowing* puede ser implementado tan eficientemente como –y, para algunos problemas, incluso más eficientemente que– la resolución SLD de los programas lógicos tradicionales [CF92, Han92].

Un componente muy importante para conseguir una implementación eficiente de *narrowing* es el utilizar una estrategia de selección de *redexes* adecuada, ya que el *narrowing* ordinario (Definición 2.2.1) tiene un alto grado de indeterminismo *don't know*[6].

Más concretamente, la ineficiencia del *narrowing* es el resultado de la combinación de los dos grados de libertad del cálculo: 1) la elección del *redex*, y 2) la elección de la regla de reescritura. Una *estrategia de narrowing* consiste, entonces, en sustituir algunas elecciones *don't know* por elecciones *don't care* [CF92] (concretamente, las que se relacionan con 1).

Presentamos a continuación un cálculo de *narrowing* genérico con estrategia, a partir del cual se pueden definir como instancias distintos refinamientos del *narrowing*

[6]Preservamos la terminología en inglés para denotar los dos tipos estándar de indeterminismo: indeterminismo *don't know*, cuando todas las opciones deben ser consideradas, e indeterminismo *don't care*, cuando basta con seleccionar una de las opciones e ignorar el resto.

(y, por supuesto, el propio procedimiento de *narrowing* ordinario). Formulamos el cálculo de acuerdo a la partición de los objetivos ecuacionales en *esqueleto* y *entorno*, como en la formulación del *narrowing* básico de [Höl89]. La parte esqueleto es una secuencia g de ecuaciones y la parte entorno es una sustitución θ. Las sustituciones se van componiendo en la parte entorno, pero no siempre se aplican (completamente) a la parte esqueleto. La idea consiste en mantener en la parte esqueleto g todos los *redexes* del objetivo $g\theta$. Las computaciones se realizan sobre un dominio de estados $State = Goal \times Sub$.

La definición del *narrowing* genérico con estrategia es paramétrica con respecto a la función $\varphi : Goal \rightarrow \wp I\!N^*$ (*estrategia de reducción*), que asigna a cada estado $\langle g, \sigma \rangle$ un subconjunto de $\bar{O}(g)$, y a la función $restrict_\varphi : Sub \times Rule \rightarrow Sub$[7], que restringe las sustituciones que se aplican al objetivo derivado en cada paso. La motivación para la función φ es permitir la explotación de un subconjunto de los *redexes* del objetivo, en lugar de explotarlos todos. Por otro lado, la función $restrict_\varphi$ se encarga de controlar cómo transferir, dependiendo de la estrategia, términos de la parte entorno σ a la parte esqueleto g. De esta forma, se puede reducir el espacio de búsqueda manteniendo, bajo diferentes condiciones, la completitud del cálculo.

El cálculo se define como un sistema de transición $(State, \leadsto_\varphi)$ cuya relación de transición $\leadsto_\varphi \subseteq State \times State$ formaliza los pasos de computación. Dicha relación puede verse como un grafo dirigido, cuyos nodos son estados pertenecientes al dominio $State$. El estado inicial es el nodo *fuente* y los arcos representan reducciones entre estados. Así, una secuencia de reducciones puede verse como un camino en el grafo comenzando en el nodo fuente. Para resolver un objetivo g, el algoritmo comienza con el estado inicial $\langle g, \epsilon \rangle$, e intenta derivar nuevos estados hasta alcanzar un estado terminal de la forma $\langle \top, \theta \rangle$. Cada sustitución de respuesta computada θ en un estado terminal es un $\mathcal{E}$-unificador de g.

Definición 2.3.1 (narrowing genérico con estrategia $\leadsto_\varphi$) *Dado un SRTC $\mathcal{R}$, definimos la relación de narrowing (condicional) genérico con estrategia $\leadsto_\varphi$ como la menor relación que satisface*[8]*:*

$$\frac{\begin{array}{l} u \in \varphi(g) \;\; \wedge \;\; r \equiv (\lambda \rightarrow \rho \Leftarrow C) \ll \mathcal{R} \;\; \wedge \;\; \sigma = mgu(\{(g_{|u})\theta = \lambda\}) \\ \wedge \;\; \sigma_r = restrict_\varphi(\sigma, r) \;\; \wedge \;\; g' = (C, g[\rho]_u)\sigma_r \end{array}}{\langle g, \theta \rangle \;\leadsto_\varphi\; \langle g', \theta\sigma \rangle}$$

donde φ denota una estrategia de narrowing cualquiera, y $restrict_\varphi$ denota la función (dependiente de la estrategia) que restringe la sustitución computada, teniendo en cuenta la regla de programa utilizada, antes de aplicarla al objetivo derivado.

[7] Denotamos con $Rule$ el conjunto de las reglas de programa.

[8] Escribiremos $\leadsto_{\varphi_{[u,r]}}$ cuando sea necesario distinguir la ocurrencia y la regla usadas.

La función $restrict_\varphi$ debe asegurar que todas las ocurrencias seleccionables de un estado $\langle g, \theta\rangle$ (según la estrategia φ) se encuentran en la parte esqueleto g.

Informalmente, el cálculo de *narrowing* condicional, presentado en la Definición 2.2.1, puede verse como una instancia de esta relación de *narrowing* genérico con estrategia, cuando la estrategia de reducción se define como $\varphi(g) = \bar{O}(g)$ y la función de restricción es $restrict_\varphi(\sigma, r) = \sigma$. Nótese que, en este caso, las sustituciones σ de la parte entorno ya se encuentran aplicadas en el esqueleto g del objetivo derivado. Así, la sustitución σ vuelve a aplicarse (por segunda vez) sobre la parte esqueleto en el momento de calcular el nuevo mgu. Esto, sin embargo, no plantea ningún problema ya que sólo trabajamos con sustituciones idempotentes.

Definimos a continuación la semántica operacional (conjunto de éxitos) inducida por la relación de *narrowing* genérico con estrategia.

Definición 2.3.2 (conjunto de éxitos genérico $\mathcal{O}^\varphi_\mathcal{R}$) *Dado un programa $\mathcal{R}$ y un objetivo ecuacional g, formulamos la semántica operacional (o conjunto de éxitos) de g con respecto a $\mathcal{R}$* usando la estrategia φ *como sigue*[9]*:*

$$\mathcal{O}^\varphi_\mathcal{R}(g) = \{\theta_{\restriction Var(g)} \mid \langle g, \epsilon\rangle \leadsto^*_\varphi \langle \top, \theta\rangle\},$$

en la que las derivaciones $\langle g, \epsilon\rangle \leadsto^*_\varphi \langle \top, \theta\rangle$ se realizan usando las reglas del programa extendido $\mathcal{R}_+$.[10]

En el resto del trabajo, denotaremos la relación de *narrowing* ordinario por $\leadsto_{\varphi_\Box}$, $\leadsto_\Box$ o, simplemente, $\leadsto$. De forma similar, la estrategia de *narrowing* ordinario estará denotada por $\varphi_\Box$, mientras que la semántica operacional por *narrowing* ordinario se denotará como $\mathcal{O}^{\varphi_\Box}_\mathcal{R}$, $\mathcal{O}^\Box_\mathcal{R}$ o, simplemente, $\mathcal{O}_\mathcal{R}$.

El número de refinamientos del procedimiento de *narrowing* que se han propuesto en la literatura es enorme. En [Han94b], se citan hasta 18 tipos distintos de estrategias de *narrowing*, la mayor parte de las cuales se pueden agrupar en tres clases, dependiendo de la estrategia con que se seleccionan los *redexes* del objetivo:

- Sin ninguna estrategia, se deben explotar todos los *redexes* del objetivo en cada paso (e.g. *narrowing* ordinario [Sla74], *narrowing* condicional ordinario [Hus85, Kap87]), o con alguna ligera restricción (e.g. *narrowing* básico [Hul80, NRS89, Rét87], *narrowing* condicional básico [Höl89, MH94]).

- Dar prioridad a los *redexes* más internos (*innermost*) del objetivo (e.g. *narrowing* condicional "innermost" [Fri85], *narrowing* condicional "innermost" básico [Höl89]).

[9]De forma similar a la Definición 2.2.2, omitiremos el subíndice $\mathcal{R}$ cuando el programa esté fijado por el contexto.

[10]Usamos una notación común ($\mathcal{R}_+$) para denotar, de forma genérica, la extensión del programa $\mathcal{R}$ con las reglas de la igualdad propias de cada estrategia.

- Dar prioridad a los *redexes* más externos (*outermost*) del objetivo (e.g. *narrowing* "outermost" [Ech88], *narrowing* "outer" [You89], *narrowing* perezoso [Red85], *narrowing* "demand-driven" [MR92], *narrowing* "needed" [AEH94]).

Queremos señalar claramente que no todos los cálculos de *narrowing* con estrategia mencionados se pueden ver como instancia de nuestra definición genérica; pero sí aquéllos que resultan relevantes para los resultados que se desarrollarán en esta tesis, y que podemos entender como representantes sencillos de cada una de las estrategias expuestas. En los tres apartados siguientes presentamos, como instancias de la relación de *narrowing* genérico con estrategia, las estrategias de *narrowing* condicional básico, *narrowing* condicional "innermost" y *narrowing* condicional perezoso.

Narrowing Básico

El *narrowing* (condicional) básico es una forma restringida del *narrowing* (condicional) en el que sólo se consideran para ser reducidos los términos que ocupan posiciones *básicas* [Hul80, MH94]. Informalmente, una ocurrencia básica es una ocurrencia no variable del objetivo inicial o una ocurrencia del objetivo correspondiente a un término no variable que proviene de la parte derecha o del cuerpo de una regla aplicada en un paso previo. La idea subyacente al concepto de *ocurrencia básica* es evitar pasos de *narrowing* sobre subtérminos que hayan sido introducidos por instanciación. *Narrowing* condicional básico es un algoritmo completo de $\mathcal{E}$-unificación para teorías de Horn ecuacionales canónicas por niveles [MH94].

A continuación presentamos la relación de *narrowing* condicional básico como instancia del cálculo de la Definición 2.3.1.

Definición 2.3.3 (narrowing condicional básico $\leadsto_{\blacksquare}$) *La relación de narrowing (condicional) básico $\leadsto_{\varphi_{\blacksquare}}$ (o, simplemente, $\leadsto_{\blacksquare}$) se define como una instancia de la relación de narrowing genérico, donde la estrategia $\varphi_{\blacksquare}$ adoptada es:*

$$\varphi_{\blacksquare}(g) = \bar{O}(g)$$

y la función de restricción se define como:

$$restrict_{\blacksquare}(\sigma, r) = \epsilon.$$

La semántica operacional asociada se denota por $\mathcal{O}_{\mathcal{R}}^{\varphi_{\blacksquare}}$ o, simplemente, por $\mathcal{O}_{\mathcal{R}}^{\blacksquare}$. Las computaciones se realizan sobre el programa extendido $\mathcal{R}_{+} = \mathcal{R} \cup \{x = x \rightarrow true\}$.

De esta forma, la función $\varphi_{\blacksquare}$ selecciona únicamente *redexes* básicos. Debido a que para todo objetivo g y para toda sustitución computada σ, $restrict_{\blacksquare}(\sigma, r) = \epsilon$, todas las ocurrencias básicas de $g\sigma$ se encuentran en g (y coinciden con sus ocurrencias no variables), mientras que las no básicas se encuentran únicamente en el codominio de la sustitución σ.

La siguiente proposición establece la completitud del cálculo.

Proposición 2.3.4 (completitud del narrowing básico [MH94]**)**
Dado un SRTC canónico por niveles $\mathcal{R}$ y un objetivo ecuacional g, $\mathcal{O}^{\blacksquare}_{\mathcal{R}}(g)$ es un conjunto completo de $\mathcal{E}$-unificadores de g.

Narrowing "Innermost"

Presentamos ahora una estrategia de *narrowing* en la que los pasos de computación se realizan sólo sobre las ocurrencias más internas (*innermost*) del objetivo ecuacional. Esta estrategia se corresponde con la estrategia de evaluación de Prolog y con la evaluación impaciente (*eager*) de los lenguajes funcionales. La mayor parte de los conceptos presentados en este punto son una adaptación de [Fri85].

Un término "innermost" t es una operación aplicada a términos constructores, i.e. $t \equiv f(t_1, \ldots, t_n)$, donde $f \in \mathcal{F}$ y, para todo $i = 1, \ldots, n$, $t_i \in \tau(\mathcal{C} \cup V)$. Un SRTC se dice *basado en constructores* (CB, "constructor based") si la parte izquierda λ de cada regla es un término "innermost". Esto implica que no pueden haber anidamientos en las partes izquierdas de las cabezas de las cláusulas ni axiomas entre los constructores. Esta es una clase razonable de programas desde el punto de vista de la programación funcional. Muchas teorías ecuacionales que ocurren en la práctica siguen esta disciplina, e.g. en la especificación de los tipos abstractos de datos.

Un símbolo de función se dice completamente definido si no ocurre en ningún término básico en forma normal, es decir, todos los símbolos de función son reducibles sobre todos los posibles términos básicos (del género apropiado[11]). Un SRTC $\mathcal{R}$ se dice *completamente definido* (CD, "completely defined") si todos los símbolos de función definidos (i.e. pertenecientes a $\mathcal{F}$) están completamente definidos. En un SRTC completamente definido, el conjunto de términos básicos en forma normal coincide con el conjunto de términos irreducibles (o constructores) $\tau(\mathcal{C})$ sobre $\mathcal{C}$. En teorías con un sólo género, es extraño encontrar programas completamente definidos; sin embargo, es usual cuando se usan tipos y cada función se encuentra definida sobre todos los constructores pertenecientes al tipo de sus argumentos. Ejemplos de lenguajes lógico-funcionales que siguen la disciplina CB-CD son, por ejemplo, SLOG [Fri85] y LPG [BE86, BE94].

En una estrategia "innermost", la función φ debe asignar a cada objetivo g una de las ocurrencias más internas ("innermost") de g, según el orden $\leq$ de prefijo. Sin pérdida de generalidad, de las posibles ocurrencias más internas seleccionamos aquélla que aparezca más a la izquierda ("leftmost innermost"). Formalmente, definimos el

[11] Ya que los géneros no son relevantes para los objetivos de este trabajo, los omitimos por simplicidad y sólo consideramos signaturas con un género. La extensión a signaturas heterogéneas es inmediata [Pad88].

conjunto de ocurrencias más internas $Inn(g)$ de un objetivo g como:

$$Inn(g) = \{u \in \bar{O}(g) \mid g_{|u} \text{ es un término “innermost” }\}.$$

Además, definimos un orden $\preccurlyeq$ entre las ocurrencias de $Inn(g)$ de la forma:

$$u \preccurlyeq w \;\Leftrightarrow\; \exists p_0, p_1, p_2 \in I\!N^*,\ \exists i, j > 0.\ u = p_0.i.p_1 \;\wedge\; w = p_0.j.p_2 \;\wedge\; i < j.$$

Intuitivamente, $u \preccurlyeq w$ implica que la ocurrencia u señala un término que aparece más a la izquierda del término señalado por w.

Definición 2.3.5 (narrowing condicional “innermost” $\leadsto_{\blacktriangleleft}$) *La relación de narrowing (condicional) “innermost” $\leadsto_{\varphi_\blacktriangleleft}$ (o, simplemente, $\leadsto_{\blacktriangleleft}$) se define como una instancia de la relación de narrowing genérico con estrategia, donde la estrategia $\varphi_\blacktriangleleft$ es:*

$$\varphi_\blacktriangleleft(g) = \{u\} \;\; si \;\; u \in Inn(g) \;\wedge\; \forall w \in Inn(g).\ (u \neq w \;\Rightarrow\; u \preccurlyeq w)$$

y la función de restricción se define como:

$$restrict_\blacktriangleleft(\sigma, r) = \epsilon.$$

La semántica operacional asociada se denota por $\mathcal{O}_{\mathcal{R}}^{\varphi_\blacktriangleleft}$ o, simplemente, por $\mathcal{O}_{\mathcal{R}}^{\blacktriangleleft}$. Las computaciones se realizan sobre el programa extendido $\mathcal{R}_+ = \mathcal{R} \cup \{x = x \to true\}$.

En el caso del *narrowing* con estrategia “innermost”, las sustituciones computadas no contienen símbolos de función definidos [Fri85] y, por tanto, no es necesario aplicar las sustituciones computadas en cada paso a la parte esqueleto (ya que no pueden introducir nuevos *redexes*). Consecuentemente, la función $restrict_\blacktriangleleft$ se define en la forma obvia para entregar la sustitución vacía.

Decimos que σ es una *sustitución constructora*, si $\sigma \equiv \{x_1/t_1, \ldots, x_n/t_n\}$ y $t_i \in \tau(\mathcal{C})$, $i = 1, \ldots, n$. La siguiente proposición establece la completitud del procedimiento de *narrowing* condicional “innermost” para programas canónicos que sigan la disciplina CB-CD.

Proposición 2.3.6 (completitud del narrowing “innermost” [Fri85]**)** *Sea $\mathcal{R}$ un SRTC canónico CB-CD, g un objetivo ecuacional y σ una solución básica constructora para el objetivo g tal que $Var(g) \subseteq Dom(\sigma)$. Entonces, existe una sustitución de respuesta computada $\theta \in \mathcal{O}_{\mathcal{R}}^{\blacktriangleleft}(g)$ tal que $\theta \leq \sigma[Var(g)]$.*

La condición $Var(g) \subseteq Dom(\sigma)$ en la premisa de la proposición anterior garantiza que $g\sigma$ sea básico. El siguiente ejemplo revela que esta condición no puede ser eliminada, lo cual se ignora, por ejemplo, en [Han94b, Höl89].

Ejemplo 2 *Consideremos el siguiente programa canónico CB-CD $\mathcal{R}$:*

$$\begin{array}{rcl} f(0,y) & \to & y \\ g(0) & \to & 0 \end{array}$$

La sustitución básica constructora $\sigma = \{x/0\}$ es una solución de la ecuación $f(x,g(y)) = g(y)$. Sin embargo, narrowing condicional "innermost" no es capaz de computar una respuesta más general (de hecho, sólo computa la respuesta $\{x/0, y/0\}$). En este caso, la sustitución σ no satisface la condición $Var(g) \subseteq Dom(\sigma)$.

Resulta fácil extender esta estrategia a teorías con funciones que no están completamente definidas, añadiendo simplemente la llamada *regla de reflexión*, que permite ignorar una llamada a función más interna cuando ésta no puede ser reducida [Höl89]. Por simplicidad, asumimos que todas las funciones están completamente definidas y, por tanto, *narrowing* "innermost" es suficiente para computar todas las soluciones.

Este tipo de estrategia, con algunos refinamientos, es la base de lenguajes lógico-funcionales como ALF [Han90], eager-Babel [KLMR90a] o SLOG [Fri85]. Como ya hemos comentado, algunas de las restricciones sobre los programas (e.g. estar completemente definidos), se pueden eliminar realizando ligeras variaciones en la estrategia $\varphi_{\blacktriangleleft}$ del cálculo. Sin embargo, la exigencia de que las reglas del programa sean terminantes, no puede ser eliminada sin pérdida de completitud. Esto puede ser un inconveniente cuando se quieren explotar técnicas típicas de la programación funcional como, por ejemplo, las que trabajan con estructuras de datos infinitas o con funciones parcialmente definidas. En el siguiente punto, presentamos una estrategia de *narrowing* perezoso que preserva la completitud del *narrowing* para programas no terminantes.

Narrowing Perezoso

La estrategia de evaluación perezosa (*lazy evaluation*) en los lenguajes funcionales consiste, básicamente, en no evaluar los argumentos de una función a menos que su evaluación sea necesaria (i.e. que sus argumentos sean "demandados") para computar el resultado. Las estrategias perezosas poseen varias propiedades importantes. En primer lugar, permiten trabajar con estructuras de datos infinitas, ya que los términos no necesitan ser siempre completamente evaluados. Por otra parte, dado un término t, si existe alguna estrategia de evaluación que permite computar la forma normal de t de manera finita, la estrategia perezosa también termina computando dicho valor. Dado que el concepto de posición "demandada" tiene varias interpretaciones posibles, la adaptación de esta estrategia a los lenguajes lógico-funcionales ha dado lugar a distintas estrategias perezosas de *narrowing* (ver, por ejemplo, [AEH94, DG89, Ech88, MKLR90, MR92, Red85, You89]). En nuestro caso, vamos a definir una estrategia de *narrowing* perezoso guiado por la demanda en el estilo de [Red85], y por tanto similar al mecanismo operacional del lenguaje lógico-funcional Babel [MR92].

Debido a la presencia de funciones no terminantes, los resultados de completitud para *narrowing* perezoso se establecen con respecto a una interpretación no estándar de la igualdad: la identidad entre objetos finitos. Los lenguajes lógico-funcionales con una semántica operacional basada en *narrowing* perezoso, definen la validez de una ecuación como una *igualdad estricta* entre los términos. La igualdad estricta $\approx$ considera dos términos iguales sólo si éstos se reducen a un mismo término básico y formado únicamente por símbolos constructores. La semántica de la relación $\approx$ se puede definir mediante el siguiente conjunto confluente StrEq de reglas de programa:

$$\begin{array}{rcll} c \approx c & \rightarrow & true & \forall c/0 \in \mathcal{C} \\ c(x_1,\ldots,x_n) \approx c(y_1,\ldots,y_n) & \rightarrow & true \Leftarrow x_1 \approx y_1,\ldots,x_n \approx y_n & \forall c/n \in \mathcal{C} \end{array}$$

Nótese que la igualdad estricta no posee la propiedad reflexiva $t \approx t$ para todos los términos t. La igualdad estricta es la única definición adecuada de la igualdad para el caso de programas no terminantes. Cuando trabajamos con la relación de *narrowing* perezoso, consideramos que todos los símbolos de la igualdad en los objetivos (y en las condiciones de las reglas) son $\approx$. Asímismo, consideramos ahora que la extensión de un programa $\mathcal{R}$ para tratar la igualdad sintáctica es el programa extendido ($\mathcal{R} \cup$ StrEq) que, por abuso, denotaremos también como $\mathcal{R}_+$.

Existen una serie de condiciones (decidibles) que garantizan la confluencia en el caso de programas CB, incluso en ausencia de la propiedad de terminación: *linealidad por la izquierda* y *no ambigüedad* [GHR92, MR92]. A continuación, formulamos un procedimiento de *narrowing* perezoso que es completo, en algún sentido bien definido, para programas que satisfacen esta doble condición.

Definición 2.3.7 (linealidad por la izquierda) *Un programa $\mathcal{R}$ cumple la propiedad de* linealidad por la izquierda *si, para toda regla $(\lambda \rightarrow \rho \Leftarrow C) \in \mathcal{R}$, λ no contiene ocurrencias repetidas de una misma variable.*

Definición 2.3.8 (no ambigüedad) *Un programa CB $\mathcal{R}$ es* no ambigüo *si, para todo par de reglas de $\mathcal{R}$ definiendo el mismo símbolo de función:*

$$\begin{array}{lcl} f(t_1,\ldots,t_n) & \rightarrow & \rho_1 \Leftarrow C_1 \\ f(s_1,\ldots,s_n) & \rightarrow & \rho_2 \Leftarrow C_2, \end{array}$$

se cumple, al menos, una de las siguientes condiciones:

1. *No superposición: $f(t_1,\ldots,t_n)$ y $f(s_1,\ldots,s_n)$ no unifican.*

2. *Fusión de partes derechas: $\sigma = mgu(\{f(t_1,\ldots,t_n) = f(s_1,\ldots,s_n)\})$ y $\rho_1\sigma \equiv \rho_2\sigma$.*

3. *Incompatibilidad de guardas:* $\sigma = mgu(\{f(t_1,\ldots,t_n) = f(s_1,\ldots,s_n)\})$ *y* $(C_1 \cup C_2)\sigma$ *es suficientemente inconsistente.*

En [GHR92] se define un método constructivo para comprobar la condición (suficiente) de no ambigüedad de la Definición 2.3.8.

Cuando trabajamos con programas CB lineales por la izquierda, la unificación entre las partes izquierdas de las reglas y los términos del objetivo poseen siempre una serie de características comunes que, en general, denominaremos como *problema de unificación lineal.*

Definición 2.3.9 (problema de unificación lineal) *Un problema de unificación lineal es un par de términos* $\langle f(d_1,\ldots,d_n), f(t_1,\ldots,t_n)\rangle$*, donde* $f(d_1,\ldots,d_n)$ *y* $f(t_1,\ldots,t_n)$ *no comparten variables, y* $f(d_1,\ldots,d_n)$ *es un término lineal e "innermost".*

Los problemas de unificación lineal se pueden resolver mediante cualquier versión del algoritmo de unificación (ver, por ejemplo, [LMM88]). Siguiendo [MR92], distinguimos el caso en el que la unificación no tiene éxito debido a un conflicto entre un constructor c y un símbolo de función f, ya que tal situación se puede considerar como una demanda de mayor evaluación de f. El siguiente algoritmo es una reformulación del algoritmo de unificación para el caso de los problemas de unificación lineales de [MR92]. Nótese que, debido a la linealidad por la izquierda, la comprobación conocida como "occur-check" no es necesaria.

Definición 2.3.10 (configuración LU) *Una configuración* LU *es un par* (U,σ)*, donde* U *es un conjunto y* σ *es una sustitución.*

Definición 2.3.11 (relación de unificación $\to_{\mathsf{LU}}$) *Definimos la relación de unificación* $\to_{\mathsf{LU}}$ *como la menor relación que satisface:*

1. $(\{c(d_1,\ldots,d_n) \downarrow_u c(t_1,\ldots,t_n)\} \cup U,\sigma) \to_{\mathsf{LU}} (\{d_1 \downarrow_{u.1} t_1,\ldots,d_n \downarrow_{u.n} t_n\} \cup U,\sigma)$, *donde* $c/n \in \mathcal{C}$, $n \geq 0$.

2. $(\{x \downarrow_u t\} \cup U,\sigma) \to_{\mathsf{LU}} (U\{x/t\},\sigma\{x/t\})$, *donde* $t \notin V$.

3. $(\{d \downarrow_u x\} \cup U,\sigma) \to_{\mathsf{LU}} (U\{x/d\},\sigma\{x/d\})$.

4. $(\{c(d_1,\ldots,d_n) \downarrow_u c'(t_1,\ldots,t_m)\} \cup U,\sigma) \to_{\mathsf{LU}} (\{\mathtt{fail}\},\sigma)$, *donde* $c/n, c'/m \in \mathcal{C}$, $c \not\equiv c'$, *y* $n,m \geq 0$.

Es fácil observar que las computaciones $\to^*_{\mathsf{LU}}$ pueden terminar de tres formas distintas: computando el mgu de los términos de entrada, devolviendo `fail` (en el caso de que los términos no unifiquen) o devolviendo un conjunto de pares $c(d_1,\ldots,d_n) \downarrow_u f(t_1,\ldots,t_n)$, en el que cada ocurrencia u indica una posición demandada. La siguiente definición formaliza dicho comportamiento.

Definición 2.3.12 (comportamiento de $\to_{\mathsf{LU}}$)
Sea $\Gamma \equiv \langle f(d_1, \ldots, d_n), f(t_1, \ldots, t_n)\rangle$ *un problema de unificación lineal. Dada la computación*[12]*:*

$$(\{d_1 \downarrow_1 t_1, \ldots, d_n \downarrow_n t_n\}, \epsilon) \to^*_{\mathsf{LU}} (U, \sigma) \not\to_{\mathsf{LU}}$$

definimos la función $\mathsf{LU}(\Gamma)$ *como sigue:*

$$\mathsf{LU}(\Gamma) = \begin{cases} (\textsc{Success}, \sigma) & \textit{si } U = \emptyset \\ (\textsc{Fail}, \emptyset) & \textit{si } U = \{\mathtt{fail}\} \\ (\textsc{Demand}, P) & \textit{en cualquier otro caso, donde} \\ & P = \{u \mid (c(d'_1, \ldots, d'_m) \downarrow_u g(t'_1, \ldots, t'_k)) \in U\} \\ & \textit{es el conjunto de posiciones demandadas.} \end{cases}$$

Informalmente, una estrategia perezosa debe seleccionar, en cada paso de computación, la ocurrencia de un término externo, excepto cuando hayan subtérminos demandados por el algoritmo de unificación lineal. La siguiente definición formaliza el cálculo de *narrowing* perezoso.

Definición 2.3.13 (narrowing condicional perezoso $\rightsquigarrow_{\blacktriangleright}$) *La relación de narrowing (condicional) perezoso* $\rightsquigarrow_{\varphi_{\blacktriangleright}}$ *(o, simplemente,* $\rightsquigarrow_{\blacktriangleright}$*) se define como una instancia de la relación de narrowing genérico con estrategia, donde la estrategia* $\varphi_{\blacktriangleright}$ *se define de manera inductiva como sigue: dado un objetivo* $g \equiv (e_1, \ldots, e_n)$[13]*,*

$$\begin{array}{lcl} \varphi_{\blacktriangleright}(g) & = & \bigcup_{k=1}^{m} \varphi_{\text{-}}(g, i, k), \quad \textit{con } i \in \{1, \ldots, n\} \\ \varphi_{\text{-}}(g, u, k) & = & \textit{si } \lambda_k[\Lambda] \equiv g[u] \textit{ entonces} \\ & & \begin{cases} \{u\} & \textit{si } \mathsf{LU}(\langle \lambda_k, g_{|u}\rangle) = (\textsc{Success}, \sigma) \\ \bigcup_{u' \in P} \bigcup_{k'=1}^{m} \varphi_{\text{-}}(g, u.u', k') & \textit{si } \mathsf{LU}(\langle \lambda_k, g_{|u}\rangle) = (\textsc{Demand}, P) \\ \emptyset & \textit{si } \mathsf{LU}(\langle \lambda_k, g_{|u}\rangle) = (\textsc{Fail}, \emptyset) \end{cases} \end{array}$$

donde $r_k \equiv (\lambda_k \to \rho_k \Leftarrow C_k) \ll \mathcal{R}_+$*. La función de restricción se define como sigue:*

$$restrict_{\blacktriangleright}(\sigma, r) = \sigma_{\lceil Var(r)}$$

(i.e. σ *restringida a las variables de la regla empleada en el paso de computación). La semántica operacional asociada se denota por* $\mathcal{O}_{\mathcal{R}}^{\varphi_{\blacktriangleright}}$ *o, simplemente, por* $\mathcal{O}_{\mathcal{R}}^{\blacktriangleright}$*. Las computaciones se realizan usando las reglas del programa extendido* $\mathcal{R}_+ = (\mathcal{R} \cup \textsc{StrEq})$.

Tal y como se expone en [MR92], la sustitución computada σ en cada paso de *narrowing* perezoso puede verse como la unión de dos sustituciones $\sigma_g \cup \sigma_r$ representando, respectivamente, las restricciones de σ a las variables del objetivo y a las variables de

[12] Por $(U, \sigma) \not\to_{\mathsf{LU}}$ indicamos que no es posible probar una transición $\to_{\mathsf{LU}}$ a partir del estado (U, σ).

[13] La ocurrencia i que marca la ecuación considerada puede seleccionarse de forma indeterminista *don't care* sin pérdida de completitud [LLR93].

la regla empleada. Es interesante destacar que, debido a la linealidad por la izquierda y a la disciplina de constructores (CB) de los programas, la sustitución σ_g no contiene nunca símbolos de función definidos [MR92] (y, por tanto, los nuevos *redexes* introducidos vendrán necesariamente de la parte σ_r). Por ello, la función $restrict_{\blacktriangleright}$ sólo necesita entregar la sustitución σ_r para ser aplicada y sólo afecta a las variables de la regla r (nunca propaga *redexes* al "contexto"). Es decir, debido a la definición del *narrowing* genérico con estrategia y de la función $restrict_{\blacktriangleright}$, se cumple que, para todo estado $\langle g, \theta \rangle$, la sustitución $\theta_{\upharpoonright Var(g)}$ sólo puede contener términos constructores. Esto se corresponde con la filosofía de la división de los estados en esqueleto y entorno, de forma que todas las ocurrencias explotables se encuentran siempre en la parte esqueleto y, de esta forma, la función $\varphi_{\blacktriangleright}$ no necesita disponer tampoco de la información almacenada en la parte de entorno para calcular las ocurrencias explotables.

La siguiente proposición establece la completitud del cálculo.

Proposición 2.3.14 (completitud del narrowing perezoso [MR92]**)**
Sea $\mathcal{R}$ un programa CB, lineal por la izquierda y no ambigüo, g un objetivo ecuacional y σ una sustitución básica (sobre constructores) tal que, para toda ecuación $s = t$ en g, se cumple $(s\sigma)\downarrow \equiv (t\sigma)\downarrow$, donde $(t\sigma)\downarrow \in \tau(\mathcal{C})$. Entonces, existe una sustitución de respuesta computada θ para $\mathcal{R} \cup \{g\}$ usando $\leadsto_{\blacktriangleright}$ tal que $\theta \leq \sigma[Var(g)]$.

Capítulo 3

Semántica Composicional

La composicionalidad es una propiedad que ha sido reconocida como fundamental en la semántica de los lenguajes de programación [Sco82, Sto77]. Dado un operador de composición $\diamond$, decimos que una semántica es composicional cuando el significado (semántica) de una construcción compuesta $\mathcal{S}(C_1 \diamond C_2)$ se puede definir a partir de los significados de sus componentes $\mathcal{S}(C_1)$ y $\mathcal{S}(C_2)$, es decir, existe una función adecuada $f_\diamond$ tal que $\mathcal{S}(C_1 \diamond C_2) = f_\diamond(\mathcal{S}(C_1), \mathcal{S}(C_2))$. En el caso de los programas lógicos [GL91], se han investigado ampliamente las propiedades de los dos operadores de composición más importantes, concretamente, la composición conjuntiva (AND) –composición de átomos del objetivo o del cuerpo de una cláusula– y la composición disyuntiva (OR) –composición de (conjuntos de) cláusulas–.

Existe una fuerte relación entre las propiedades de composicionalidad y el paralelismo (implícito) de los lenguajes de programación. En particular, la propiedad de composicionalidad AND puede servir de base teórica para desarrollar modelos de ejecución paralela [HR95] (dependientes, en principio), mientras que la propiedad de composicionalidad OR permite, por ejemplo, el desarrollo de métodos de análisis modulares, así como la definición de modelos de ejecución para explotar el paralelismo OR del lenguaje [CDG93]. Como ya hemos comentado, existen instancias de la relación de *narrowing* que pueden ser implementadas más eficientemente que la resolución SLD de los programas lógicos (como es el caso del *narrowing* "innermost" con normalización) [Han91, Han92] gracias a la componente funcional y, además, el modelo de computación resultante ofrece buenas oportunidades para explotar el paralelismo inherente de los programas. Por ejemplo, en [DL90] se describe un tipo de paralelismo OR en el que, para cada ocurrencia del objetivo y cada regla del programa, se explotan de manera concurrente los pasos de *narrowing* alternativos de acuerdo con una cierta función heurística. En [AFRV94], se presentan una serie de propiedades de composicionalidad OR para la relación de *narrowing* condicional básico, que permiten introducir modularidad en la semántica del programa y en el proceso de análisis. En este capítulo, presentamos los principales resultados de nuestras investigaciones sobre la propiedad de composicionalidad AND para programas lógico-funcionales.

En la Sección 3.1, introducimos una versión "semántica" del operador de composición paralela de sustituciones [HR89, Jac89, Pal90], empleado en los modelos de paralelismo conjuntivo de los programas lógicos. Usando este nuevo operador, demostramos un primer resultado de composicionalidad para las soluciones al problema de la unificación semántica de un conjunto de ecuaciones. Sin embargo, el operador semántico de composición paralela es, en general, indecidible. Por ello, en la Sección 3.2.1, mostramos cómo se puede mantener (bajo ciertas restricciones) la composicionalidad de la semántica usando un operador de composición paralela puramente "sintáctico". En segundo lugar, establecemos las condiciones bajo las que la relación de *narrowing* genérico con estrategia posee la propiedad de completitud fuerte. Por último, en la Sección 3.2.2 mostramos cómo definir, dada una estrategia de *narrowing* secuencial, una versión composicional del cálculo en la que se permite la explotación paralela de las distintas ecuaciones del objetivo. Asímismo, demostramos la equivalencia operacional (respecto a las respuestas computadas) entre el cálculo secuencial y su correspondiente versión composicional.

3.1 Composición Paralela Ecuacional

En primer lugar, recordamos la noción de *composición paralela* de sustituciones, denotada por $\Uparrow$ [HR89, Jac89, Pal90]. Informalmente, la composición paralela es la operación de unificación generalizada a sustituciones.

La composición paralela se corresponde con una de las operaciones básicas usadas en el modelo de paralelismo conjuntivo (AND) de los programas lógicos [HR89, Jac89, Pal90]. Concretamente, cuando dos subobjetivos (del mismo objetivo) se explotan en paralelo, las sustituciones computadas (independientemente) deben ser combinadas para obtener el resultado final. Esta "combinación" se puede realizar como sigue. Dadas dos sustituciones idempotentes θ_1 y θ_2, definimos:

$$\theta_1 \Uparrow \theta_2 = mgu(\widehat{\theta}_1 \cup \widehat{\theta}_2).$$

La composición paralela de sustituciones es idempotente, conmutativa, asociativa y tiene como elemento neutro ϵ. El operador $\Uparrow$ se extiende a conjuntos de sustituciones de la forma:

$$\Theta_1 \Uparrow \Theta_2 = \begin{cases} \displaystyle\bigcup_{\theta_1\in\Theta_1,\theta_2\in\Theta_2} \{\theta_1 \Uparrow \theta_2\} & \text{si es diferente de } \{fail\} \\ \emptyset & \text{en otro caso.} \end{cases}$$

La composición paralela fue propuesta en [Pal90] como base para una caracterización composicional de la semántica de los programas lógicos en cláusulas de Horn.

Podemos generalizar la noción de composición paralela para considerar la unificación bajo teorías de Horn ecuacionales. En la siguiente definición formalizamos la noción de *composición paralela ecuacional*, denotada por $\Uparrow_{\mathcal{E}}$.

Definición 3.1.1 *Sean $\theta_1, \theta_2 \in Sub$. Definimos el operador $\Uparrow_{\mathcal{E}} : Sub \times Sub \to \wp Sub$ como sigue:*

$$\theta_1 \Uparrow_{\mathcal{E}} \theta_2 \ = \mathcal{U}_{\mathcal{E}}(\widehat{\theta_1} \cup \widehat{\theta_2}).$$

Es inmediato demostrar que el operador $\Uparrow_{\mathcal{E}}$ es conmutativo, asociativo y tiene como elemento nulo $fail$. Veamos un ejemplo que ilustra el comportamiento del operador semántico de composición paralela $\Uparrow_{\mathcal{E}}$.

Ejemplo 3 *Sea $\mathcal{E} = \{f(0) = 0,\ f(g(X)) = g(X),\ g(0) = c(0),\ g(c(X)) = g(X)\}$.*

1. *Si $\theta_1 = \{X/g(Z)\}$ y $\theta_2 = \{X/c(Z)\}$, entonces $\{X/c(0), Z/0\} \in \theta_1 \Uparrow_{\mathcal{E}} \theta_2$.*

2. *Si $\theta_1 = \{X/f(0)\}$ y $\theta_2 = \{X/g(Z)\}$, entonces $\theta_1 \Uparrow_{\mathcal{E}} \theta_2 = \emptyset$.*

El operador $\Uparrow_{\mathcal{E}}$ se puede extender a conjuntos de sustituciones como sigue.

Definición 3.1.2 *Dados $\Theta_1, \Theta_2 \in \wp Sub$, definimos:*

$$\Theta_1 \Uparrow_{\mathcal{E}} \Theta_2 = \bigcup_{\theta_1 \in \Theta_1, \theta_2 \in \Theta_2} \theta_1 \Uparrow_{\mathcal{E}} \theta_2.$$

Dado un problema de $\mathcal{E}$-unificación $E = E_1 \cup E_2$, una caracterización composicional del conjunto de todos los $\mathcal{E}$-unificadores de E puede darse como sigue.

Proposición 3.1.3 *Sean E, E_1 y E_2 conjuntos de ecuaciones. Sean $E = E_1 \cup E_2$, $\Theta_1 = \mathcal{U}_{\mathcal{E}}(E_1)$ y $\Theta_2 = \mathcal{U}_{\mathcal{E}}(E_2)$. Entonces $\mathcal{U}_{\mathcal{E}}(E) = \Theta_1 \Uparrow_{\mathcal{E}} \Theta_2$.*

Demostración.

($\subseteq$) Dada la sustitución $\vartheta \in \mathcal{U}_{\mathcal{E}}(E_1 \cup E_2)$, entonces $\vartheta \in \mathcal{U}_{\mathcal{E}}(E_1) = \Theta_1$ y $\vartheta \in \mathcal{U}_{\mathcal{E}}(E_2) = \Theta_2$. Por tanto, $\vartheta \in \vartheta \Uparrow_{\mathcal{E}} \vartheta \subseteq \Theta_1 \Uparrow_{\mathcal{E}} \Theta_2$.

($\supseteq$) Dada la sustitución $\theta \in \Theta_1 \Uparrow_{\mathcal{E}} \Theta_2$, por la Definición 3.1.2, $\exists \theta_1 \in \Theta_1, \exists \theta_2 \in \Theta_2$ tales que $\theta \in \mathcal{U}_{\mathcal{E}}(\widehat{\theta_1} \cup \widehat{\theta_2})$. Entonces, $\theta \in \mathcal{U}_{\mathcal{E}}(E_1 \cup E_2)$ ya que:

$$\theta \in \mathcal{U}_{\mathcal{E}}(\widehat{\theta_1}) \ \Rightarrow\ \theta \in \mathcal{U}_{\mathcal{E}}(E_1), \text{ y}$$
$$\theta \in \mathcal{U}_{\mathcal{E}}(\widehat{\theta_2}) \ \Rightarrow\ \theta \in \mathcal{U}_{\mathcal{E}}(E_2).$$

□

Se puede observar que es mucho más complejo evaluar $\Uparrow_{\mathcal{E}}$ que la operación sintáctica $\Uparrow$. Sin embargo, la composición paralela ecuacional se puede redefinir en términos de unificadores "más generales" en el caso de teorías *finitarias*, para las que las soluciones a un problema de $\mathcal{E}$-unificación E se pueden representar siempre por un conjunto completo y minimal $\mu\mathcal{U}_{\mathcal{E}}(E)$ de $\mathcal{E}$-unificadores (maximalmente generales), que es único bajo equivalencia [Sie89]. Las teorías ecuacionales cuyo *tipo de unificación* es finitario, juegan un papel importante en la programación lógica con igualdad [JLM84]. En general, el tipo de unificación de una teoría ecuacional es indecidible. Por otro lado, para una teoría finitaria el requerimiento de minimalidad es a menudo demasiado fuerte, ya que un algoritmo que genere un superconjunto de $\mu\mathcal{U}_{\mathcal{E}}(E)$ puede ser mucho más eficiente que uno minimal (y por tanto, a veces, preferible).

En la siguiente sección mostramos que, bajo determinadas restricciones, es posible seguir trabajando con el operador (sintáctico) de composición paralela $\Uparrow$.

3.2 Composicionalidad de la Semántica

En esta sección, establecemos una serie de propiedades de composicionalidad para la semántica del conjunto de éxitos $\mathcal{O}^{\varphi}_{\mathcal{R}}$ correspondiente a un cálculo de *narrowing* con estrategia φ. A continuación, mostramos cómo utilizar estas propiedades para definir un cálculo de *narrowing* en el que las computaciones para cada ecuación del objetivo se realizan de manera independiente (paralela), "reconciliando" a continuación las sustituciones obtenidas para formar la sustitución global. Este modelo de computación paralela para *narrowing* fue mencionado por primera vez por Uday Reddy en [Red85].

3.2.1 Composicionalidad del Conjunto de Éxitos

Siguiendo las definiciones estándar, lo que "observamos" acerca de un objetivo g en un programa $\mathcal{R}$ es su *conjunto de éxitos* $\mathcal{O}^{\varphi}_{\mathcal{R}}(g)$. Estamos interesados, por tanto, en demostrar bajo qué condiciones dicho conjunto de éxitos se puede construir de manera composicional, usando el operador sintáctico $\Uparrow$. Los siguiente lemas técnicos son necesarios para demostrar la mencionada propiedad de composicionalidad.

Lema 3.2.1 [LMM88, Pal90] *Sean ϑ_1 y ϑ_2 sustituciones idempotentes, entonces:*

$$\vartheta_1 \Uparrow \vartheta_2 = \vartheta_1 mgu(\widehat{\vartheta}_2 \vartheta_1) = \vartheta_2 mgu(\widehat{\vartheta}_1 \vartheta_2).$$

Lema 3.2.2 *Sea g una secuencia de ecuaciones y θ una sustitución idempotente. Entonces, $mgu(g\theta) = mgu(\widehat{mgu}(g)\theta)$.*

DEMOSTRACIÓN.
$mgu(g\theta) = mgu(g \wedge \widehat{\theta})_{\upharpoonright Var(g)} = mgu(\widehat{mgu}(g) \wedge \widehat{mgu}(\widehat{\theta}))_{\upharpoonright Var(g)} = mgu(\widehat{mgu}(g)\theta)$. □

En la siguiente definición se introduce, únicamente por motivos técnicos (puesto que simplificará las pruebas), una nueva relación de *narrowing* denotada por $\leadsto_{\varphi}^{\Uparrow}$. En la nueva relación, el mgu de cada paso se calcula sin tener en cuenta la sustitución actual del estado, y se reemplaza la composición de sustituciones estándar por el operador de composición paralela.

Definición 3.2.3 *Dado un programa $\mathcal{R}$ y una estrategia de narrowing φ, definimos la relación de narrowing $\leadsto_{\varphi}^{\Uparrow}$ como la menor relación que satisface:*

$$\frac{\begin{array}{l} u \in \varphi(g) \;\wedge\; r \equiv (\lambda \to \rho \Leftarrow C) \ll \mathcal{R} \;\wedge\; \delta = mgu(\{g_{|u} = \lambda\}) \\ \wedge\; \delta_r = restrict_{\varphi}(\delta, r) \;\wedge\; \theta \Uparrow \delta \not\equiv fail \end{array}}{\langle g, \theta\rangle \;\leadsto_{\varphi}^{\Uparrow}\; \langle (C, g[\rho]_u)\delta_r, \theta \Uparrow \delta\rangle}$$

Intuitivamente, los estados generados por la relación $\leadsto_{\varphi}^{\Uparrow}$ contendrán un conjunto de ecuaciones (posiblemente) más general que en el caso de la relación $\leadsto_{\varphi}$, debido a la no aplicación de la sustitución actual del estado para computar el nuevo mgu. Esta diferencia, sin embargo, no afecta al conjunto de ocurrencias explotables de un estado; si todas las ocurrencias explotables del estado $\langle g, \theta\rangle$ se encuentran en g, las ocurrencias explotables del estado derivado se pueden obtener tanto a partir de la sustitución $mgu(\{g_{|u}\theta = \lambda\})$ como de la sustitución $mgu(\{g_{|u} = \lambda\})$. Por otra parte, si las ecuaciones pueden ser más generales, la relación $\leadsto_{\varphi}^{\Uparrow}$ podría considerar más ocurrencias para ser explotadas que la relación $\leadsto_{\varphi}$. La condición $\theta \Uparrow \delta \not\equiv fail$ se encarga de evitar este problema, prohibiendo aquellos pasos de derivación en los que la sustitución δ no es 'compatible' con la sustitución actual θ. De hecho, respecto a la parte entorno, ambas relaciones generan las mismas sustituciones para los estados derivados.

La siguiente proposición demuestra la equivalencia entre la relación de *narrowing* genérico con estrategia y la nueva relación $\leadsto_{\varphi}^{\Uparrow}$. Informalmente, este resultado es una consecuencia del Lema 3.2.1 y del hecho de que todas las ocurrencias explotables de un estado se encuentren en la parte esqueleto del objetivo.

Proposición 3.2.4 *Dado un programa $\mathcal{R}$ y un objetivo ecuacional g, los conjuntos de respuestas computadas por $\leadsto_{\varphi}$ y $\leadsto_{\varphi}^{\Uparrow}$, para g en $\mathcal{R}$, coinciden.*

DEMOSTRACIÓN. Vamos a probar un resultado más general: dado un programa $\mathcal{R}$, un objetivo g_0 y una sustitución ϑ_0, entonces existe la derivación:

$$\langle g_0, \vartheta_0\rangle \leadsto_{\varphi} \langle g_1, \vartheta_1\rangle \leadsto_{\varphi} \ldots \leadsto_{\varphi} \langle g_n, \vartheta_n\rangle$$

si y sólo si existe la derivación:

$$\langle g_0, \vartheta_0\rangle \equiv \langle h_0, \theta_0\rangle \rightsquigarrow^{\Uparrow}_{\varphi} \langle h_1, \theta_1\rangle \rightsquigarrow^{\Uparrow}_{\varphi} \ldots \rightsquigarrow^{\Uparrow}_{\varphi} \langle h_n, \theta_n\rangle$$

donde $\vartheta_i = \theta_i$, $g_i\vartheta_i = h_i\theta_i$ y $\varphi(g_i) \subseteq \varphi(h_i)$, para todo $i = 1, \ldots, n$.

($\Rightarrow$) Consideremos la derivación:

$$\langle g_0, \vartheta_0\rangle \rightsquigarrow_{\varphi} \ldots \rightsquigarrow_{\varphi} \langle g_{n-1}, \vartheta_{n-1}\rangle \rightsquigarrow_{\varphi} \langle g_n, \vartheta_n\rangle.$$

Realizamos la prueba por inducción sobre el número de pasos n en la derivación. Para $n = 0$ el resultado es trivial. Consideremos, pues, el caso inductivo $n > 1$. Por la hipótesis de inducción, existe la derivación:

$$\langle h_0, \theta_0\rangle \rightsquigarrow^{\Uparrow}_{\varphi} \ldots \rightsquigarrow^{\Uparrow}_{\varphi} \langle h_{n-1}, \theta_{n-1}\rangle$$

donde $\vartheta_i = \theta_i$, $g_i\vartheta_i = h_i\theta_i$ y $\varphi(g_i) \subseteq \varphi(h_i)$, para todo $i = 1, \ldots, n-1$.
Consideremos ahora el último paso de la derivación:

$$\langle g_{n-1}, \vartheta_{n-1}\rangle \rightsquigarrow_{\varphi} \langle g_n, \vartheta_n\rangle$$

y supongamos $u \in \varphi(g_{n-1})$, $r \equiv (\lambda \to \rho \Leftarrow C) \ll \mathcal{R}$, $\sigma = mgu(\{g_{n-1|u}\vartheta_{n-1} = \lambda\})$ y $\sigma_r = restrict_{\varphi}(\sigma, r)$. Entonces, $g_n = (C, g_{n-1}[\rho]_u)\sigma_r$ y $\vartheta_n = \vartheta_{n-1}\sigma$.
Dado que $\varphi(g_i) \subseteq \varphi(h_i)$, $i = 1, \ldots, n-1$, entonces $u \in \varphi(h_{n-1})$. Dado que $Var(\lambda) \cap Var(\vartheta_{n-1}) = \emptyset$ y, por definición, todas las ocurrencias reducibles de $g_{n-1}\vartheta_{n-1}$ se encuentran en g_{n-1}, entonces:

$$\begin{array}{rcl}
\vartheta_n & = & \vartheta_{n-1}\sigma \\
& = & \vartheta_{n-1} mgu(\{g_{n-1|u}\vartheta_{n-1} = \lambda\}) \\
& = & \vartheta_{n-1} mgu(\{g_{n-1|u} = \lambda\}\vartheta_{n-1}) \\
& = & \vartheta_{n-1} mgu(\{h_{n-1|u} = \lambda\}\theta_{n-1}) \ \ \text{(ya que } g_{n-1}\vartheta_{n-1} = h_{n-1}\theta_{n-1}) \\
& = & \theta_{n-1} mgu(\{h_{n-1|u} = \lambda\}\theta_{n-1}) \ \ \text{(ya que } \vartheta_{n-1} = \theta_{n-1}) \\
& = & \theta_{n-1} mgu(\widehat{mgu}(\{h_{n-1|u} = \lambda\})\theta_{n-1}) \ \ \text{(por el Lema 3.2.2)} \\
& = & \theta_{n-1} \Uparrow mgu(\{h_{n-1|u} = \lambda\}) \ \ \text{(por el Lema 3.2.1)} \\
& = & \theta_{n-1} \Uparrow \delta \\
& = & \theta_n
\end{array}$$

donde $\delta = mgu(\{h_{n-1|u} = \lambda\})$. Sabemos, por tanto, que existe la transición:

$$\langle h_{n-1}, \theta_{n-1}\rangle \rightsquigarrow^{\Uparrow}_{\varphi} \langle h_n, \theta_n\rangle$$

tal que $\theta_n = \theta_{n-1} \Uparrow \delta = \vartheta_{n-1}\sigma = \vartheta_n$. Demostramos ahora que se siguen cumpliendo las condiciones impuestas sobre la parte esqueleto.
Puesto que $g_{n-1}\vartheta_{n-1} = h_{n-1}\theta_{n-1}$, entonces $g_{n-1}\vartheta_n = g_{n-1}\vartheta_{n-1}\sigma = h_{n-1}\theta_{n-1}\sigma = h_{n-1}\vartheta_{n-1}\sigma$ (ya que $\vartheta_{n-1} = \theta_{n-1}$) $= h_{n-1}\vartheta_n = h_{n-1}\theta_n$ (ya que $\vartheta_n = \theta_n$). Sea $\delta_r =$

$restrict_\varphi(\delta, r)$. Ahora, ya que $g_{n-1}\vartheta_n = h_{n-1}\theta_n$, $\vartheta_n = \theta_n$ y las sustituciones σ y δ son idempotentes, se cumple que $g_n\vartheta_n = (C, g_{n-1}[\rho]_u)\sigma_r\vartheta_n = (C, g_{n-1}[\rho]_u)\vartheta_n = (C\vartheta_n, g_{n-1}\vartheta_n[\rho\vartheta_n]_u) = (C\theta_n, h_{n-1}\theta_n[\rho\theta_n]_u) = (C, h_{n-1}[\rho]_u)\theta_n = (C, h_{n-1}[\rho]_u)\delta_r\theta_n = h_n\theta_n$. Por último, dado que $\varphi(g_{n-1}) \subseteq \varphi(h_{n-1})$, las nuevas ocurrencias explotables son recogidas tanto por la sustitución σ como por la sustitución δ, con lo que trivialmente se cumple $\varphi(g_n) \subseteq \varphi(h_n)$.

($\Leftarrow$) Consideremos la derivación:

$$\langle h_0, \theta_0\rangle \leadsto^{\Uparrow}_{\varphi} \ldots \leadsto^{\Uparrow}_{\varphi} \langle h_{n-1}, \theta_{n-1}\rangle \leadsto^{\Uparrow}_{\varphi} \langle h_n, \theta_n\rangle.$$

Realizamos la prueba por inducción sobre el número de pasos n en la derivación. Para $n = 0$ el resultado es trivial. Consideremos, pues, el caso inductivo $n > 1$. Por la hipótesis de inducción, existe la derivación:

$$\langle g_0, \vartheta_0\rangle \leadsto_{\varphi} \ldots \leadsto_{\varphi} \langle g_{n-1}, \vartheta_{n-1}\rangle$$

donde $\vartheta_i = \theta_i$, $g_i\vartheta_i = h_i\theta_i$ y $\varphi(g_i) \subseteq \varphi(h_i)$, para todo $i = 1, \ldots, n-1$.
Consideremos ahora el último paso de la derivación:

$$\langle h_{n-1}, \theta_{n-1}\rangle \leadsto^{\Uparrow}_{\varphi} \langle h_n, \theta_n\rangle$$

y supongamos $u \in \varphi(h_{n-1})$, $r \equiv (\lambda \to \rho \Leftarrow C) \ll \mathcal{R}$, $\delta = mgu(\{h_{n-1|u} = \lambda\})$ y $\delta_r = restrict_\varphi(\delta, r)$. Entonces, $h_n = (C, h_{n-1}[\rho]_u)\delta_r$ y $\theta_n = \theta_{n-1} \Uparrow \delta \not\equiv fail$.
Al igual que en el caso anterior, se verifica la equivalencia:

$$\begin{array}{rcl}
\vartheta_n & = & \vartheta_{n-1}\sigma \\
& = & \vartheta_{n-1}mgu(\{g_{n-1|u}\vartheta_{n-1} = \lambda\}) \\
& = & \vartheta_{n-1}mgu(\{g_{n-1|u} = \lambda\}\vartheta_{n-1}) \\
& = & \vartheta_{n-1}mgu(\{h_{n-1|u} = \lambda\}\theta_{n-1}) \ \ (\text{ya que } g_{n-1}\vartheta_{n-1} = h_{n-1}\theta_{n-1}) \\
& = & \theta_{n-1}mgu(\{h_{n-1|u} = \lambda\}\theta_{n-1}) \ \ (\text{ya que } \vartheta_{n-1} = \theta_{n-1}) \\
& = & \theta_{n-1}mgu(\widehat{mgu}(\{h_{n-1|u} = \lambda\})\theta_{n-1}) \ \ (\text{por el Lema 3.2.2}) \\
& = & \theta_{n-1} \Uparrow mgu(\{h_{n-1|u} = \lambda\}) \ \ (\text{por el Lema 3.2.1}) \\
& = & \theta_{n-1} \Uparrow \delta \\
& = & \theta_n
\end{array}$$

y entonces se cumple $\sigma = mgu(\{g_{n-1|u}\vartheta_{n-1} = \lambda\}) \not\equiv fail$ y $u \in \varphi(g_{n-1})$.
La prueba de las condiciones $g_n\vartheta_n = h_n\theta_n$ y $\varphi(g_n) \subseteq \varphi(h_n)$ es perfectamente análoga a la del caso anterior.

Para terminar, la proposición se sigue del resultado demostrado, para los valores $\vartheta_0 \equiv \theta_0 \equiv \epsilon$ y $g_n \equiv h_n \equiv \top$. □

Nótese que, para las estrategias básica e "innermost", los estados en ambas derivaciones son idénticos, ya que las sustituciones no son en ningún caso aplicadas a la parte esqueleto. En la estrategia perezosa, sin embargo, los estados contienen (posiblemente) un conjunto de ecuaciones distinto, pero esta diferencia sólo puede afectar a los términos constructores (ambos conjuntos de ecuaciones contendrán las mismas funciones definidas). En el resto del trabajo no distinguiremos entre las relaciones $\leadsto_\varphi$ y $\leadsto_\varphi^\Uparrow$, ya que ambas tienen la habilidad para computar las mismas soluciones, denotando ambas relaciones como $\leadsto_\varphi$ cuando no haya lugar a confusión. El uso de una u otra relación vendrá indicado únicamente por el operador de composición empleado: la composición estándar en el caso de la relación $\leadsto_\varphi$ y la composición paralela en el caso de la relación $\leadsto_\varphi^\Uparrow$.

Independencia del Entorno

En este apartado, introducimos las condiciones que garantizan que la relación de *narrowing* genérico con estrategia $\leadsto_\varphi$ es composicional con respecto al operador conjuntivo, i.e. con respecto a la unión o concatenación de secuencias de ecuaciones, usando el operador sintáctico $\Uparrow$ de composición paralela de respuestas. En primer lugar, introducimos una condición sobre las estrategias de *narrowing* que limita las posibles instancias de la relación de *narrowing* genérico.

Definición 3.2.5 (estrategia independiente del entorno) *Una estrategia de narrowing φ es* independiente del entorno *si para toda derivación de la forma:*

$$\langle (g_1, g_2), \theta\rangle \leadsto_\varphi^* \langle (g_1', g_2)\sigma', \theta\sigma\rangle$$

en la que no se ha explotado ninguna ocurrencia correspondiente a g_2, se cumple:

$$\varphi(g_2\sigma') \subseteq \varphi(g_2).$$

Por abuso de notación, en ocasiones diremos que la propia relación $\leadsto_\varphi$ es independiente del entorno.

Informalmente, una estrategia de *narrowing* se dice independiente del entorno si, dado un objetivo compuesto (g_1, g_2), las sustituciones computadas por pasos de *narrowing* sobre ecuaciones pertenecientes a g_1 pueden restringir las ocurrencias explotables de g_2, pero nunca pueden dar lugar a nuevos *redexes* en g_2. Esta noción es similar, aunque más débil, a la de estrategia "básica". Trivialmente, si una estrategia es básica, entonces es independiente del entorno. Sin embargo, lo contrario no es cierto. Por ejemplo, la estrategia perezosa es independiente del entorno (como se demostrará en el Lema 3.2.9) pero, contrariamente a lo que se afirma en [You89], no es una estrategia básica. El siguiente ejemplo ilustra este punto.

Ejemplo 4 *Sea $\mathcal{R}$ el programa:*

$$\begin{array}{lcl} f(x) & \to & c(x) \\ g(0) & \to & 0 \end{array}$$

y $g \equiv (f(g(y)) = c(0))$ *un objetivo. La siguiente derivación perezosa:*

$$\begin{array}{lll}\langle \underline{f(g(y))} = c(0), \epsilon\rangle & \leadsto_{\blacktriangleright} & \langle \underline{c(g(y)) = c(0)}, \{x/g(y)\}\rangle \\ & \leadsto_{\blacktriangleright} & \langle (\underline{g(y)} = 0, true), \{x/g(y), y_1/g(y), y_2/0\}\rangle \\ & \leadsto_{\blacktriangleright} & \langle (\underline{0 = 0}, true), \{x/g(0), y/0, y_1/g(0), y_2/0\}\rangle \\ & \leadsto_{\blacktriangleright} & \langle \top, \{x/g(0), y/0, y_1/g(0), y_2/0\}\rangle\end{array}$$

no es una derivación básica. Concretamente, el paso:

$$\langle (\underline{g(y)} = 0, true), \{x/g(y), y_1/g(y), y_2/0\}\rangle \leadsto_{\blacktriangleright} \langle (0 = 0, true), \{x/g(0), y/0, y_1/g(0), y_2/0\}\rangle$$

no es un paso básico, ya que el redex $g(y)$ *ha sido introducido por instanciación.*

Cuando trabajamos con estrategias independientes del entorno, la condición $\varphi(g_2\sigma') \subseteq \varphi(g_2)$ permite formular el *narrowing* genérico con estrategia, introducido en la Definición 2.3.1, en la forma:

$$\frac{\begin{array}{l}u \in \varphi(g) \;\; \wedge \;\; r \equiv (\lambda \;\to\; \rho \;\Leftarrow\; C) \ll \mathcal{R} \;\; \wedge \;\; \sigma = mgu(\{(g_{|u})\theta = \lambda\}) \\ \wedge \;\; \sigma_r = restrict_\varphi(\sigma, r) \;\; \wedge \;\; g' = (C\sigma_r, g[\rho\sigma_r]_u)\end{array}}{\langle g, \theta\rangle \;\leadsto_\varphi\; \langle g', \theta\sigma\rangle}$$

donde la sustitución σ_r sólo se aplica a la condición C y a la parte derecha ρ de la regla empleada en el paso de derivación. De forma análoga, para estrategias independientes del entorno la relación $\leadsto_\varphi^{\Uparrow}$, introducida en la Definición 3.2.3, se puede reformular como sigue:

$$\frac{\begin{array}{l}u \in \varphi(g) \;\; \wedge \;\; r \equiv (\lambda \;\to\; \rho \;\Leftarrow\; C) \ll \mathcal{R} \;\; \wedge \;\; \delta = mgu(\{g_{|u} = \lambda\}) \\ \wedge \;\; \delta_r = restrict_\varphi(\delta, r) \;\; \wedge \;\; \theta \Uparrow \delta \not\equiv fail\end{array}}{\langle g, \theta\rangle \;\leadsto_\varphi^{\Uparrow}\; \langle (C\delta_r, g[\rho\delta_r]_u), \theta \Uparrow \delta\rangle}$$

En lo que sigue, haremos uso de estas equivalencias cuando sea conveniente.

El motivo de la restricción a estrategias independientes del entorno se puede entender de forma precisa en la demostración del siguiente teorema. A continuación, veremos también un ejemplo que ilustra la necesidad de esta propiedad para el resultado de composicionalidad.

Hacemos notar que no vamos a considerar en la demostración del teorema el problema de los distintos renombramientos de las reglas del programa en las computaciones paralelas para g_1 y g_2, que pueden causar conflictos entre las variables de las respuestas computadas para los subobjetivos paralelos. Para soluciones clásicas a este problema ver, por ejemplo, [dBP90, Sar91].

Teorema 3.2.6 *Sea φ una estrategia de narrowing independiente del entorno. Dados dos objetivos ecuacionales g_1 y g_2, se cumple que*[1]*:*

$$\begin{aligned}&\langle (g_1, g_2), \sigma\rangle \leadsto^n_\varphi \langle g', \sigma\theta\rangle \quad \textit{sii}\\ &\qquad \langle g_1, \sigma\rangle \leadsto^{n_1}_\varphi \langle g'_1, \sigma\theta_1\rangle \quad \textit{y} \quad \langle g_2, \sigma\rangle \leadsto^{n_2}_\varphi \langle g'_2, \sigma\theta_2\rangle,\\ &\qquad \theta = \theta_1 \Uparrow \theta_2 \not\equiv \mathit{fail}, \quad n = n_1 + n_2 \quad \textit{y} \quad g' = (g'_1, g'_2).\end{aligned}$$

Demostración.
En la prueba, consideramos los objetivos ecuacionales módulo reordenamiento de ecuaciones. Esto nos permite suponer, a lo largo de la demostración, que la ocurrencia de un término en el objetivo compuesto (g_1, g_2) se corresponde con la ocurrencia del mismo término en el objetivo g_1 ó g_2. Asimismo, realizamos la prueba usando la variante $\leadsto^{\Uparrow}_\varphi$ de la relación de *narrowing* genérico (denotada simplemente como $\leadsto_\varphi$).

($\Rightarrow$) Sea $\langle (g_1, g_2), \sigma\rangle \equiv \langle h_1, \sigma \Uparrow \vartheta_1\rangle \leadsto_\varphi \ldots \leadsto_\varphi \langle h_n, \sigma \Uparrow \vartheta_n\rangle \leadsto_\varphi \langle g', \sigma \Uparrow \theta\rangle$, $(\vartheta_1 \equiv \epsilon)$.
Demostramos el lema por inducción sobre la longitud n de la derivación.
El caso base $(n = 1)$ es inmediato. Consideramos, pues, el caso inductivo $(n > 1)$. Si existe la derivación:
$\langle (g_1, g_2), \sigma\rangle \equiv \langle h_1, \sigma \Uparrow \vartheta_1\rangle \leadsto_\varphi \ldots \leadsto_\varphi \langle h_n, \sigma \Uparrow \vartheta_n\rangle \leadsto_\varphi \langle g', \sigma \Uparrow \vartheta_n \Uparrow \delta\rangle \equiv \langle g', \sigma \Uparrow \theta\rangle$,
entonces existe una regla $r \equiv (\lambda \to \rho \Leftarrow C) \ll \mathcal{R}$ y una ocurrencia $u \in \varphi(h_n)$ tales que $\delta = mgu(\{h_{n|u} = \lambda\}) \not\equiv \mathit{fail}$ y $g' = (C, h_n[\rho]_u)\delta_r$.
Por la hipótesis de inducción, existen las derivaciones:
$\langle g_1, \sigma\rangle \equiv \langle g_{11}, \sigma \Uparrow \vartheta_{11}\rangle \leadsto_\varphi \ldots \leadsto_\varphi \langle g_{1m}, \sigma \Uparrow \vartheta_{1m}\rangle$ y
$\langle g_2, \sigma\rangle \equiv \langle g_{21}, \sigma \Uparrow \vartheta_{21}\rangle \leadsto_\varphi \ldots \leadsto_\varphi \langle g_{2k}, \sigma \Uparrow \vartheta_{2k}\rangle$, $(\vartheta_{11} \equiv \vartheta_{21} \equiv \epsilon)$,
tales que $\vartheta_n = \vartheta_{1m} \Uparrow \vartheta_{2k} \not\equiv \mathit{fail}$, $n - 1 = (m - 1) + (k - 1)$ y $h_n = (g_{1m}, g_{2k})$.
Ya que $u \in \varphi(h_n)$ y $h_n = (g_{1m}, g_{2k})$, entonces $u \in \varphi(g_{1m})$ o bien $u \in \varphi(g_{2k})$. Sea $u \in \varphi(g_{1m})$ (el caso $u \in \varphi(g_{2k})$ es perfectamente análogo). Ahora, es suficiente con demostrar que $\langle g_{1m}, \sigma \Uparrow \vartheta_{1m}\rangle \leadsto_\varphi \langle g'_1, \sigma \Uparrow \theta_1\rangle$, con $\theta = \theta_1 \Uparrow \vartheta_{2k} \not\equiv \mathit{fail}$ y $g' = (g'_1, g_{2k})$, ya que $n = n_1 + n_2$ es obvio para $n_1 = m$ y $n_2 = k - 1$.
Ya que $\sigma \Uparrow \vartheta_n \Uparrow \delta \not\equiv \mathit{fail}$ y $\vartheta_n = \vartheta_{1m} \Uparrow \vartheta_{2k}$, entonces $\vartheta_{1m} \Uparrow \vartheta_{2k} \Uparrow \delta \not\equiv \mathit{fail}$. Ahora, ya que $u \in \varphi(g_{1m})$, entonces $\langle g_{1m}, \sigma \Uparrow \vartheta_{1m}\rangle \leadsto_\varphi \langle g'_1, \sigma \Uparrow \vartheta_{1m} \Uparrow \delta\rangle \equiv \langle g'_1, \sigma \Uparrow \theta_1\rangle$, donde $\theta_1 = \vartheta_{1m} \Uparrow \delta$ y $g'_1 = (C, g_{1m}[\rho]_u)\delta_r$. Entonces, $\theta_1 \Uparrow \vartheta_{2k} = (\vartheta_{1m} \Uparrow \delta) \Uparrow \vartheta_{2k} = \vartheta_{1m} \Uparrow \vartheta_{2k} \Uparrow \delta = \vartheta_n \Uparrow \delta = \theta$ (por la conmutatividad y asociatividad de $\Uparrow$).
Finalmente, puesto que $h_n = (g_{1m}, g_{2k})$ y φ es independiente del entorno, entonces $g' = (C, h_n[\rho]_u)\delta_r = (C\delta_r, h_n[\rho\delta_r]_u) = (C\delta_r, g_{1m}[\rho\delta_r]_u, g_{2k}) = (g'_1, g_{2k})$, módulo reordenamiento de ecuaciones.

($\Leftarrow$) Consideremos las derivaciones:
$\langle g_1, \sigma\rangle \equiv \langle g_{10}, \sigma \Uparrow \vartheta_{10}\rangle \leadsto_\varphi \ldots \leadsto_\varphi \langle g_{1m}, \sigma \Uparrow \vartheta_{1m}\rangle \leadsto_\varphi \langle g'_1, \sigma \Uparrow \vartheta_{1m} \Uparrow \delta\rangle \equiv$

[1]Denotamos por $\leadsto^n_\varphi$ una derivación de longitud n.

$\langle g_1', \sigma \Uparrow \theta_1 \rangle$ y
$\langle g_2, \sigma \rangle \equiv \langle g_{20}, \sigma \Uparrow \vartheta_{20} \rangle \leadsto_\varphi \ldots \leadsto_\varphi \langle g_{2k}, \sigma \Uparrow \vartheta_{2k} \rangle \equiv \langle g_2', \sigma \Uparrow \theta_2 \rangle$, $(\vartheta_{10} \equiv \vartheta_{20} \equiv \epsilon)$,
con $(m+1)+k = n$, $n \geq 1$, $m, k \geq 0$.
Realizamos la prueba por inducción sobre la suma n de las longitudes de las derivaciones.
El caso base ($n = 1$) es inmediato. Consideramos, pues, el caso inductivo ($n > 1$). Ya que $\langle g_{1m}, \sigma \Uparrow \vartheta_{1m} \rangle \leadsto_\varphi \langle g_1', \sigma \Uparrow \vartheta_{1m} \Uparrow \delta \rangle$, entonces existe una regla $r \equiv (\lambda \to \rho \Leftarrow C) \ll \mathcal{R}$ y una ocurrencia $u \in \varphi(g_{1m})$ tales que $\delta = mgu(\{g_{1m|u} = \lambda\})$ y $g_1' = (C, g_{1m}[\rho]_u)$.
Por la hipótesis de inducción, existe una derivación:

$$\langle (g_1, g_2), \sigma \rangle \leadsto_\varphi^{n-1} \langle h_{n-1}, \sigma \Uparrow (\vartheta_{1m} \Uparrow \theta_2) \rangle,$$

con $h_{n-1} = (g_{1m}, g_2')$. Entonces, es suficiente con probar la transición:

$$\langle h_{n-1}, \sigma \Uparrow (\vartheta_{1m} \Uparrow \theta_2) \rangle \leadsto_\varphi \langle h_n, \sigma \Uparrow (\theta_1 \Uparrow \theta_2) \rangle.$$

Ya que $\theta = (\theta_1 \Uparrow \theta_2) = (\vartheta_{1m} \Uparrow \delta \Uparrow \theta_2) \not\equiv fail$, entonces $\vartheta_{1m} \Uparrow \theta_2 \Uparrow \delta \not\equiv fail$ (por la conmutatividad y asociatividad de $\Uparrow$) y, por tanto, $\langle h_{n-1}, \sigma \Uparrow (\vartheta_{1m} \Uparrow \theta_2) \rangle \leadsto_\varphi \langle g', \sigma \Uparrow (\vartheta_{1m} \Uparrow \theta_2) \Uparrow \delta \rangle$. Ahora, ya que $\theta_1 = \vartheta_{1m} \Uparrow \delta$, tenemos que $(\vartheta_{1m} \Uparrow \theta_2) \Uparrow \delta = (\vartheta_{1m} \Uparrow \delta) \Uparrow \theta_2 = \theta_1 \Uparrow \theta_2$. De forma análoga al caso anterior, se cumple trivialmente que $g' = (g_1', g_2')$ bajo reordenamiento de ecuaciones, y que $n = n_1 + n_2$, con $n_1 = m + 1$ y $n_2 = k$. $\square$

Ilustramos ahora, mediante un ejemplo, la relación entre la restricción a estrategias independientes del entorno y la propiedad de composicionalidad AND.

Ejemplo 5 *Consideremos la siguiente derivación de éxito para el objetivo* (g_1, g_2)*:*

$$\langle (g_1, g_2), \epsilon \rangle \leadsto_\varphi^* \langle (\top, g_2\sigma'), \sigma \rangle \leadsto_\varphi^* \langle \top, \sigma\theta \rangle$$

junto con las derivaciones independientes:

$$\langle g_1, \epsilon \rangle \leadsto_\varphi^* \langle \top, \sigma \rangle$$
$$\langle g_2, \epsilon \rangle \leadsto_\varphi^* \langle \top, \delta \rangle.$$

Resulta fácil comprobar que, en general, los estados computados independientemente contendrán en la parte entorno una sustitución más general que en el caso secuencial. Este es el caso de los estados $\langle (\top, g_2\sigma'), \sigma \rangle$ *y* $\langle g_2, \epsilon \rangle$*, donde* $\epsilon \leq \sigma$*. Para obtener el resultado de composicionalidad deseado, es necesario que cualquier paso de narrowing que se pueda realizar sobre* $\langle (\top, g_2\sigma'), \sigma \rangle$*, se pueda realizar también sobre* $\langle g_2, \epsilon \rangle$*. Por tanto, toda ocurrencia* $u \in \varphi(g_2\sigma')$ *debe pertenecer también a* $\varphi(g_2)$*. Es decir, la sustitución* σ *(obtenida al explotar las ecuaciones de* g_1*) no debe introducir nuevos redexes en* g_2*. En general, sí puede ocurrir lo contrario, es decir, sí pueden aparecer* *nuevos redexes en las derivaciones independientes que no se correspondan con los de la derivación secuencial para* (g_1, g_2)*. Esto, sin embargo, no es ningún problema ya que el operador de composición paralela* $\Uparrow$ *se encarga de eliminar dichas computaciones inconsistentes.*

El siguiente ejemplo muestra que la relación de *narrowing* ordinario no es independiente del entorno (ni siquiera en el caso incondicional).

Ejemplo 6 *Sea* $\mathcal{R}$ *el siguiente programa:*

$$\begin{array}{rcl} twist(0) & \rightarrow & s(0) \\ twist(s(0)) & \rightarrow & 0 \\ twist(one(y)) & \rightarrow & 0 \\ one(0) & \rightarrow & s(0) \\ one(s(y)) & \rightarrow & one(y) \end{array}$$

y consideremos el objetivo $g \equiv (twist(x) = 0,\ x = twist(0))$*. Entonces, existe la siguiente derivación por narrowing (ordinario):*

$$\langle(\underline{twist(x)} = 0,\ x = twist(0)), \epsilon\rangle \leadsto_{\square} \langle(0 = 0,\ one(y) = twist(0)), \{x/one(y)\}\rangle.$$

La estrategia $\varphi_{\square}$ *no es independiente del entorno, ya que:*

$$\varphi_{\square}(one(y) = twist(0)) = \{1.1, 1.2\} \not\subseteq \{1.2\} = \varphi_{\square}(x = twist(0)).$$

A continuación mostramos que las estrategias de *narrowing* condicional básico, *narrowing* condicional "innermost" y *narrowing* perezoso sí son independientes del entorno.

Lema 3.2.7 *La estrategia de narrowing condicional básico* $\varphi_{\blacksquare}$ *es independiente del entorno.*

Demostración. El resultado es trivial, ya que la sustitución computada nunca se aplica en la parte esqueleto de los estados. Concretamente, la fución $restrict_{\blacksquare}(\sigma, r) = \epsilon$ para toda regla r y sustitución σ. □

Lema 3.2.8 *La estrategia de narrowing condicional "innermost"* $\varphi_{\blacktriangleleft}$ *es independiente del entorno.*

Demostración. De forma análoga al caso básico, el resultado se sigue del hecho de que la sustitución computada nunca se aplica en la parte esqueleto de los estados (i.e. $restrict_{\blacktriangleleft}(\sigma, r) = \epsilon$ para toda regla r y sustitución σ). □

Lema 3.2.9 *La estrategia de narrowing condicional perezoso* $\varphi_{\blacktriangleright}$ *es independiente del entorno.*

DEMOSTRACIÓN. Ahora, la función $restrict_{\blacktriangleright}$ sí puede devolver una sustitución que podría introducir nuevas ocurrencias explotables. Consideremos la transición:

$$\langle (g_1, g_2), \theta \rangle \leadsto_{\blacktriangleright} \langle (g_1', g_2)\sigma_r, \theta\sigma \rangle$$

donde $\sigma = mgu(\{g_{1|u}\theta = \lambda\})$ y $\sigma_r = restrict_{\blacktriangleright}(\sigma, r) = \sigma_{\lceil Var(r)}$ (siendo r la regla empleada en la reducción). Puesto que, por la definición de la función $restrict_{\blacktriangleright}$, la sustitución σ_r sólo puede afectar a las variables de la regla r, entonces se cumple que $g_2\sigma_r = g_2$. Por tanto, para toda derivación:

$$\langle (g_1, g_2), \theta \rangle \leadsto^*_{\blacktriangleright} \langle (g_1', g_2)\delta', \theta\delta \rangle$$

se cumple que $g_2\delta' = g_2$ y, por tanto, trivialmente se cumple la condición de independencia del entorno $\varphi_{\blacktriangleright}(g_2\delta') \subseteq \varphi_{\blacktriangleright}(g_2)$. □

Composicionalidad AND

Establecida la independiencia del entorno de una estrategia, resulta sencillo demostrar la composicionalidad AND de la correspondiente relación de *narrowing* como consecuencia del Teorema 3.2.6.

Corolario 3.2.10 *Sea $\mathcal{R}$ un programa, g_1, g_2 dos objetivos ecuacionales y φ una estrategia independiente del entorno. Entonces,*

$$\mathcal{O}^{\varphi}_{\mathcal{R}}(g_1, g_2) = \mathcal{O}^{\varphi}_{\mathcal{R}}(g_1) \Uparrow \mathcal{O}^{\varphi}_{\mathcal{R}}(g_2).$$

DEMOSTRACIÓN. La prueba es un caso particular del Teorema 3.2.6 para $\sigma \equiv \epsilon$ y $g_1 \equiv g_2 \equiv \top$. □

El resultado enunciado en el Corolario 3.2.10 se aplica directamente a las semánticas $\mathcal{O}^{\blacksquare}_{\mathcal{R}}$, $\mathcal{O}^{\blacktriangleleft}_{\mathcal{R}}$ y $\mathcal{O}^{\blacksquare}_{\mathcal{R}}$ puesto que, por los Lemas 3.2.7, 3.2.8 y 3.2.9, dichas estrategias son independientes del entorno. Queda establecida, de esta forma, la composicionalidad AND de las estrategias de *narrowing* condicional básico, *narrowing* condicional "innermost" y *narrowing* perezoso con respecto al operador $\Uparrow$ de composición paralela.

Como hemos comentado, estos resultados de composicionalidad dependen de forma esencial del hecho de estar tratando con una estrategia independiente del entorno. Veamos un ejemplo que muestra la no composicionalidad del *narrowing* ordinario usando el operador (sintáctico) de composición paralela $\Uparrow$.

Ejemplo 7 *Sea $\mathcal{R}$ el programa del Ejemplo 6, y consideremos de nuevo el objetivo $g \equiv (twist(x) = 0,\ x = twist(0))$. Entonces, existe la siguiente derivación por* *narrowing (ordinario):*

$$\begin{array}{rl}\langle (twist(x) = 0, x = \underline{twist(0)}), \epsilon\rangle & \leadsto_{\square} \langle (\underline{twist(x)} = 0, x = s(0)), \epsilon\rangle \\ & \leadsto_{\square} \langle (0 = 0, \underline{one(y)} = s(0)), \{x/one(y)\}\rangle \\ & \leadsto_{\square} \langle (0 = 0, \underline{s(0) = s(0)}), \{x/one(0), y/0\}\rangle \\ & \leadsto_{\square} \langle (\underline{0 = 0}, true), \{x/one(0), y/0\}\rangle \\ & \leadsto_{\square} \langle \top, \{x/one(0), y/0\}\rangle \end{array}$$

con sustitución de respuesta computada $\theta = \{x/one(0)\}$. *Sin embargo, ya que el objetivo* $g_1 \equiv (x = twist(0))$ *sólo computa las sustituciones* $\{\{x/twist(0)\}, \{x/s(0)\}\}$, *la solución* θ *no puede ser obtenida por la composición (sintáctica) de las sustituciones de respuesta computadas para* g_1 *y* g_2 *independientemente.*

Completitud Fuerte

Antes de finalizar este apartado vamos a demostrar que, si una estrategia de *narrowing* posee la propiedad de composicionalidad AND, entonces el resultado de completitud de dicha estrategia se puede enunciar como completitud fuerte (*strong completeness*). La completitud fuerte significa completitud independientemente de la función de selección de ecuaciones empleada [Höl89, OMI95], i.e. no se pierde completitud cuando se restringen las aplicaciones de la regla de *narrowing* a una sola ecuación en cada paso (las ecuaciones del objetivo se pueden seleccionar de forma indeterminista *dont't care*). Introducimos, en primer lugar, la noción de *función de selección.*

Definición 3.2.11 (función de selección [OMI95]**)** *Una* función de selección *es una aplicación que asigna a todo objetivo* $g \equiv (e_1, \ldots, e_n)$, $n > 0$, *un número natural* $i \in \{1, \ldots, n\}$ *denotando una ecuación* e_i *de* g *distinta de true.*

Decimos que una derivación de narrowing $\mathcal{D}$ respeta *una función de selección* $\mathcal{S}$, *si el redex seleccionado en cada paso* $\langle g, \sigma\rangle \leadsto_{\varphi} \langle g', \theta\rangle$ *de* $\mathcal{D}$ *pertenece a la ecuación* $\mathcal{S}(g)$.

Necesitamos, en primer lugar, el siguiente lema previo en el que se establece la conmutatividad de los pasos de *narrowing* para estrategias independientes del entorno.

Lema 3.2.12 *Sea* $\mathcal{R}$ *un programa y* φ *una estrategia independiente del entorno. Dado el objetivo ecuacional* (g_1, g_2), *entonces toda derivación de la forma:*

$$\langle (g_1, g_2), \sigma\rangle \leadsto_{\varphi_{[u_1, r_1]}} \langle (g_1', g_2)\theta_r, \sigma \Uparrow \theta\rangle \leadsto_{\varphi_{[u_2, r_2]}} \langle (g_1', g_2')\theta_r\delta_r, \sigma \Uparrow \theta \Uparrow \delta\rangle$$

donde $u_1 \in \varphi(g_1)$ *y* $u_2 \in \varphi((g_1', g_2)\theta_r) - \varphi(g_1'\theta_r)$, *se puede reordenar de la forma:*

$$\langle (g_1, g_2), \sigma\rangle \leadsto_{\varphi_{[u_2, r_2]}} \langle (g_1, g_2')\delta_r', \sigma \Uparrow \delta'\rangle \leadsto_{\varphi_{[u_1, r_1]}} \langle (g_1', g_2')\delta_r'\theta_r', \sigma \Uparrow \delta' \Uparrow \theta'\rangle$$

tal que $(g_1', g_2')\theta_r\delta_r = (g_1', g_2')\delta_r'\theta_r'$ *y* $\sigma \Uparrow \theta \Uparrow \delta = \sigma \Uparrow \delta' \Uparrow \theta'$.

DEMOSTRACIÓN. En la prueba, consideramos los objetivos ecuacionales módulo reordenamiento de ecuaciones. Esto nos permite suponer a lo largo de la demostración que la ocurrencia de un término en el objetivo compuesto (g_1, g_2) se corresponde con la ocurrencia del mismo término en el objetivo g_1 ó g_2.

Consideremos la derivación:

$$\langle (g_1, g_2), \sigma\rangle \leadsto_{\varphi[u_1, r_1]} \langle (g_1', g_2)\theta_r, \sigma \Uparrow \theta\rangle \leadsto_{\varphi[u_2, r_2]} \langle (g_1', g_2')\theta_r\delta_r, \sigma \Uparrow \theta \Uparrow \delta\rangle$$

donde $u_1 \in \varphi(g_1)$ y $u_2 \in \varphi(g_2\theta_r)$. Entonces existen $r_1 \equiv (\lambda_1 \to \rho_1 \Leftarrow C_1) \ll \mathcal{R}$, $\theta = mgu(\{g_{1|u_1} = \lambda_1\})$, $\theta_r = restrict_\varphi(\theta, r_1)$ y $r_2 \equiv (\lambda_2 \to \rho_2 \Leftarrow C_2) \ll \mathcal{R}$, $\delta = mgu(\{g_{2|u_2}\theta_r = \lambda_2\})$, $\delta_r = restrict_\varphi(\delta, r_2)$ tales que $g_1' = (C_1, g_1[\rho_1]_{u_1})$ y $g_2' = (C_2, g_2[\rho_2]_{u_2})$. Dado que φ es una estrategia independiente del entorno, es posible escribir la anterior derivación de la forma:

$$\begin{array}{lll} \langle (g_1, g_2), \sigma\rangle & \leadsto_{\varphi[u_1, r_1]} & \langle (C_1\theta_r, g_1[\rho_1\theta_r]_{u_1}, g_2), \sigma \Uparrow \theta\rangle \\ & \leadsto_{\varphi[u_2, r_2]} & \langle (C_2\delta_r, C_1\theta_r, g_1[\rho_1\theta_r]_{u_1}, g_2[\rho_2\delta_r]_{u_2}), \sigma \Uparrow \theta \Uparrow \delta\rangle. \end{array}$$

Por tanto, $g_2 = g_2\theta_r$ y $\delta = mgu(\{g_{2|u_2} = \lambda_2\})$ y, por la definición de la relación de *narrowing*, existe la derivación:

$$\begin{array}{lll} \langle (g_1, g_2), \sigma\rangle & \leadsto_{\varphi[u_2, r_2]} & \langle (C_2\delta_r, g_1, g_2[\rho_2\delta_r]_{u_2}), \sigma \Uparrow \delta\rangle \\ & \leadsto_{\varphi[u_1, r_1]} & \langle (C_1\theta_r, C_2\delta_r, g_1[\rho_1\theta_r]_{u_1}, g_2[\rho_2\delta_r]_{u_2}), \sigma \Uparrow \delta \Uparrow \theta\rangle. \end{array}$$

Ahora, $(C_2\delta_r, C_1\theta_r, g_1[\rho_1\theta_r]_{u_1}, g_2[\rho_2\delta_r]_{u_2}) = (C_1\theta_r, C_2\delta_r, g_1[\rho_1\theta_r]_{u_1}, g_2[\rho_2\delta_r]_{u_2})$ bajo reordenamiento de ecuaciones, y $\sigma \Uparrow \theta \Uparrow \delta = \sigma \Uparrow \delta \Uparrow \theta$, por la conmutatividad del operador $\Uparrow$. $\square$

El siguiente teorema establece la completitud fuerte de una estrategia de *narrowing* composicional bajo las mismas condiciones exigidas para garantizar la completitud de ésta.

Teorema 3.2.13 (completitud fuerte) *Sea $\mathcal{R}$ un programa, $\mathcal{S}$ una función de selección y φ una estrategia independiente del entorno. Entonces, para toda derivación $\mathcal{D} \equiv \langle g, \epsilon\rangle \leadsto_\varphi^* \langle \top, \theta\rangle$, existe una derivación $\mathcal{D}' \equiv \langle g, \epsilon\rangle \leadsto_\varphi^* \langle \top, \theta\rangle$ que respeta $\mathcal{S}$.*

DEMOSTRACIÓN. La prueba es inmediata a partir del Lema 3.2.12 ya que, aplicando repetidamente pasos de conmutación, es posible reordenar cualquier derivación $\mathcal{D}$ en una nueva derivación $\mathcal{D}'$ que respete $\mathcal{S}$. $\square$

En el siguiente punto, explotamos la composicionalidad de la semántica para incorporar paralelismo en el cálculo de *narrowing*.

3.2.2 Narrowing Composicional

En este apartado, formalizamos una caracterización composicional de la semántica operacional de los programas lógico-funcionales, siguiendo un estilo similar al presentado en [Pal90] para programas lógicos. La idea básica consiste en permitir que distintas ecuaciones del objetivo sean reducidas independientemente, "combinando" a continuación las sustituciones obtenidas. Este modelo de *narrowing* paralelo fue planteado por primera vez en [Red85].

Sin embargo, el modelo de computación resultante no es siempre equivalente al modelo secuencial, i.e. no calcula el mismo conjunto de respuestas computadas. Para ello, un requisito indispensable es que la relación de *narrowing* que se emplea como base del cálculo paralelo posea la propiedad de composicionalidad AND. Veamos, primero, cómo extender la relación de *narrowing* genérico con estrategia para permitir la reducción paralela de las ecuaciones del objetivo.

Definición 3.2.14 (narrowing composicional con estrategia $\longmapsto_\varphi$)
Dado un programa $\mathcal{R}$, definimos la relación de narrowing composicional con estrategia $\longmapsto_\varphi$ como la menor relación que satisface[2]:

$$(1)\quad \frac{}{\langle true, \theta\rangle \longmapsto_\varphi \langle true, \theta\rangle}$$

$$(2)\quad \frac{\begin{array}{l} u \in \varphi(e) \ \wedge\ r \equiv (\lambda \to \rho \Leftarrow C) \ll \mathcal{R} \ \wedge\ \sigma = mgu(\{e_{|u}\theta = \lambda\}) \\ \wedge\ \sigma_r = restrict_\varphi(\sigma, r) \ \wedge\ g_e = (C, e[\rho]_u)\sigma_r \end{array}}{\langle e, \theta\rangle \ \longmapsto_\varphi\ \langle g_e, \theta\sigma\rangle}$$

$$(3)\quad \frac{\langle g_1, \theta\rangle \longmapsto_\varphi \langle g_1', \theta\theta_1\rangle \ \wedge\ \langle g_2, \theta\rangle \longmapsto_\varphi \langle g_2', \theta\theta_2\rangle \ \wedge\ \theta_1 \Uparrow \theta_2 \not\equiv fail}{\langle (g_1, g_2), \theta\rangle \longmapsto_\varphi \langle (g_1', g_2'), \theta(\theta_1 \Uparrow \theta_2)\rangle}$$

Informalmente, en la computación del modelo formalizado por este sistema de transición, todas las ecuaciones en el objetivo ecuacional pueden ser reducidas al mismo tiempo. Entonces, las sustituciones resultantes de dichas computaciones locales se combinan por medio del operador de composición paralela para obtener el resultado global (de cada paso). La regla (1) se introduce simplemente para permitir que el cálculo ignore aquellas ecuaciones del objetivo que ya han sido reducidas a *true*.

De forma análoga a la definición de la relación de *narrowing* genérico con estrategia, el cálculo composicional se puede instanciar para las distintas relaciones de *narrowing* que hemos presentado en la Sección 2.3. De esta forma, las relaciones $\longmapsto_\square$, $\longmapsto_\blacksquare$, $\longmapsto_\blacktriangleleft$ y $\longmapsto_\blacktriangleright$ denotan, respectivamente, las versiones composicionales de las relaciones de *narrowing* condicional ordinario, básico, "innermost" y perezoso. A

[2] Escribiremos $\longmapsto_{\varphi_{[u,r]}}$ cuando sea necesario distinguir la ocurrencia y la regla usadas.

continuación, ilustramos el comportamiento del cálculo tomando como instancia la relación de *narrowing* composicional básico.

Ejemplo 8 *Consideremos de nuevo el programa $\mathcal{R}$ del Ejemplo 6. Dado el objetivo $g \equiv (w = one(y),\ twist(y) = 0)$, existe la siguiente derivación por narrowing composicional básico:*

$\langle (w = one(y),\ twist(y) = 0), \epsilon\rangle \longmapsto_{\blacksquare} \langle (w = one(x),\ 0 = 0), \{y/s(0), x/0\}\rangle$

$\quad$ *(ya que* $\langle w = \underline{one(y)}, \epsilon\rangle \longmapsto_{\blacksquare} \langle w = one(x), \{y/s(x)\}\rangle$,

$\qquad\qquad \langle \underline{twist(y)} = 0, \epsilon\rangle \longmapsto_{\blacksquare} \langle 0 = 0, \{y/s(0)\}\rangle$ *y*

$\qquad\qquad \{y/s(x)\} \Uparrow \{y/s(0)\} = \{y/s(0), x/0\} \not\equiv fail$*)*

$\langle (w = one(x),\ 0 = 0), \{y/s(0), x/0\}\rangle \longmapsto_{\blacksquare} \langle (w = s(0), true), \{y/s(0), x/0\}\rangle$

$\quad$ *(ya que* $\langle w = \underline{one(x)}, \{y/s(0), x/0\}\rangle \longmapsto_{\blacksquare} \langle w = s(0), \{y/s(0), x/0\}\rangle$ *y*

$\qquad\qquad \langle \underline{0 = 0}, \{y/s(0), x/0\}\rangle \longmapsto_{\blacksquare} \langle true, \{y/s(0), x/0\}\rangle$*)*

$\langle (w = s(0), true), \{y/s(0), x/0\}\rangle \longmapsto_{\blacksquare} \langle \top, \{y/s(0), x/0, w/s(0)\}\rangle$

con sustitución de respuesta computada $\theta = \{y/s(0), w/s(0)\}$.

El modelo de computación formalizado en la Definición 3.2.14 puede ser tomado como base para un modelo de computación AND-paralelo (dependiente) de programas lógico-funcionales. A continuación, se define la semántica operacional del conjunto de éxitos asociada al nuevo cálculo.

Definición 3.2.15 (conjunto de éxitos composicional $\mathcal{C}^{\varphi}_{\mathcal{R}}$) *Dado un programa $\mathcal{R}$ y un objetivo ecuacional g, formulamos la semántica operacional (o conjunto de éxitos) composicional de g con respecto a $\mathcal{R}$ usando la estrategia φ como sigue:*

$$\mathcal{C}^{\varphi}_{\mathcal{R}}(g) = \{\theta_{\lceil Var(g)} \mid \langle g, \epsilon\rangle \longmapsto^{*}_{\varphi} \langle \top, \theta\rangle\},$$

en la que las derivaciones $\langle g, \epsilon\rangle \longmapsto^{*}_{\varphi} \langle \top, \theta\rangle$ se realizan usando las reglas del programa extendido $\mathcal{R}_{+}$.

Establecemos a continuación la equivalencia entre la semántica composicional $\mathcal{C}^{\varphi}$ y la semántica estándar $\mathcal{O}^{\varphi}$. Como era de esperar, dicha equivalencia sólo se cumple para estrategias independientes del entorno, ya que la regla (3) de la relación $\longmapsto_{\varphi}$ requiere la propiedad de composicionalidad AND para no perder la completitud. Necesitamos, en primer lugar, el siguiente lema.

Lema 3.2.16 *Dados dos objetivos ecuacionales g_1 y g_2, se cumple que:*

$$\begin{array}{l} \textit{si} \ \ \langle g_1, \sigma\rangle \longmapsto^{n}_{\varphi} \langle g'_1, \sigma \Uparrow \theta_1\rangle,\ \langle g_2, \sigma\rangle \longmapsto^{n}_{\varphi} \langle g'_2, \sigma \Uparrow \theta_2\rangle \ \ \textit{y} \ \ \theta_1 \Uparrow \theta_2 \not\equiv fail \\ \textit{entonces} \ \ \langle (g_1, g_2), \sigma\rangle \longmapsto^{n}_{\varphi} \langle g', \sigma \Uparrow \theta\rangle,\ \ \textit{con}\ g' = (g'_1, g'_2) \ \ \textit{y} \ \ \theta = \theta_1 \Uparrow \theta_2. \end{array}$$

Demostración. Realizamos la prueba usando una variante $\longmapsto^{\Uparrow}_{\varphi}$ de la relación de *narrowing* composicional (denotada simplemente como $\longmapsto_{\varphi}$), en el estilo de la relación introducida en la Definición 3.2.3. La equivalencia entre la relación $\longmapsto_{\varphi}^{\Uparrow}$ y la relación $\longmapsto_{\varphi}$ se puede demostrar de forma inmediata de manera similar al Lema 3.2.4. Demostramos el lema por inducción sobre la longitud n de las derivaciones.

$n = 1$. El resultado se sigue trivialmente haciendo uso de la regla (3) del cálculo composicional.

Consideremos el caso inductivo $n > 1$. Dadas las derivaciones:
$\langle g_1, \sigma\rangle \longmapsto_{\varphi}^{n-1} \langle h', \sigma \Uparrow \vartheta'\rangle \longmapsto_{\varphi} \langle g_1', \sigma \Uparrow \vartheta' \Uparrow \delta'\rangle \equiv \langle g_1', \sigma \Uparrow \theta_1\rangle$ y
$\langle g_2, \sigma\rangle \longmapsto_{\varphi}^{n-1} \langle h'', \sigma \Uparrow \vartheta''\rangle \longmapsto_{\varphi} \langle g_2', \sigma \Uparrow \vartheta'' \Uparrow \delta''\rangle \equiv \langle g_2', \sigma \Uparrow \theta_2\rangle$,
por la hipótesis de inducción, existe la derivación $\langle (g_1, g_2), \sigma\rangle \longmapsto_{\varphi}^{n-1} \langle h, \sigma \Uparrow \vartheta\rangle$, con $h = (h', h'')$ y $\vartheta = \vartheta' \Uparrow \vartheta''$. Sean $h' = (e_1, \ldots, e_k)$ y $h'' = (e_{k+1}, \ldots, e_m)$ tales que $h = (e_1, \ldots, e_m)$.
Dado que $\langle h', \sigma \Uparrow \vartheta'\rangle \longmapsto_{\varphi} \langle g_1', \sigma \Uparrow \vartheta' \Uparrow \delta'\rangle$ entonces, por las reglas del cálculo composicional, existen $(\lambda_i \to \rho_i \Leftarrow C_i)$, $u_i \in \varphi(e_i)$ y $\delta_i' = mgu(\{e_{i|u_i} = \lambda_i\}) \not\equiv fail$, $i = 1, \ldots, k$, tales que $\delta' = \delta_1' \Uparrow \ldots \Uparrow \delta_k'$. De manera análoga, ya que $\langle h'', \sigma \Uparrow \vartheta''\rangle \longmapsto_{\varphi} \langle g_2', \sigma \Uparrow \vartheta'' \Uparrow \delta''\rangle$ entonces, por las reglas del cálculo composicional, existen $(\lambda_i \to \rho_i \Leftarrow C_i)$, $u_i \in \varphi(e_i)$ y $\delta_i'' = mgu(\{e_{i|u_i} = \lambda_i\}) \not\equiv fail$, $i = k+1, \ldots, m$, tales que $\delta'' = \delta_{k+1}'' \Uparrow \ldots \Uparrow \delta_m''$.
Ahora, es suficiente con demostrar el paso $\langle h, \sigma \Uparrow \vartheta\rangle \longmapsto_{\varphi} \langle g', \sigma \Uparrow \vartheta \Uparrow \delta\rangle \equiv \langle g', \sigma \Uparrow \theta\rangle$. Para ello, consideramos las mismas ocurrencias $u_i \in \varphi(e_i)$, $i = 1, \ldots, m$, para las que deben existir las sustituciones $\delta_i = mgu(\{e_{i|u_i} = \lambda_i\}) \not\equiv fail$, $i = 1, \ldots, m$. Puesto que $\delta_i = \delta_i'$, $i = 1, \ldots, k$ y $\delta_j = \delta_j''$, $j = k+1, \ldots, m$, entonces $\delta = \delta' \Uparrow \delta'' \not\equiv fail$. Ya que $\vartheta = \vartheta' \Uparrow \vartheta''$, entonces $\theta = \vartheta \Uparrow \delta = \vartheta' \Uparrow \vartheta'' \Uparrow \delta' \Uparrow \delta'' = (\vartheta' \Uparrow \delta') \Uparrow (\vartheta'' \Uparrow \delta'') = \theta_1 \Uparrow \theta_2$. Por tanto, es posible probar la transición $\langle h, \sigma \Uparrow \vartheta\rangle \longmapsto_{\varphi} \langle g', \sigma \Uparrow \vartheta \Uparrow \delta\rangle \equiv \langle g', \sigma \Uparrow \theta\rangle$ con $g' = (g_1', g_2')$ y $\theta = \theta_1 \Uparrow \theta_2$, tal y como era requerido. □

Podemos demostrar ahora la equivalencia entre la semántica secuencial y su versión composicional.

Teorema 3.2.17 *Dado un programa $\mathcal{R}$, una estrategia φ independiente del entorno y un objetivo g, se cumple $\mathcal{O}_{\mathcal{R}}^{\varphi}(g) = \mathcal{C}_{\mathcal{R}}^{\varphi}(g)$.*

DEMOSTRACIÓN.
($\subseteq$) Demostramos que, para toda derivación de éxito $\langle g_0, \sigma\rangle \leadsto_{\varphi} \ldots \leadsto_{\varphi} \langle \top, \sigma\vartheta_m\rangle$ tal que $\vartheta_m \not\equiv fail$, existe una derivación de éxito composicional $\langle g_0, \sigma\rangle \longmapsto_{\varphi} \ldots \longmapsto_{\varphi} \langle \top, \sigma\theta_{n_m}\rangle$ tal que $\vartheta_m = \theta_{n_m}$. Realizamos la prueba por inducción sobre la longitud m de la primera derivación. Por simplicidad, asumimos que el renombramiento de las cláusulas elegido en la derivación secuencial coincide con el elegido para la derivación composicional.

Sea $m = 1$. Si $\langle g_0, \sigma\rangle \leadsto_\varphi \langle \top, \sigma\vartheta_1\rangle$, entonces g_0 consta de una sola ecuación y, por la regla (2) del cálculo composicional, $\langle g_0, \sigma\rangle \longmapsto_\varphi \langle \top, \sigma\theta_1\rangle$, con $\theta_1 = \vartheta_1$.

Consideramos ahora el caso inductivo. Si $m > 1$ entonces $\langle g_0, \sigma\rangle \leadsto_\varphi \langle g_1, \sigma\vartheta_1\rangle \leadsto_\varphi \ldots \leadsto_\varphi \langle \top, \sigma\vartheta_m\rangle$, con $\vartheta_m \not\equiv fail$. Consideramos 2 casos:

(i) Sea $g_0 = e$. Por la hipótesis de inducción, existe $\langle g_1, \sigma\vartheta_1\rangle \longmapsto_\varphi \ldots \longmapsto_\varphi \langle \top, \sigma\theta_{n_m}\rangle$ tal que $\vartheta_m = \theta_{n_m}$. Ya que $\langle g_0, \sigma\rangle \leadsto_\varphi \langle g_1, \sigma\vartheta_1\rangle$ y $g_0 = e$, entonces por la regla (2) de la Definición 3.2.14, $\langle g_0, \sigma\rangle \longmapsto_\varphi \langle g_1, \sigma\theta_1\rangle$ con $\theta_1 = \vartheta_1$.

(ii) Sea $g_0 = (g'_0, g''_0)$, donde g'_0 y g''_0 son secuencias de ecuaciones no vacías. Entonces, por el Teorema 3.2.6, existen las derivaciones $\langle g'_0, \sigma\rangle \leadsto_\varphi \ldots \leadsto_\varphi \langle \top, \sigma\vartheta'_i\rangle$ y $\langle g''_0, \sigma\rangle \leadsto_\varphi \ldots \leadsto_\varphi \langle \top, \sigma\vartheta''_k\rangle$ tales que $\vartheta_m = \vartheta'_i \Uparrow \vartheta''_k$, $\vartheta_m \not\equiv fail$, $m > i$ y $m > k$. Por la hipótesis de inducción, existen las derivaciones $\langle g'_0, \sigma\rangle \longmapsto_\varphi \ldots \longmapsto_\varphi \langle \top, \sigma\theta'_{n_i}\rangle$ y $\langle g''_0, \sigma\rangle \longmapsto_\varphi \ldots \longmapsto_\varphi \langle \top, \sigma\theta''_{n_k}\rangle$ tales que $\vartheta'_i = \theta'_{n_i}$ y $\vartheta''_k = \theta''_{n_k}$. Finalmente, por el Lema 3.2.16 (consideramos tantos pasos $\langle \top, \delta\rangle \longmapsto_\varphi \langle \top, \delta\rangle$ como sean necesarios para que $i = k$), existe la derivación $\langle g_0, \sigma\rangle \equiv \langle (g'_0, g''_0), \sigma\rangle \longmapsto_\varphi \ldots \longmapsto_\varphi \langle \top, \sigma\theta_{n_m}\rangle$ tal que $\theta_{n_m} = \theta'_{n_i} \Uparrow \theta''_{n_k} = \vartheta'_i \Uparrow \vartheta''_k = \vartheta_m$.

($\supseteq$) Vamos a demostrar un resultado más general. Concretamente, demostramos que para cada derivación composicional $\langle g_0, \epsilon\rangle \longmapsto_\varphi \ldots \longmapsto_\varphi \langle g_n, \theta_n\rangle$ tal que $\theta_n \not\equiv fail$, existe una derivación secuencial correspondiente $\langle g'_0, \epsilon\rangle \leadsto_\varphi \ldots \leadsto_\varphi \langle g'_{m_n}, \vartheta_{m_n}\rangle$ tal que $g'_{m_n} = g_n$ y $\vartheta_{m_n} = \theta_n$. Demostramos el resultado por inducción sobre la longitud n de la primera derivación. Asumimos la misma consideración respecto al renombramiento de las cláusulas que en el caso anterior.

Sea $n = 1$ y asumimos $g_0 = (e_1, \ldots, e_k)$, $k \geq 1$. Si $\langle g_0, \epsilon\rangle \longmapsto_\varphi \langle g_1, \theta_1\rangle$ entonces, por las reglas de la Definición 3.2.14, existen $r_i \equiv (\lambda_i \to \rho_i \Leftarrow C_i)$, $u_i \in \varphi(e_i)$, $\sigma_i = mgu(\{e_{i|u_i} = \lambda_i\}) \not\equiv fail$ y $\sigma'_i = restrict_\varphi(\sigma_i, r_i)$, $i = 1, \ldots, k$, tales que $\langle e_i, \epsilon\rangle \longmapsto_\varphi \langle (C_i, e_i[\rho_i]_{u_i})\sigma'_i, \sigma_i\rangle \equiv \langle c_i, \sigma_i\rangle$ y $g_1 = (c_1, \ldots, c_k)$, $\theta_1 = \sigma_1 \Uparrow \ldots \Uparrow \sigma_k$, $\theta_1 \not\equiv fail$. Por la Definición 2.3.1, existen las derivaciones $\langle e_i, \epsilon\rangle \leadsto_\varphi \langle c_i, \sigma_i\rangle$. Dado que φ es independiente del entorno, por el Teorema 3.2.6 ($\Leftarrow$), existe la derivación $\langle g_0, \epsilon\rangle \leadsto^*_\varphi \langle g_{m_1}, \vartheta_{m_1}\rangle$, con $g_{m_1} = (c_1, \ldots, c_k) = g_1$ y $\vartheta_{m_1} = \sigma_1 \Uparrow \ldots \Uparrow \sigma_k = \theta_1$.

Consideremos ahora el caso inductivo. Si $n > 1$, entonces $\langle g_0, \epsilon\rangle \longmapsto_\varphi \ldots \longmapsto_\varphi \langle g_{n-1}, \theta_{n-1}\rangle \longmapsto_\varphi \langle g_n, \theta_n\rangle$, con $\theta_n \not\equiv fail$ y, por la hipótesis de inducción, existe $\langle g'_0, \epsilon\rangle \leadsto_\varphi \ldots \leadsto_\varphi \langle g'_{m_{n-1}}, \vartheta_{m_{n-1}}\rangle$ ($g'_0 \equiv g_0$) tal que $g'_{m_{n-1}} = g_{n-1}$ y $\vartheta_{m_{n-1}} = \theta_{n-1}$. Asumimos que $g_{n-1} = (e_1, \ldots, e_k)$, $k \geq 1$. Si $\langle g_{n-1}, \theta_{n-1}\rangle \longmapsto_\varphi \langle g_n, \theta_n\rangle$ entonces, por las reglas de la Definición 3.2.14, existen $r_i \equiv (\lambda_i \to \rho_i \Leftarrow C_i)$, $u_i \in \varphi(e_i)$, $\sigma_i = mgu(\{e_{i|u_i}\theta_{n-1} = \lambda_i\}) \not\equiv fail$ y $\sigma'_i = restrict_\varphi(\sigma_i, r_i)$, $i = 1, \ldots, k$, tales que $\langle e_i, \theta_{n-1}\rangle \longmapsto_\varphi \langle (C_i, e_i[\rho_i]_{u_i})\sigma'_i, \theta_{n-1}\sigma_i\rangle \equiv \langle c_i, \theta_{n-1}\sigma_i\rangle$, $g_n = (c_1, \ldots, c_k)$

y $\theta_n = \theta_{n-1}(\sigma_1 \Uparrow \ldots \Uparrow \sigma_k) \not\equiv fail$. Por último, por la Definición 2.3.1, existen las derivaciones $\langle e_i, \theta_{n-1}\rangle \leadsto_\varphi \langle c_i, \theta_{n-1}\sigma_i\rangle$ y, por el Teorema 3.2.6 ($\Leftarrow$), se cumple $\langle g'_{m_{n-1}}, \vartheta_{m_{n-1}}\rangle \leadsto^*_\varphi \langle g'_{m_n}, \vartheta_{m_n}\rangle$, con $g_{m_n} = (c_1, \ldots, c_k) = g_n$ y $\vartheta_{m_n} = \theta_{n-1}(\sigma_1 \Uparrow \ldots \Uparrow \sigma_k) = \theta_n$, tal y como deseábamos. □

Como consecuencia del Teorema 3.2.17, cuando la estrategia es independiente del entorno, toda solución encontrada por *narrowing* secuencial es encontrada también por su versión composicional. Por tanto, las relaciones de *narrowing* composicional básico, *narrowing* composicional "innermost" y *narrowing* composicional perezoso permiten obtener un conjunto completo de $\mathcal{E}$-unificadores.

Corolario 3.2.18 (completitud) *Dado un programa $\mathcal{R}$, un objetivo g y una estrategia independiente del entorno φ, $\mathcal{C}^\varphi_\mathcal{R}(g)$ es un conjunto completo de $\mathcal{E}$-unificadores*[3] *de g si $\mathcal{O}^\varphi_\mathcal{R}(g)$ lo es.*

DEMOSTRACIÓN. Inmediata por el Teorema 3.2.17. □

Otra consecuencia directa de la equivalencia entre la semántica secuencial y su versión composicional es que, si una de ellas posee la propiedad de composicionalidad AND, entonces la otra también la posee.

Corolario 3.2.19 *Sea $\mathcal{R}$ un programa, g_1, g_2 dos objetivos ecuacionales y φ una estrategia independiente del entorno. Entonces,*

$$\mathcal{C}^\varphi_\mathcal{R}(g_1, g_2) = \mathcal{C}^\varphi_\mathcal{R}(g_1) \Uparrow \mathcal{C}^\varphi_\mathcal{R}(g_2).$$

DEMOSTRACIÓN. Inmediata por el Corolario 3.2.10 y el Teorema 3.2.17. □

El Corolario 3.2.19 se aplica de forma inmediata a las semánticas $\mathcal{C}^{\blacksquare}_\mathcal{R}$, $\mathcal{C}^{\blacktriangleleft}_\mathcal{R}$ y $\mathcal{C}^{\blacksquare}_\mathcal{R}$ puesto que, por los Lemas 3.2.7, 3.2.8 y 3.2.9, dichas estrategias son independientes del entorno. Queda establecida, de esta forma, la composicionalidad AND de las estrategias de *narrowing* composicional básico, *narrowing* composicional "innermost" y *narrowing* composicional perezoso.

3.3 Observaciones

En primer lugar, nótese que el modelo de computación presentado en la Definición 3.2.14 no ha sido definido para conseguir paralelismo maximal, en el sentido de que no todos los *redexes* del objetivo pueden realizar un paso de *narrowing* independientemente. Concretamente, los *redexes* que ocurren en una misma ecuación no pueden ser reducidos en paralelo, mientras que sí podrían hacerlo. Para evitar esta limitación,

[3] Aquí $\mathcal{E}$ denota la teoría de Horn ecuacional axiomatizada por $\mathcal{R}$.

basta con añadir al cálculo la siguiente regla de *aplanamiento*, siempre que preserve las soluciones alcanzables en la correspondiente estrategia:

$$(4) \quad \frac{u \in \varphi(g) \ \wedge \ x \text{ es una nueva variable}}{\langle g, \theta \rangle \longmapsto_{\varphi} \langle (g[x]_u, g_{|u} = x), \theta \rangle}$$

donde $g_{|u}$ y $g[x]_u$ contienen al menos un símbolo de función definido [NRS89]. Por ejemplo, esta regla de aplanamiento preserva las respuestas computadas con respecto a la estrategia básica [NRS89]. De esta forma, se podría incluir en el cálculo composicional (utilizando alguna heurística) para controlar el nivel de granularidad en el paralelismo, sin que la corrección del cálculo se vea afectada.

La aproximación que hemos presentado difiere de otros modelos de ejecución AND-paralelos, como el que aparece en [KMH92], en el que las subexpresiones son reducidas en paralelo únicamente si son *independientes*, es decir, si no comparten variables [HR95]. Se impone un modelo de sincronización "conducido por la demanda" que obliga a los procesos a esperar el valor de una subexpresión paralela si dicho valor es demandado. Por otro lado, en [KH91], se explota un modelo de ejecución AND-paralelo *dependiente* (i.e. se permite la reducción paralela de términos que comparten variables), pero el mecanismo de sincronización impuesto produce demasiada sobrecarga.

El cálculo composicional presentado en la Definición 3.2.14 se puede considerar un modelo de ejecución AND-paralelo dependiente. Este tipo de modelos suele introducir, en general, una pérdida importante de eficiencia respecto a los modelos independientes [Zha94]. Concretamente, se debe utilizar la operación de composición paralela en cada paso (para obtener la solución global, en caso de que las distintas sustituciones generadas sean compatibles) y, además, se puede perder tiempo intentando unificar con reglas que no generen sustituciones compatibles. En el caso de la programación lógica, se han obtenido mejoras significativas sobre el modelo dependiente explotando en paralelo sólo aquellos átomos que unifican (en tiempo de ejecución) con una sola cláusula del programa (átomos deterministas). Este modelo determinista de paralelismo dependiente se usa, por ejemplo, en los lenguaje Andorra-I [Cos91] y P-Prolog [Yan87]. Esto sugiere que la adaptación de esta técnica al caso de programas lógico-funcionales podría incrementar también la eficiencia del cálculo propuesto.

Por último, destacar que, forzando la unión de las soluciones paralelas cada vez que las ecuaciones del objetivo han realizado un paso independientemente, el modelo de ejecución resultante puede no cubrir todo el paralelismo potencialmente explotable. La composicionalidad AND de las semánticas secuenciales (Corolario 3.2.10) sugiere la existencia de otros esquemas de computación posibles. Por ejemplo, podemos resolver en paralelo todas los ecuaciones del objetivo, componiendo sus soluciones únicamente cuando dichas ecuaciones están completamente resueltas, en lugar de forzar la sincronización de las ramas paralelas en cada paso. Este esquema se empleará, posteriormente, para desarrollar un análisis estático composicional de programas lógico-funcionales.

Parte I

Análisis Semántico

Capítulo 4

Narrowing Abstracto

En este capítulo desarrollamos un marco para el análisis estático de programas lógico-funcionales, basado en la idea de construir aproximaciones de la semántica operacional del programa. Formalizamos un esquema de análisis simple, uniforme y potente que admite instancias para diferentes relaciones de *narrowing* (formalizadas mediante sistemas de transición), y permite estudiar distintos tipos de propiedades (relacionadas con el conjunto de éxitos) de manera correcta.

En la Sección 4.1, recordamos brevemente algunos conceptos básicos sobre la teoría de la interpretación abstracta y, en particular, sobre los análisis formalizados en [AFM95, CFM94]. En la Sección 4.2 presentamos los nuevos dominios y operadores abstractos, así como el sistema de transición abstracto que formaliza la relación de *narrowing* aproximado. La corrección del análisis se demuestra de forma general respecto a un programa abstracto, e independientemente de la estrategia de *narrowing* concreta que se aproxima (siempre que sea independiente del entorno). En la Sección 4.3, mostramos una técnica general para asegurar la terminación del análisis. Por último, en la Sección 4.4, formulamos una versión composicional del *narrowing* abstracto que permite introducir incrementalidad y paralelismo en el proceso de análisis.

4.1 Interpretación Abstracta

Los análisis de flujo de datos son un componente esencial de muchas de las herramientas de programación actuales. Sin embargo, los análisis pueden ser muy complejos y, por tanto, difíciles de diseñar y demostrar correctos. La teoría de la *interpretación abstracta*, introducida por Cousot y Cousot [CC77], ofrece alguna ayuda para formalizar la relación entre análisis y semántica. La intuición detrás de la interpretación abstracta, consiste en considerar el análisis como una semántica no estándar, en la que los datos son sustituidos por "descripciones de datos" y a los operadores se les asocia una interpretación no estándar. La interpretación abstracta resulta útil en la medida en que facilita la demostración de la corrección de los análisis existentes, y ayuda en el diseño de nuevos análisis.

Existen varios marcos para la interpretación abstracta, diferenciándose entre sí por la forma en que la interpretación no estándar se relaciona con la interpretación estándar. El marco original de Cousot y Cousot [CC77, CC79] fue desarrollado en el contexto de los lenguajes de programación imperativos. La corrección del análisis se formaliza en términos de una función de "abstracción", que asigna a cada objeto su mejor descripción, y una función de "concretización", que asigna a cada descripción el mayor objeto que ésta describe. Un segundo marco, definido por Burn, Hankin y Abramsky [BHA86] y utilizado por Nielson [Nie88] (desarrollado en el contexto de los lenguajes funcionales), elimina el requerimiento de una función de concretización, exigiendo sólo la existencia de una función de abstracción continua. Un tercer marco, definido por Marriott y Sondergaard [MS89, MS92], y desarrollado en el contexto de los lenguajes de programación lógicos, requiere la existencia de una función de concretización, pero elimina la necesidad de una función de abstracción. Un cuarto marco, definido por Mycroft y Jones [MJ86], permite una relación de aproximación arbitraria y fue desarrollado en el contexto de los lenguajes no recursivos.

Por último, en [Mar93], Marriott define un marco general basado en funciones de concretización y abstracción, pero extendidas de manera que a las descripciones de objetos se les asocia, no elementos individuales, sino conjuntos de elementos. Más aún, no se exige la existencia de ambas funciones. En su máxima generalidad, este marco permite cualquier tipo de relación de aproximación, siendo así equivalente al marco de Mycroft y Jones. En [CFM94], se define una instancia de dicho marco basada en abstraer sistemas de transición, con la que se formaliza una aproximación de la semántica operacional de los lenguajes lógicos concurrentes. Asímismo, los distintos análisis de insatisfacibilidad ecuacional descritos en [AFM95] pueden verse como una instancia del marco de Marriott para programas lógico-funcionales.

Siguiendo una aproximación similar a la presentada en [AFM95, CFM94], definimos la *relación de aproximación* en términos de una *función de concretización*, que asigna a los elementos del dominio no estándar aquellos elementos del dominio estándar que están siendo descritos.

Definición 4.1.1 (descripción [CFM94]**)** *Una* descripción $Desc = \langle D, \gamma, E\rangle$ *consta de un dominio de descripción (un poset) D, un dominio de datos (un poset) E, y una función (anti-)monótona de concretización $\gamma : D \rightarrow \wp E$.*

Cuando el dominio de datos E se corresponde con el dominio de términos $\mathcal{T}$, el dominio de sustituciones Sub o el dominio de objetivos $Goal$, la descripción asociada recibe el nombre de *descripción de términos*, *descripción de sustituciones* o *descripción de objetivos*, respectivamente.

En la Definición 4.1.1, exigimos que la función de concretización γ sea anti-monótona (respecto al orden de inclusión de conjuntos). El motivo de esta restricción es simplemente reflejar la idea de que, cuanto menor es un elemento del dominio abstracto (menos preciso), mayor es el conjunto de elementos del dominio concreto que éste describe. Formalizamos a continuación nuestra noción de *aproximación*, basada en la función de concretización.

Definición 4.1.2 (relación de aproximación [CFM94]**)**
Decimos que d γ-aproxima a e, y lo denotamos $d \propto_\gamma e$, si y sólo si $e \in \gamma(d)$. La relación de aproximación se puede extender a funciones y productos cartesianos como sigue:

- *Dadas dos descripciones $\langle D_1, \gamma_1, E_1\rangle$ y $\langle D_2, \gamma_2, E_2\rangle$, y las funciones $F : D_1 \to D_2$ y $F' : E_1 \to E_2$, decimos que $F \propto F'$ sii $\forall d \in D_1,\ \forall e \in E_1.\ (d \propto_{\gamma_1} e \Rightarrow F(d) \propto_{\gamma_2} F'(e))$.*
- *Dadas las descripciones $\langle D_1, \gamma_1, E_1\rangle$ y $\langle D_2, \gamma_2, E_2\rangle$ y las tuplas $(d_1, d_2) : D_1 \times D_2$ y $(e_1, e_2) : E_1 \times E_2$. Entonces $(d_1, d_2) \propto (e_1, e_2)$ sii $d_1 \propto_{\gamma_1} e_1 \wedge d_2 \propto_{\gamma_2} e_2$.*

Cuando esté claro por el contexto, diremos que d aproxima a e y lo denotaremos simplemente como $d \propto e$. Por abuso de notación, a veces representaremos con $Desc$ tanto una descripción como el dominio de dicha descripción.

El dominio de estados abstractos es paramétrico respecto a los dominios de objetivos y sustituciones abstractos. Cuando en un análisis particular se fijan dichos dominios, el dominio de estados abstractos sobre el que se realizarán las computaciones aproximadas viene dado por la siguiente definición.

Definición 4.1.3 *Sea $Goal_{\mathcal{A}}$ una descripción de objetivos y $Sub_{\mathcal{A}}$ una descripción de sustituciones. Definimos el* dominio de estados abstractos $State_{\mathcal{A}}$*, inducido por $Goal_{\mathcal{A}}$ y $Sub_{\mathcal{A}}$, como $State_{\mathcal{A}} = \{\langle g, \kappa\rangle \mid g \in Goal_{\mathcal{A}}, \kappa \in Sub_{\mathcal{A}}\}$.*

Definimos las funciones $subs : State_{\mathcal{A}} \to Sub_{\mathcal{A}}$ y $goal : State_{\mathcal{A}} \to Goal_{\mathcal{A}}$, asociadas a $State_{\mathcal{A}}$, de la forma: $subs(s) = g$ y $goal(s) = \kappa$, si $s \equiv \langle g, \kappa\rangle$. Las definiciones de subs y goal se extienden de forma natural a conjuntos de estados.

Haciendo uso de la definición estándar de aproximación (Definición 4.1.1), podemos formalizar ahora la relación existente entre los dominios de estados abstractos y concretos.

Definición 4.1.4 (descripción de estados inducida) *Sea $State_{\mathcal{A}}$ un dominio de estados abstractos inducido por una descripción de objetivos $Goal_{\mathcal{A}}$ y una descripción de sustituciones $Sub_{\mathcal{A}}$. Sea $s' \in State_{\mathcal{A}}$ y $s \in State$, entonces $s' \propto s$ sii:*

1. *$subs(s') \propto subs(s)$, y*

2. $goal(s') \propto goal(s)$.

Definimos la función de concretización $\gamma : State_{\mathcal{A}} \to \wp State$ de la forma:

$$\gamma(s') = \{s \in State \mid s' \propto s\}.$$

Así, la descripción de estados inducida por $Goal_{\mathcal{A}}$ y $Sub_{\mathcal{A}}$ es $\langle State_{\mathcal{A}}, \gamma, State\rangle$.

Definimos ahora el concepto de sistema de transición abstracto. La definición es paramétrica con respecto a un dominio de estados abstractos y una relación de reducción entre estados abstractos. Estos elementos deben ser instanciados en un eventual proceso de especialización del marco.

Definición 4.1.5 (sistema de transición abstracto [AFM95]**)**
Sea $\mathcal{R}$ un programa, s un estado y $State_{\mathcal{A}}$ una descripción de estados inducida. Sea $\overset{\mathcal{A}}{\hookrightarrow}_{\mathcal{R}} \subseteq State_{\mathcal{A}} \times State_{\mathcal{A}}$ una relación (dependiente de $\mathcal{R}$). Un sistema de transición abstracto *para $\overset{\mathcal{A}}{\hookrightarrow}_{\mathcal{R}}$, s y $State_{\mathcal{A}}$ es un grafo de transición $\mathcal{G}$ con elementos de $State_{\mathcal{A}}$ como nodos y definida en términos de la relación de reducción abstracta $\overset{\mathcal{A}}{\hookrightarrow}_{\mathcal{R}}$. El grafo $\mathcal{G}$ debe satisfacer las siguientes condiciones:*

- *El nodo fuente de $\mathcal{G}$ es una aproximación de s.*
- *Dados los estados abstractos $t, t' \in State_{\mathcal{A}}$, existe un arco de t a t' sii $t \overset{\mathcal{A}}{\hookrightarrow}_{\mathcal{R}} t'$.*

Un sistema de transición abstracto debe aproximar al sistema de transición que define la semántica operacional concreta. La idea consiste, básicamente, en aproximar el comportamiento de un programa y un estado inicial mediante un sistema de transición abstracto *finito* cuyos nodos están etiquetados con descripciones de estados, de forma que todos los estados de éxito concretos $\langle \top, \theta\rangle$ estén aproximados por estados abstractos del grafo de transición. En la Definición 4.2.14 introducimos una instancia del concepto de sistema de transición abstracto, que aproxima correctamente a la semántica operacional concreta basada en la relación de *narrowing* genérico con estrategia.

4.2 Semántica Operacional Abstracta

En esta sección exponemos los fundamentos del análisis que presentamos. Empezamos definiendo los dominios aproximados y establecemos de forma precisa su relación con los dominios concretos. Posteriormente, estudiamos cómo se deben abstraer los operadores involucrados en el cálculo de *narrowing*, y demostramos la corrección de la aproximación con respecto a los operadores concretos.

Antes de presentar la relación de *narrowing* abstracto queremos puntualizar que, por simplicidad en el desarrollo, vamos a tomar como base para el análisis la relación de *narrowing* $\leadsto^{\Uparrow}_{\varphi}$ (presentada en la Definición 3.2.3). De esta forma, los operadores a aproximar son, esencialmente, el operador de unificación sintáctica mgu y el operador de composición paralela $\Uparrow$.

4.2.1 Dominios Abstractos

La relación de *narrowing* concreta se define sobre estados compuestos por un objetivo y una sustitución. Para definir una relación de *narrowing* aproximado, debemos primero establecer los nuevos dominios abstractos, así como su relación con los dominios concretos. Definimos, en primer lugar, el dominio de los objetivos abstractos. Por abuso de notación, en lo que sigue vamos a denotar de la misma forma un preorden y el correspondiente orden parcial inducido sobre las clases de la relación de equivalencia asociada con el preorden.

Definición 4.2.1 (términos y objetivos abstractos)
Denotamos por $\mathcal{T} = (\tau(\Sigma \cup V), \leq)$ el dominio estándar de (las clases de equivalencia de) términos ordenados por el orden parcial estándar $\leq$ inducido por el preorden sobre términos dado por la relación de ser "más general". Sea $\perp$ un nuevo símbolo de variable tal que $\perp \notin V$. Denotamos por $\mathcal{T}_{\mathcal{A}} = (\tau(\Sigma \cup V \cup \{\perp\}), \preceq)$ el dominio de términos sobre la signatura extendida con el símbolo $\perp$ (términos abstractos), donde el orden parcial $\preceq$ se define como sigue:

$(a)\ \forall t \in \mathcal{T}_{\mathcal{A}}.\ \perp \preceq t\ \wedge\ t \preceq t,\ y$
$(b)\ \forall s_1, \ldots, s_n, t_1, \ldots, t_n \in \mathcal{T}_{\mathcal{A}}, \forall f/n \in \Sigma.$
$$s_1 \preceq t_1 \wedge \ldots \wedge s_n \preceq t_n\ \Rightarrow\ f(s_1, \ldots, s_n) \preceq f(t_1, \ldots, t_n).$$

Sea $Goal_{\mathcal{A}}$ el dominio de los objetivos (abstractos) definidos sobre términos de $\mathcal{T}_{\mathcal{A}}$. Extendemos el orden parcial $\preceq$ a objetivos abstractos como sigue:

$(a)\ \forall s, t, s', t' \in \mathcal{T}_{\mathcal{A}}.\ s' = t' \preceq s = t\ \Leftrightarrow\ s' \preceq s\ \wedge\ t' \preceq t,\ y$
$(b)\ \forall g, g' \in Goal_{\mathcal{A}}.\ g' \preceq g\ \Leftrightarrow\ \forall e' \in g'.\ \exists e \in g.\ e' \preceq e.$

Nótese que el dominio (infinito) de objetivos abstractos es una extensión propia del dominio concreto (i.e. $Goal \subseteq Goal_{\mathcal{A}}$). Informalmente, hemos introducido el símbolo especial $\perp$ para representar a cualquier término. Desde el punto de vista lógico, $\perp$ representa una variable cuantificada existencialmente [AFM95, Mah88, Mah90] y, desde el punto de vista operacional, se puede considerar como una aproximación de la variable anónima de Prolog. Antes de establecer de forma precisa la relación entre el dominio abstracto $Goal_{\mathcal{A}}$ y el dominio concreto $Goal$, introducimos la función auxiliar $[\![\ ]\!] : Goal_{\mathcal{A}} \rightarrow Eqn$. Definimos $[\![g']\!] = E$, donde la n-tupla de ocurrencias de $\perp$ en g' es reemplazada en E por una n-tupla de variables nuevas cuantificadas existencialmente. Los siguientes lemas clarifican la relación entre los órdenes parciales $\preceq$ y $\leq$.

Lema 4.2.2 *Sean $t, t' \in \mathcal{T}_{\mathcal{A}}$. Entonces,*

$$t' \preceq t \Leftrightarrow \forall u \in O(t).\ (\exists w \leq u.\ t'_{|w} \equiv \bot) \vee (u \in O(t') \wedge t'[u] \equiv t[u]).$$

Demostración. Inmediato por la Definición 4.2.1. □

Lema 4.2.3 *Sean e, e' dos ecuaciones abstractas tales que $e' \preceq e$. Entonces $[\![e']\!] \leq [\![e]\!]$.*

Demostración. Demostramos el lema por inducción sobre el número n de ocurrencias de $\bot$ en e'.

- Sea $n = 1$. Asumimos que la ocurrencia a la que aparece $\bot$ es u, es decir, $e'_{|u} \equiv \bot$ y $e_{|u} \equiv t$. Dado que sólo hay una ocurrencia de $\bot$ en e', por el Lema 4.2.2, se cumple que $e'[y]_u \equiv e[y]_u$, donde $y \notin Var(e') \cup Var(e)$. Por tanto, $[\![e]\!] \Leftrightarrow [\![\exists x.\ (e[x]_u \wedge x = t)]\!]$ (donde $x \notin Var(e) \cup Var(e')$) $\Leftrightarrow \exists x.\ (e[x]_u \wedge [\![x = t]\!]) \Rightarrow \exists x.\ (e[x]_u \wedge \exists y.\ x = y)$ (donde $y \notin Var(e) \cup Var(e') \cup \{x\}$) $\Leftrightarrow \exists y.\ (e[x]_u \wedge \exists x.\ x = y) \Rightarrow \exists y.\ e[y]_u \Leftrightarrow [\![e']\!]$. Por definición, $[\![e]\!] \Rightarrow [\![e']\!] \Leftrightarrow [\![e']\!] \leq [\![e]\!]$.
- Sea $n > 1$. Asumimos que $e'_{|u} \equiv \bot$ y $e_{|u} \equiv t$. Entonces, $[\![e]\!] \Leftrightarrow [\![\exists x.\ (e[x]_u \wedge x = t)]\!]$ (donde $x \notin Var(e) \cup Var(e')$) $\Leftrightarrow \exists x.\ ([\![e[x]_u]\!] \wedge [\![x = t]\!]) \Rightarrow \exists x.\ ([\![e'[x]_u]\!] \wedge \exists y.\ x = y)$ (por la hipótesis de inducción, ya que $e'[x]_u$ contiene $n-1$ ocurrencias de $\bot$ y $e'[x]_u \preceq e[x]_u$, donde $y \notin Var(e) \cup Var(e') \cup \{x\}$) $\Leftrightarrow \exists y.\ ([\![e'[y]_u]\!] \wedge \exists x.\ x = y) \Rightarrow \exists y.\ [\![e'[y]_u]\!] \Leftrightarrow [\![e']\!]$. Por definición, $[\![e]\!] \Rightarrow [\![e']\!] \Leftrightarrow [\![e']\!] \leq [\![e]\!]$.

□

La siguiente proposición establece las principales propiedades del orden $\preceq$ entre objetivos abstractos.

Proposición 4.2.4 *Sean $g, g' \in Goal_{\mathcal{A}}$ dos objetivos abstractos tales que $g' \preceq g$. Entonces se cumple:*

1. *$[\![g']\!] \leq [\![g]\!]$ (equivalentemente, $[\![g]\!] \Rightarrow [\![g']\!]$), y*
2. *si $g \equiv \top$, entonces $g' \equiv \top$.*

Demostración.

1. Si $g' \preceq g$, entonces $\forall e' \in g'.\ \exists e \in g.\ e' \preceq e$. Por tanto, existe una función $f : g' \to g$ tal que $\forall e' \in g'.\ e' \preceq f(e')$. Sea $g' \equiv (e'_1, \ldots, e'_n)$ y $g \equiv (e_1, \ldots, e_m)$. Entonces se cumple:

$$\begin{array}{rcl} [\![g']\!] & = & [\![e'_1]\!] \wedge \ldots \wedge [\![e'_n]\!] \\ & \leq & [\![f(e'_1)]\!] \wedge \ldots \wedge [\![f(e'_n)]\!] \text{ (por el Lema 4.2.3)} \\ & \leq & [\![e_1]\!] \wedge \ldots \wedge [\![e_m]\!] \text{ (ya que } \forall e' \in g'.\ \exists e \in g.\ f(e') = e) \\ & = & [\![g]\!] \end{array}$$

2. Si $g' \preceq \top$ entonces, por la Definición 4.2.1, se cumple $\forall e' \in g'.\ e' \preceq true \Leftrightarrow [\![e']\!] \leq true \Leftrightarrow e' \equiv true$, con lo que $g' \equiv \bot$.

□

A continuación, definimos el dominio de sustituciones abstractas $Sub_{\mathcal{A}}$.

Definición 4.2.5 (sustituciones abstractas) *Denotamos por* $(Sub_{\mathcal{A}}, \preceq)$ *el dominio de las sustituciones (abstractas) de la forma* $\{x_1/t_1, \ldots, x_n/t_n\}$ *donde, para todo* $i = 1, \ldots, n$, *las variables* $x_i \in V$ *son distintas entre sí y no aparecen en ninguno de los términos* $t_1, \ldots, t_n \in \mathcal{T}_{\mathcal{A}}$. *Dadas dos sustituciones abstractas* $\theta, \kappa \in Sub_{\mathcal{A}}$, *el preorden* $\preceq$ *se define de la forma:*

$$\kappa \preceq \theta \Leftrightarrow [\![\widehat{\theta}]\!] \Rightarrow [\![\widehat{\kappa}]\!].$$

Como ha quedado establecido en la sección anterior, la relación entre los dominios abstractos y los correspondientes dominios concretos se formaliza en términos de descripciones. Introducimos ahora las descripciones de términos, objetivos y sustituciones.

Definición 4.2.6 (descripción de términos, objetivos y sustituciones) *La* descripción de términos *se denota por* $\langle(\mathcal{T}_{\mathcal{A}}, \preceq), \gamma, (\mathcal{T}, \leq)\rangle$, *donde la función de concretización* $\gamma : \mathcal{T}_{\mathcal{A}} \rightarrow \wp\mathcal{T}$ *se define por:*

$$\gamma(t') = \{t \in \mathcal{T} \mid t' \preceq t\}.$$

La descripción de objetivos *se denota por* $\langle(Goal_{\mathcal{A}}, \preceq), \gamma, (Goal, \leq)\rangle$, *donde la función de concretización* $\gamma : Goal_{\mathcal{A}} \rightarrow \wp Goal$ *se define por:*

$$\gamma(g') = \{g \in Goal \mid g' \preceq g\}.$$

La descripción de sustituciones *se denota por* $\langle(Sub_{\mathcal{A}}, \preceq), \gamma, (Sub, \leq)\rangle$, *donde la función de concretización* $\gamma : Sub_{\mathcal{A}} \rightarrow \wp Sub$ *se define por:*

$$\gamma(\kappa) = \{\theta \in Sub \mid \kappa \preceq \theta\}.$$

A partir de las descripciones de objetivos y sustituciones, se obtiene directamente la descripción de estados inducida, tal y como se formalizó en la Definición 4.1.4.

En el siguiente apartado, presentamos el sistema de transición de *narrowing* condicional abstracto, es decir, la versión aproximada de la relación de *narrowing* genérico con estrategia. En el sistema de transición abstracto, las computaciones no se realizan sobre el programa original sino sobre una versión aproximada del mismo, convenientemente transformado para garantizar la terminación del análisis, mientras todavía permite extraer la información relevante para el análisis. Dicha versión "compilada" del programa original, además de garantizar que el análisis es finito (i.e. que no existen derivaciones abstractas infinitas), debe de cumplir ciertas restricciones, como la de imitar de manera correcta las computaciones concretas (i.e. el programa abstracto debe ser una aproximación correcta del concreto).

De forma similar a como se formuló la descripción de estados inducida (Definición 4.1.4), presentamos a continuación la descripción de programas inducida.

Definición 4.2.7 *Sea $\mathcal{T}_{\mathcal{A}}$ una descripción de términos y $Goal_{\mathcal{A}}$ una descripción de objetivos. Definimos el* dominio de programas abstractos *$Srtc_{\mathcal{A}}$, inducido por $\mathcal{T}_{\mathcal{A}}$ y $Goal_{\mathcal{A}}$, como $Srtc_{\mathcal{A}} = \{\mathcal{R}_{\mathcal{A}} \mid (\lambda \to \rho \Leftarrow C) \in \mathcal{R}_{\mathcal{A}},\ \lambda, \rho \in \mathcal{T}_{\mathcal{A}},\ C \in Goal_{\mathcal{A}}\}$.*

Definición 4.2.8 (descripción de programas inducida) *Sea $Srtc_{\mathcal{A}}$ un dominio de programas abstractos inducido por una descripción de términos $\mathcal{T}_{\mathcal{A}}$ y una descripción de objetivos $Goal_{\mathcal{A}}$. Dados los programas $\mathcal{R}_{\mathcal{A}} \in Srtc_{\mathcal{A}}$ y $\mathcal{R} \in Srtc$, decimos que el programa abstracto $\mathcal{R}_{\mathcal{A}}$ aproxima al programa concreto $\mathcal{R}$, y lo denotamos como $\mathcal{R}_{\mathcal{A}} \propto \mathcal{R}$, sii:*

$$\forall(\lambda \to \rho \Leftarrow C) \in \mathcal{R}.\ \exists(\lambda_{\mathcal{A}} \to \rho_{\mathcal{A}} \Leftarrow C_{\mathcal{A}}) \in \mathcal{R}_{\mathcal{A}}.\ \lambda_{\mathcal{A}} = \lambda \wedge \rho_{\mathcal{A}} \propto \rho \wedge C_{\mathcal{A}} \propto C.$$

Definimos la función de concretización $\gamma : Srtc_{\mathcal{A}} \to \wp Srtc$ de la forma:

$$\gamma(\mathcal{R}_{\mathcal{A}}) = \{\mathcal{R} \in Srtc \mid \mathcal{R}_{\mathcal{A}} \propto \mathcal{R}\}.$$

Así, la descripción de programas inducida por $\mathcal{T}_{\mathcal{A}}$ y $Goal_{\mathcal{A}}$ es $\langle Srtc_{\mathcal{A}}, \gamma, Srtc\rangle$.

Como hemos comentado, la relación de *narrowing* abstracto se define, en general, con respecto a un programa abstracto $\mathcal{R}_{\mathcal{A}}$ (que, en ocasiones, puede coincidir con el programa concreto $\mathcal{R}$). La decisión de abstraer el programa proviene de la necesidad de asegurar la terminación del análisis. Si para un objetivo dado, las computaciones sobre el programa concreto son siempre finitas, el programa abstracto puede coincidir con el concreto. En la práctica, existen varios métodos para estudiar la terminación de *narrowing* respecto a un programa como, por ejemplo, los *grafos de dependencias funcionales* usados en [DS87], o los *grafos de términos* (dependientes del objetivo) usados en [CR91, RKKL85]. En la Sección 4.3.1 presentamos una técnica general, independiente del objetivo y basada en la noción de grafo de "prevención de bucles" para obtener un programa abstracto cuyas computaciones son siempre finitas. En la Sección 4.3.2, se presenta una instancia particular de dicha técnica.

Una vez definidos los dominios abstractos, en el siguiente apartado presentamos la versión abstracta de los operadores concretos.

4.2.2 Narrowing Abstracto

El cálculo de *narrowing* abstracto realiza las computaciones sobre los estados abstractos de $State_{\mathcal{A}}$, usando las reglas de un programa abstracto $\mathcal{R}_{\mathcal{A}}$. Definimos ahora las correspondientes versiones aproximadas de las operaciones involucradas en el cálculo de *narrowing*. Concretamente, los únicos operadores que necesitan ser redefinidos son el operador de unificación sintáctica mgu y el operador de composición paralela $\Uparrow$. El resto de operadores que trabajan sobre los dominios abstractos se definen de la misma forma que sus respectivas versiones concretas, considerando el símbolo $\perp$ como una variable más.

Definimos el operador de unificación abstracta $mgu_{\mathcal{A}}$ para una secuencia de ecuaciones abtractas $g \in Goal_{\mathcal{A}}$ de la siguiente forma. Primero, reemplazamos todas las ocurrencias de $\perp$ en g por variables cuantificadas existencialmente. A continuación, computamos una forma resuelta del conjunto de ecuaciones resultante y, finalmente, volvemos a reemplazar las variables cuantificadas por $\perp$. La siguiente definición formaliza esta idea.

Definición 4.2.9 (unificación abstracta)
Definimos el operador $mgu_{\mathcal{A}} : Goal_{\mathcal{A}} \to Sub_{\mathcal{A}}$ *como sigue. Dado un objetivo abstracto* $g \in Goal_{\mathcal{A}}$, $mgu_{\mathcal{A}}(g) = sub(E\kappa)$, *donde* $\exists y_1 \ldots \exists y_n.\ E = solve([\![g]\!])$ *y* $\kappa \equiv \{y_1/\perp, \ldots, y_n/\perp\}$.

Veamos un ejemplo que ilustra la noción de unificación abstracta.

Ejemplo 9 *Sea* $g \equiv (f(x,x) = f(y,\perp), x = a)$. *En primer lugar, calculamos* $[\![g]\!] = \exists z.\ (f(x,x) = f(y,z) \wedge x = a)$. *Ahora, su forma resuelta* $solve([\![g]\!])$ *es igual a* $(x = a \wedge y = a)$ *y, por tanto,* $mgu_{\mathcal{A}}(g) = \{x/a, y/a\}$.

La siguiente proposición muestra en qué sentido el mgu abstracto es el unificador "más general" de un conjunto de ecuaciones.

Proposición 4.2.10 *Dado un objetivo* $g \in Goal_{\mathcal{A}}$, $\forall \theta \in unif([\![g]\!]).\ mgu_{\mathcal{A}}(g) \preceq \theta$.

DEMOSTRACIÓN. Por la Definición 4.2.9, $[\![mgu_{\mathcal{A}}(g)]\!] \Leftrightarrow solve([\![g]\!])$. Por tanto, $unif(solve([\![g]\!])) = unif([\![g]\!])$. Sea $\theta \in unif([\![mgu_{\mathcal{A}}(g)]\!])$, entonces $\theta \Rightarrow [\![mgu_{\mathcal{A}}(g)]\!]$, es decir, $mgu_{\mathcal{A}}(g) \preceq \theta$. $\square$

A partir de la definición del operador $mgu_{\mathcal{A}}$, resulta inmediato formular la definición del operador abstracto de composición paralela.

Definición 4.2.11 (composición paralela abstracta) *Sean* $\kappa_1, \kappa_2 \in Sub_{\mathcal{A}}$. *Definimos el operador abstracto de composición paralela* $\Uparrow_{\mathcal{A}}: Sub_{\mathcal{A}} \times Sub_{\mathcal{A}} \to Sub_{\mathcal{A}}$ *como sigue:*

$$\kappa_1 \Uparrow_{\mathcal{A}} \kappa_2 = mgu_{\mathcal{A}}(\widehat{\kappa}_1 \cup \widehat{\kappa}_2).$$

Es trivial demostrar que la composición paralela abstracta sigue siendo idempotente, conmutativa, asociativa y tiene como elemento neutro ϵ. El operador $\Uparrow_{\mathcal{A}}$ se puede extender a conjuntos de sustituciones abstractas de manera análoga a como se definió para el operador concreto $\Uparrow$ en la Sección 3.1. En el siguiente lema, demostramos que el operador $mgu_{\mathcal{A}}$ es una aproximación correcta del operador mgu.

Lema 4.2.12 *Sea $g' \in Goal_{\mathcal{A}}$ y $g \in Goal$. Si $g' \propto g$, entonces $mgu_{\mathcal{A}}(g') \propto mgu(g)$.*

Demostración. Si $g' \propto g$ entonces, por la Definición 4.2.6, $g' \preceq g$. Por la Proposición 4.2.4, $(g' \preceq g) \Leftrightarrow (g \Rightarrow [\![g']\!])$. Por tanto, $mgu(g) \Leftrightarrow solve(g) \Rightarrow solve([\![g']\!]) \Leftrightarrow [\![mgu_{\mathcal{A}}(g')]\!]$. Finalmente, si $mgu(g) \Rightarrow [\![mgu_{\mathcal{A}}(g')]\!]$, entonces $mgu_{\mathcal{A}}(g') \preceq mgu(g)$ y, por la Definición 4.2.6, $mgu(g') \propto mgu(g)$. □

La corrección del operador abstracto $\Uparrow_{\mathcal{A}}$ se demuestra en el siguiente lema.

Lema 4.2.13 *Sean $\sigma', \theta' \in Sub_{\mathcal{A}}$ y $\sigma, \theta \in Sub$. Si $\sigma' \propto \sigma$ y $\theta' \propto \theta$, entonces $(\sigma' \Uparrow_{\mathcal{A}} \theta') \propto (\sigma \Uparrow \theta)$.*

Demostración. Si $\sigma' \propto \sigma$ y $\theta' \propto \theta$, entonces $[\![\widehat{\sigma}]\!] \Rightarrow [\![\widehat{\sigma}']\!]$ y $[\![\widehat{\theta}]\!] \Rightarrow [\![\widehat{\theta}']\!]$. Ahora, $[\![\widehat{\sigma} \cup \widehat{\theta}]\!] \Leftrightarrow [\![\widehat{\sigma}]\!] \wedge [\![\widehat{\theta}]\!] \Rightarrow [\![\widehat{\sigma}']\!] \wedge [\![\widehat{\theta}']\!] \Leftrightarrow [\![\widehat{\sigma}' \cup \widehat{\theta}']\!]$. Si $[\![\widehat{\sigma} \cup \widehat{\theta}]\!] \Rightarrow [\![\widehat{\sigma}' \cup \widehat{\theta}']\!]$ entonces $\widehat{\sigma}' \cup \widehat{\theta}' \propto \widehat{\sigma} \cup \widehat{\theta}$ y, por el Lema 4.2.12, $\sigma' \Uparrow_{\mathcal{A}} \theta' \equiv mgu_{\mathcal{A}}(\widehat{\sigma}' \cup \widehat{\theta}') \propto mgu(\widehat{\sigma} \cup \widehat{\theta}) \equiv \sigma \Uparrow \theta$.
□

Definimos, a continuación, el sistema de transición abstracto que formaliza la relación de *narrowing* aproximado. El cálculo abstracto se define sobre estados abstractos de $State_{\mathcal{A}}$, y realiza las computaciones con respecto a un programa abstracto de $Srtc_{\mathcal{A}}$. El cálculo es equivalente al introducido en la Definición 3.2.3, reemplazando los dominios y operadores concretos por sus versiones abstractas.

Definición 4.2.14 (narrowing con estrategia abstracto $\hookrightarrow_{\varphi}$) *Dado un programa abstracto $\mathcal{R}_{\mathcal{A}} \in Srtc_{\mathcal{A}}$, definimos la relación de narrowing (condicional) abstracto $\hookrightarrow_{\varphi}$ como la menor relación que satisface:*

$$\frac{\begin{array}{l} u \in \varphi(g) \;\wedge\; r \equiv (\lambda \to \rho \Leftarrow C) \ll \mathcal{R}_{\mathcal{A}} \;\wedge\; \delta = mgu_{\mathcal{A}}(\{g_{|u} = \lambda\}) \\ \wedge\; \delta_r = restrict_{\varphi}(\delta, r) \;\wedge\; g' = (C, g[\rho]_u)\delta_r \end{array}}{\langle g, \kappa \rangle \;\hookrightarrow_{\varphi}\; \langle g', \kappa \Uparrow_{\mathcal{A}} \delta \rangle}$$

Al igual que en el caso concreto, la relación de *narrowing* abstracto se define de manera genérica respecto a una estrategia φ. Denotamos por $\mathcal{R}^{+}_{\mathcal{A}}$ a la correspondiente extensión del programa abstracto $\mathcal{R}_{\mathcal{A}}$ con las reglas para tratar la igualdad sintáctica. Definimos, a continuación, la semántica operacional abstracta para el conjunto de éxitos.

Definición 4.2.15 (conjunto de éxitos abstracto $\Delta^{\varphi}_{\mathcal{R}_{\mathcal{A}}}$) *Dado un programa abstracto $\mathcal{R}_{\mathcal{A}} \in Srtc_{\mathcal{A}}$ y un objetivo abstracto $g \in Goal_{\mathcal{A}}$, formulamos la semántica operacional (o conjunto de éxitos) abstracta de g con respecto a $\mathcal{R}_{\mathcal{A}}$, usando la estrategia φ, como sigue:*

$$\Delta^{\varphi}_{\mathcal{R}_{\mathcal{A}}}(g) = \{\kappa\lceil_{Var(g)} \mid \langle g, \epsilon\rangle \hookrightarrow^{*}_{\varphi} \langle \top, \kappa\rangle\},$$

en la que las derivaciones $\langle g, \epsilon\rangle \hookrightarrow^{*}_{\varphi} \langle \top, \kappa\rangle$ se realizan usando las reglas del programa extendido $\mathcal{R}^{+}_{\mathcal{A}}$.

La relación del sistema de transición abstracto debe aproximar correctamente a la relación del sistema de transición concreto. La siguiente definición formaliza este concepto.

Definición 4.2.16 *Sea $\rightsquigarrow_{\varphi} \subseteq State \times State$ una relación de narrowing y $\hookrightarrow_{\varphi} \subseteq State_{\mathcal{A}} \times State_{\mathcal{A}}$ un sistema de transición abstracto. Decimos que $\hookrightarrow_{\varphi}$ aproxima a $\rightsquigarrow_{\varphi}$ (y lo denotamos por $\hookrightarrow_{\varphi} \propto \rightsquigarrow_{\varphi}$) si, dados $s, t \in State$ tal que $s \rightsquigarrow_{\varphi} t$ entonces, si existe un estado abstracto $s' \in State_{\mathcal{A}}$ tal que $s' \propto s$, se cumple:*

$$(\exists t' \in State_{\mathcal{A}}.\ s' \hookrightarrow_{\varphi} t' \ \wedge\ t' \propto t) \ \vee\ s' \propto t.$$

Intuitivamente, dado un estado concreto $s \in State$ y un estado abstracto $s' \in State_{\mathcal{A}}$ tal que $s' \propto s$, si puede probarse una transición concreta $s \rightsquigarrow_{\varphi} t$, entonces pueden ocurrir dos cosas: a) el *redex* seleccionado en la reducción concreta ha sido abstraído en el estado abstracto s' y, por tanto, se cumple que $s' \propto t$; b) el *redex* seleccionado existe en el estado abstracto, y entonces la relación abstracta debe probar la transición $s' \hookrightarrow_{\varphi} t'$ de forma que $t' \propto t$.

Para demostrar la corrección de la relación de *narrowing* abstracto, es necesario restringir de nuevo las posibles estrategias de *narrowing* a aquéllas que sean independientes del entorno. El siguiente lema establece que, si una relación de *narrowing* concreto $\rightsquigarrow_{\varphi}$ es independiente del entorno, entonces su versión abstracta $\hookrightarrow_{\varphi}$ también lo es.

Lema 4.2.17 *Sea φ una estrategia de narrowing tal que la relación $\rightsquigarrow_{\varphi}$ es independiente del entorno. Entonces, la relación abstracta $\hookrightarrow_{\varphi}$ también es independiente del entorno.*

DEMOSTRACIÓN. El resultado es inmediato por la Definición 3.2.5 y la Definición 4.2.14, ya que la única diferencia entre la relación concreta $\rightsquigarrow_{\varphi}$ y la relación abstracta $\hookrightarrow_{\varphi}$ consiste en el tratamiento de los símbolos $\perp$ en los términos abstractos, lo cual no afecta a las condiciones de la Definición 3.2.5. $\square$

En el resto del trabajo, cuando indicamos que la estrategia φ es independiente del entorno, nos referimos tanto a la relación concreta $\rightsquigarrow_{\varphi}$ como a la relación abstracta

$\hookrightarrow_\varphi$. A continuación demostramos que, para tales estrategias, si $\mathcal{R}_\mathcal{A} \propto \mathcal{R}$ entonces $\hookrightarrow_\varphi$ es una aproximación correcta de $\rightsquigarrow_\varphi$.

Lema 4.2.18 *Sea φ una estrategia independiente del entorno. Si $\mathcal{R}_\mathcal{A} \propto \mathcal{R}$, entonces $\hookrightarrow_\varphi \propto \rightsquigarrow_\varphi$.*

Demostración. Consideremos una transición de *narrowing* concreto:

$$\langle g, \sigma\rangle \rightsquigarrow_\varphi \langle (C\theta_r, g[\rho\theta_r]_u), \sigma \Uparrow \theta\rangle.$$

Por la definición de *narrowing*, existe una ocurrencia $u \in \varphi(g)$, una regla $r \equiv (\lambda \to \rho \Leftarrow C) \ll \mathcal{R}$, y una sustitución $\theta = mgu(\{g_{|u} = \lambda\})$ tal que $\sigma \Uparrow \theta \not\equiv fail$. La sustitución $\theta_r = restrict_\varphi(\theta, r)$ sólo se aplica a ρ y a C, ya que φ es independiente del entorno. Consideremos un estado $\langle g', \kappa\rangle \propto \langle g, \sigma\rangle$. Por la Definición 4.1.4, $g' \propto g$ y $\kappa \propto \sigma$. Ahora, puede ocurrir una de las dos siguientes posibilidades:

- $u \notin \varphi(g')$. En este caso, vamos a demostrar que el estado $\langle g', \kappa\rangle$ aproxima él mismo al estado $\langle (C\theta_r, g[\rho\theta_r]_u), \sigma \Uparrow \theta\rangle$. De acuerdo con la Definición 4.2.1, son dos los casos a considerar: 1) existe una ocurrencia $w \in \bar{O}(g')$ tal que $w \leq u$ y $g'_{|w} = \perp$ (el subtérmino seleccionado está aproximado por $\perp$ en g'), o 2) el subtérmino $g_{|u}$ pertenece a una ecuación e tal que $\not\exists e' \in g'.\ e' \preceq e$ (ninguna ecuación del estado abstracto aproxima a e). Vamos a demostrar que, en ambos casos, se cumple $\langle g', \kappa\rangle \propto \langle (C\theta_r, g[\rho\theta_r]_u), \sigma \Uparrow \theta\rangle$.

  En el caso 1), dado que $g' \propto g$, entonces se cumple que $\forall e' \in g'.\ \exists e \in g.\ e' \preceq e$. Por la Definición 4.2.1, $g' \propto g[\rho\theta_r]_u$ (ya que $g'_{|w} = \perp$ y $w \leq u$). Por tanto, $\forall e' \in g'.\ \exists e \in g[\rho\theta_r]_u.\ e' \preceq e$ y, trivialmente, $\forall e' \in g'.\ \exists e \in (C\theta_r, g[\rho\theta_r]_u).\ e' \preceq e$, con lo que tenemos $g' \propto (C\theta_r, g[\rho\theta_r]_u)$. Respecto a la parte entorno, si $\kappa \propto \sigma$ entonces, por la Definición 4.2.5 y el Lema 4.2.13, se cumple que $\kappa \equiv (\kappa \Uparrow_\mathcal{A} \epsilon) \propto (\sigma \Uparrow \theta)$ (ya que $\kappa \propto \sigma$ y $\epsilon \propto \theta$).

  En el caso 2), dado que $g_{|u}$ pertenece a una ecuación e tal que $\not\exists e' \in g'.\ e' \preceq e$, se sigue de forma inmediata que $\forall e' \in g'.\ \exists e \in (C\theta_r, g[\rho\theta_r]_u).\ e' \preceq e$ y, por tanto, $g' \propto (C\theta_r, g[\rho\theta_r]_u)$. Respecto a la parte entorno, el argumento de la prueba es análogo al del caso 1).

  En ambos casos, se cumple $g' \propto (C\theta_r, g[\rho\theta_r]_u)$ y $\kappa \propto (\sigma \Uparrow \theta)$. Por tanto, por la Definición 4.1.4, $\langle g', \kappa\rangle \propto \langle (C\theta_r, g[\rho\theta_r]_u), \sigma \Uparrow \theta\rangle$.

- $u \in \varphi(g')$. Supongamos que el subtérmino $g_{|u}$ pertenece a la ecuación $e \in g$, y que existe una ecuación $e' \in g'$ tal que $e' \propto e$ (en caso contrario, la prueba sería análoga a la del punto anterior). Ya que $g' \propto g$ y $e' \propto e$, por la Definición 4.2.1, se cumple que $g'_{|u} \propto g_{|u}$ y, por tanto, $(g'_{|u} = \lambda) \propto (g_{|u} = \lambda)$. Por el Lema 4.2.12, se cumple que $\theta' \equiv mgu_\mathcal{A}(\{g'_{|u} = \lambda\}) \propto mgu(\{g_{|u} = \lambda\}) \equiv \theta$. Ya que $\kappa \propto \sigma$, por el Lema 4.2.13, se sigue que $(\kappa \Uparrow_\mathcal{A} \theta') \propto (\sigma \Uparrow \theta)$.

Dado que $\theta \not\equiv fail$, entonces $\theta' \not\equiv fail$. Luego existe una regla $r' \equiv (\lambda \to \rho' \Leftarrow C') \ll \mathcal{R}_{\mathcal{A}}$[1] tal que $\rho' \propto \rho$ y $C' \propto C$ (ya que $\mathcal{R}_{\mathcal{A}} \propto \mathcal{R}$). Por tanto, se puede probar el paso de *narrowing* abstracto $\langle g', \kappa \rangle \hookrightarrow_{\varphi} \langle (C'\theta'_r, g'[\rho'\theta'_r]_u), \kappa \Uparrow_{\mathcal{A}} \theta' \rangle$, donde $\theta'_r = restrict_{\varphi}(\theta', r')$. Ya que $\theta' \propto \theta$, se cumple de manera trivial que $\theta'_r \propto \theta_r$. Dado que $Dom(\theta'_r) \subseteq Var(\lambda)$ y $Dom(\theta_r) \subseteq Var(\lambda)$, entonces $x\theta'_r \propto x\theta_r$, para toda variable $x \in Dom(\theta_r)$. En este caso, puesto que $C' \propto C$ y $\rho' \propto \rho$, entonces $C'\theta'_r \propto C\theta_r$ y $\rho'\theta'_r \propto \rho\theta_r$. Por último, si $C'\theta'_r \propto C\theta_r$ y $\rho'\theta'_r \propto \rho\theta_r$, entonces $(C'\theta'_r, g'[\rho'\theta'_r]_u) \propto (C\theta_r, g[\rho\theta_r]_u)$, lo que concluye la demostración.

□

Una vez demostrado que la relación de *narrowing* abstracto aproxima de manera correcta a la relación de *narrowing* concreto, podemos probar fácilmente la corrección parcial del análisis. Es decir, si éste termina, entonces lo hace computando una aproximación para la semántica de respuestas computadas concretas.

Teorema 4.2.19 (corrección parcial del análisis) *Sea $\mathcal{R}$ un programa y $\mathcal{R}_{\mathcal{A}}$ un programa abstracto tal que $\mathcal{R}_{\mathcal{A}} \propto \mathcal{R}$. Dado un objetivo $g \in Goal$, entonces para toda sustitución $\theta \in \mathcal{O}^{\varphi}_{\mathcal{R}}(g)$ existe una sustitución abstracta $\kappa \in \Delta^{\varphi}_{\mathcal{R}_{\mathcal{A}}}(g)$ tal que $\kappa \propto \theta$.*

Demostración. Vamos a probar el siguiente resultado más general: para todo estado s_0 tal que existe la derivación concreta:

$$s_0 \leadsto_{\varphi} \ldots \leadsto_{\varphi} s_n,$$

y para todo estado abstracto s'_0 tal que $s'_0 \propto s_0$, se cumple que existe una derivación abstracta:

$$s'_0 \hookrightarrow_{\varphi} \ldots \hookrightarrow_{\varphi} s'_m$$

con $s'_m \propto s_n$, $m \leq n$. Demostramos el teorema por inducción sobre la longitud n de la derivación concreta.

Sea $n = 1$. El teorema se sigue directamente del Lema 4.2.18 y del hecho que $\mathcal{R}_{\mathcal{A}} \propto \mathcal{R}$.

Consideremos el caso inductivo $n > 1$. Ya que $s_0 \leadsto_{\varphi} \ldots \leadsto_{\varphi} s_{n-1}$, por la hipótesis de inducción, existe una derivación abstracta $s'_0 \hookrightarrow_{\varphi} \ldots \hookrightarrow_{\varphi} s'_k$, $k \leq n-1$, tal que $s'_k \propto s_{n-1}$. Ya que $s'_k \propto s_{n-1}$ y $s_{n-1} \leadsto_{\varphi} s_n$ entonces, por el Lema 4.2.18, 1) $s'_k \hookrightarrow_{\varphi} s'_{k+1}$ y $s'_{k+1} \propto s_n$, o bien 2) $s'_k \propto s_n$. Consideremos primero el caso 1). En este caso, basta con tomar $m = k+1$ y entonces $s'_m \propto s_n$, $m \leq n$. Ahora consideramos el caso 2). En este caso, tomamos $m = k$ y, por tanto, $s'_m \propto s_n$, $m \leq n-1 \leq n$.

[1] Por simplicidad, asumimos aquí que los renombramientos elegidos para las variables de las reglas de $\mathcal{R}_{\mathcal{A}}$, son los mismos que para las reglas de $\mathcal{R}$.

Finalmente, el teorema se sigue del resultado anterior, tomando $s_0 \equiv s'_0 \equiv \langle g, \epsilon \rangle$ y $s_n \equiv \langle \top, \theta \rangle$ (i.e. $\theta \in \mathcal{O}^{\varphi}_{\mathcal{R}}(g)$), ya que si $s'_m \equiv \langle g', \kappa \rangle \propto \langle \top, \theta \rangle \equiv s_n$ entonces $\kappa \propto \theta$ y, por el Lema 4.2.4, $g' \equiv \top$, con lo que $\kappa \in \Delta^{\varphi}_{\mathcal{R}_{\mathcal{A}}}(g)$. $\square$

El teorema anterior únicamente garantiza la corrección parcial del análisis, es decir, si la relación de *narrowing* abstracto termina computando un conjunto de respuestas abstractas para el objetivo g, entonces se cumple que cada respuesta computada concreta está aproximada por una de las respuestas abstractas. Sin embargo, esto no es suficiente para definir un análisis útil. Es necesario, además, que los análisis terminen, i.e. las posibles derivaciones abstractas deben ser todas finitas. En la siguiente sección tratamos con detalle este tema.

4.3 Terminación del Análisis

En esta sección, pasamos a considerar el problema de garantizar que los análisis sean finitos. Esencialmente, dado un programa concreto $\mathcal{R}$, la terminación del análisis se asegura construyendo un programa aproximado $\mathcal{R}_{\mathcal{A}}$ que cumpla las dos condiciones siguientes:

- el programa abstracto es una aproximación correcta del programa concreto, i.e. $\mathcal{R}_{\mathcal{A}} \propto \mathcal{R}$ (*corrección*); y
- toda derivación de *narrowing* abstracto usando las reglas de $\mathcal{R}_{\mathcal{A}}$ termina en un número finito de pasos (*terminación*).

En el siguiente apartado, se define un método general de abstracción de programas que, preservando la noción de aproximación, garantiza la terminación de las derivaciones aproximadas.

4.3.1 Abstracción de Programas

En la literatura se proponen dos tipos de grafos para estudiar la terminación de un programa (ver, por ejemplo, [Bol93]):

- *grafos de detección de bucles*: siempre que una derivación tiene asociado un bucle en el grafo, dicha derivación es infinita;
- *grafos de prevención de bucles*: toda derivación infinita tiene un bucle asociado en el grafo.

En ambos casos, el grafo se construye a partir del programa considerado y, eventualmente, del propio objetivo a resolver. La información del grafo se puede usar de forma dinámica –en tiempo de ejecución– o de forma estática –en tiempo de compilación–.

El primer tipo de grafos sólo ayuda a eliminar algunas derivaciones infinitas, pero existe el riesgo de seguir explorando un espacio de búsqueda infinito (algunas derivaciones infinitas podrían no ser detectadas). Obviamente, este tipo de grafos no es interesante con respecto al análisis, ya que la terminación del mismo debe estar garantizada en cualquier caso. El segundo tipo, los grafos de prevención de bucles, pueden provocar la eliminación innecesaria de derivaciones finitas (con la correspondiente pérdida de precisión) pero, en cambio, se garantiza que todas las derivaciones infinitas son eliminadas. Este segundo tipo sí resulta útil para garantizar la terminación del análisis.

Nuestra noción de programa abstracto es paramétrica con repecto a un grafo de prevención de bucles estático, i.e. un grafo finito de dependencias entre términos que se construye a partir del programa y es independiente del objetivo. Este grafo permite reconocer aquellas derivaciones que con toda seguridad terminan. Nuestro propósito es usarlo para transformar un espacio de búsqueda posiblemente infinito en uno finito (aproximado). Con la información del grafo, generamos una versión compilada del programa original (programa abstracto), que aún produce un conjunto completo de respuestas aproximadas pero que, en ningún caso, da lugar a una derivación infinita. La siguiente definición formaliza nuestra noción de grafo de prevención de bucles.

Definición 4.3.1 (grafo de prevención de bucles) *Dado un SRTC $\mathcal{R}$, un* grafo de prevención de bucles *es una relación $\mathcal{G}_\mathcal{R}$ consistente en un conjunto finito de pares de términos, tales que:*

1. *El cierre transitivo $\mathcal{G}^+_\mathcal{R}$ es decidible.*
2. *Existe una función (posiblemente parcial) que asigna a un término $t \in \tau(\Sigma \cup V)$ un nodo $\mathring{t}$ del grafo $\mathcal{G}_\mathcal{R}$ tal que, si existe una derivación infinita de la forma:*

   $$\langle g_0, \theta_0 \rangle \leadsto_\varphi \langle g_1, \theta_1 \rangle \leadsto_\varphi \dots$$

   entonces $\exists i \geq 0.\ \langle \mathring{t}_i, \mathring{t}_i \rangle \in \mathcal{G}^+_\mathcal{R}$, donde $t_i = g_{i|u}\theta_i$ y $u \in \bar{O}(g_i)$.

En adelante diremos que $\mathcal{G}_\mathcal{R}$ tiene un "bucle" si $\langle \mathring{t}_i, \mathring{t}_i \rangle \in \mathcal{G}^+_\mathcal{R}$.

Informalmente, el grafo de prevención de bucles definido garantiza que, si existe alguna derivación de *narrowing* infinita, entonces alguno de los *redexes* seleccionados en dicha derivación tiene asociado un término en el grafo que está contenido en un bucle. Un grafo de prevención de bucles se puede ver como un tipo de "oráculo", cuya utilidad para demostrar la terminación de las derivaciones de *narrowing* se establece en el siguiente lema.

Lema 4.3.2 *Sea $\mathcal{R}$ un SRTC y $\mathcal{G}_\mathcal{R}$ un grafo de prevención de bucles para $\mathcal{R}$. Si $\mathcal{G}_\mathcal{R}$ no contiene bucles, entonces toda derivación de narrowing para $\mathcal{R}$ termina.*

DEMOSTRACIÓN. Inmediata a partir de la Definición 4.3.1 de grafo de prevención de bucles. □

Veamos un ejemplo que ilustra el concepto de grafo de prevención de bucles.

Ejemplo 10 *Sea $\mathcal{R}$ el programa:*

$$\begin{array}{rcl} x+0 & \to & x \\ x+s(y) & \to & s(x+y) \end{array}$$

y consideremos la siguiente definición para la función (parcial) $\mathring{t}$ que asocia al término $t \in \tau(\Sigma \cup V)$ un nodo t' en el grafo:

$$\mathring{t}= t' \quad si \quad mgu(\{t = t'\}) \not\equiv fail$$

(si hay varios nodos que unifican con t, se selecciona uno cualquiera y, si no hay ninguno, la función no está definida para t). Las variables que aparecen en los nodos del grafo se consideran implícitamente renombradas, de manera que no haya conflicto de variables al intentar la unificación. En este caso, el grafo definido por:

$$\mathcal{G}_{\mathcal{R}} = \{\langle x+s(y), x+s(y)\rangle\}$$

es un grafo de prevención de bucles, ya que cualquier derivación infinita, usando $\mathcal{R}$, contiene necesariamente algún término que unifique con $x+s(y)$.

A continuación, se describe un método para obtener un programa abstracto (compilado) a partir de la información recogida por un grafo de prevención de bucles. El método no depende del objetivo a resolver (sólo del programa), y permite obtener un programa abstracto para el cual todas las derivaciones de *narrowing* aproximado terminan. Más aún, independientemente del grafo de prevención de bucles usado, la semántica de un objetivo puede ser aproximada correctamente usando las reglas del programa abstracto. Informalmente, un programa se abstrae simplificando las partes derechas de las cabezas de las reglas, así como las ecuaciones en el cuerpo de las mismas. La definición es inductiva sobre la estructura de los términos y ecuaciones. La idea principal consiste en que, aquellos términos cuyo nodo asociado en el grafo posee un bucle, son eliminados del programa y sustituidos por $\perp$.

Definición 4.3.3 (programa abstracto) *Sea $\mathcal{R}$ un SRTC y $\mathcal{G}_{\mathcal{R}}$ un grafo de prevención de bucles para $\mathcal{R}$. Definimos la abstracción de $\mathcal{R}$ usando $\mathcal{G}_{\mathcal{R}}$ como sigue:*

$$\mathcal{A}(\mathcal{R}, \mathcal{G}_{\mathcal{R}}) = \{\lambda \to sh(\rho) \Leftarrow sh(C) \mid (\lambda \to \rho \Leftarrow C) \in \mathcal{R}\}$$

donde la función $sh(o)$ para un objeto sintáctico o se define inductivamente de la forma:

$$sh(o) = \begin{cases} o & si\ o \in V \\ f(sh(t_1), \ldots, sh(t_k)) & si\ o \equiv f(t_1, \ldots, t_k)\ y\ \langle \mathring{o}, \mathring{o} \rangle \notin \mathcal{G}^+_{\mathcal{R}} \\ sh(l) = sh(r) & si\ o \equiv (l = r) \\ sh(e_1), \ldots, sh(e_n) & si\ o \equiv e_1, \ldots, e_n \\ \perp & en\ otro\ caso. \end{cases}$$

El siguiente ejemplo ilustra la definición.

Ejemplo 11 *(Continúa del Ejemplo 10) La abstracción de $\mathcal{R}$, usando el grafo de prevención de bucles $\mathcal{G}_{\mathcal{R}}$, es el siguiente programa abstracto $\mathcal{R}_{\mathcal{A}} = \mathcal{A}(\mathcal{R}, \mathcal{G}_{\mathcal{R}})$:*

$$\begin{array}{rcl} x + 0 & \to & x \\ x + s(y) & \to & s(\perp) \end{array}$$

en el que la parte derecha de la segunda regla ha sido drásticamente simplificada introduciendo un $\perp$.

El siguiente lema formaliza el resultado de terminación para las derivaciones abstractas.

Lema 4.3.4 *Sea $\mathcal{R}$ un programa y $\mathcal{G}_{\mathcal{R}}$ un grafo de prevención de bucles asociado a $\mathcal{R}$. Sea $\mathcal{R}_{\mathcal{A}} = \mathcal{A}(\mathcal{R}, \mathcal{G}_{\mathcal{R}})$. Entonces, toda derivación abstracta $s_0 \hookrightarrow_{\varphi} s_1 \hookrightarrow_{\varphi} \ldots$ usando las reglas del programa $\mathcal{R}_{\mathcal{A}}$ termina en un número finito de pasos.*

Demostración. Vamos a probar que el número de pasos de una derivación de la forma:

$$\mathcal{D} \equiv (\langle g'_0, \theta'_0 \rangle \hookrightarrow_{\varphi} \langle g'_1, \theta'_1 \rangle \hookrightarrow_{\varphi} \ldots \hookrightarrow_{\varphi} \langle g'_n, \theta'_n \rangle \hookrightarrow_{\varphi} \ldots)$$

usando las reglas del programa $\mathcal{R}_{\mathcal{A}}$ es finito.

Realizamos la demostración por reducción al absurdo. Supongamos, pues, que la anterior derivación $\mathcal{D}$ es infinita. Ahora, asociamos a cada estado abstracto $\langle g'_i, \theta'_i \rangle$ de la derivación $\mathcal{D}$ un estado concreto $\langle g_i, \theta_i \rangle$ tal que $[\![\langle g'_i, \theta'_i \rangle]\!] = \exists y_1 \ldots \exists y_k.\ \langle g_i, \theta_i \rangle$, $0 \leq i \leq n$. Definimos de forma análoga $[\![r']\!] = r$, donde la tupla de ocurrencias del símbolo $\perp$ en r' ha sido sustituida en r por variables nuevas distintas. Así, definimos el SRTC $\bar{\mathcal{R}}_{\mathcal{A}}$ de la forma $\bar{\mathcal{R}}_{\mathcal{A}} = \{[\![r_{\mathcal{A}}]\!] \mid r_{\mathcal{A}} \in \mathcal{R}_{\mathcal{A}}\}$. Por construcción de $\bar{\mathcal{R}}_{\mathcal{A}}$, es obvio que podemos obtener un grafo de prevención de bucles asociado que no contiene bucles, simplemente eliminando del grafo $\mathcal{G}_{\mathcal{R}}$ del programa concreto aquellos bucles correspondientes a los términos simplificados a $\perp$ en $\mathcal{R}_{\mathcal{A}}$. Entonces, resulta inmediato que para toda derivación de la forma:

$$\langle g'_0, \theta'_0 \rangle \hookrightarrow_{\varphi} \langle g'_1, \theta'_1 \rangle \hookrightarrow_{\varphi} \ldots \hookrightarrow_{\varphi} \langle g'_n, \theta'_n \rangle \hookrightarrow_{\varphi} \ldots$$

usando las reglas del programa abstracto $\mathcal{R}_{\mathcal{A}}$, existe una derivación concreta, de igual longitud, de la forma:

$$\langle g_0, \theta_0 \rangle \rightsquigarrow_{\varphi} \langle g_1, \theta_1 \rangle \rightsquigarrow_{\varphi} \ldots \rightsquigarrow_{\varphi} \langle g_n, \theta_n \rangle \rightsquigarrow_{\varphi} \ldots$$

usando las reglas del programa $\bar{\mathcal{R}}_{\mathcal{A}}$, con lo que, aplicando el resultado del Lema 4.3.2, se obtiene la contradicción requerida. $\square$

Terminamos este apartado demostrando la corrección total del análisis. Para ello, necesitamos primero los siguientes lemas técnicos.

Lema 4.3.5 *Para todo término t, se cumple $sh(t) \propto t$.*

Demostración. El lema es inmediato por la definición de la función sh y la Definición 4.2.6, ya que la estructura del término $sh(t)$ es idéntica a la del término t, sustituyendo (posiblemente) algunos de sus subtérminos por $\perp$. $\square$

Lema 4.3.6 *Sea $\mathcal{R}$ un SRTC y $\mathcal{G}_{\mathcal{R}}$ un grafo de prevención de bucles asociado a $\mathcal{R}$. Dado el programa abstracto $\mathcal{R}_{\mathcal{A}} = \mathcal{A}(\mathcal{R}, \mathcal{G}_{\mathcal{R}})$, se cumple $\mathcal{R}_{\mathcal{A}} \propto \mathcal{R}$.*

Demostración. El resultado se sigue de manera inmediata a partir del Lema 4.3.5 y la Definición 4.2.8. $\square$

Ahora, podemos reforzar el resultado de corrección del análisis de la sección anterior como sigue.

Teorema 4.3.7 (corrección total del análisis) *Sea $\mathcal{R}$ un programa y $\mathcal{G}_{\mathcal{R}}$ un grafo de prevención de bucles asociado a $\mathcal{R}$. Sea $\mathcal{R}_{\mathcal{A}} = \mathcal{A}(\mathcal{R}, \mathcal{G}_{\mathcal{R}})$. Dado un objetivo g, se cumplen los dos siguientes resultados:*

1. *el conjunto de éxitos abstracto $\Delta^{\varphi}_{\mathcal{R}_{\mathcal{A}}}(g)$ es decidible; y*
2. *para toda sustitución $\theta \in \mathcal{O}^{\varphi}_{\mathcal{R}}(g)$ existe una sustitución abstracta $\kappa \in \Delta^{\varphi}_{\mathcal{R}_{\mathcal{A}}}(g)$ tal que $\kappa \propto \theta$.*

Demostración. El punto 1) se deriva directamente del Lema 4.3.4, ya que todas las posibles derivaciones abstractas con el programa $\mathcal{R}_{\mathcal{A}}$ son finitas. El punto 2) se sigue del Lema 4.3.6 (que garantiza que $\mathcal{R}_{\mathcal{A}} \propto \mathcal{R}$) y del Teorema 4.2.19 que establece la corrección parcial del análisis. $\square$

El análisis definido en los puntos anteriores es paramétrico con respecto a una estrategia de *narrowing* φ (independiente del entorno) y un grafo de prevención de bucles $\mathcal{G}_{\mathcal{R}}$. Los resultados obtenidos en esta sección son independientes de ambos parámetros, garantizándose la corrección y terminación del análisis para cualquier instancia del esquema.

En el siguiente apartado introducimos un método particular para construir, de forma automática, un posible grafo de prevención de bucles asociado al programa.

4.3.2 Grafo de Prevención de Bucles

Muchos de los trabajos sobre detección y prevención de bucles consideran la aplicación de grafos de detección de bucles en tiempo de ejecución. Los grafos de prevención de bucles estáticos, sin embargo, no han sido estudiados con tanta atención.

En el caso de los programas funcionales, existen varias aproximaciones que encontramos relacionadas. [CR91, RKKL85] consideran un grafo de términos que permite la detección de algunos bucles en el árbol de búsqueda que no contribuyen a ninguna solución. Los grafos se construyen usando la información tanto de las ecuaciones a ser reducidas, como de las reglas de programa usadas por *narrowing*.

En el caso de los programas lógicos, la noción estándar de grafo de dependencias entre los símbolos de predicado de un programa lógico [Llo87] se puede ver también como un tipo de grafo de detección de bucles. Una extensión de estos grafos a un dominio de "llamadas" es la noción de grafo-U (grafo de Unificación, *U-graph* [WS90]), el cual tiene en cuenta además la unificabilidad entre los átomos del programa.

A continuación mostramos una posible instancia de la Definición 4.3.1, siguiendo una idea comparable a la de los grafos-U de los programas lógicos (convenientemente extendida para tratar el anidamiento de funciones). Primero, necesitamos la siguiente definición auxiliar $\lfloor t \rfloor$, que reemplaza inductivamente cualquier subtérmino de t, cuyo símbolo de función más externo no sea constructor, por una variable nueva.

Definición 4.3.8 *Dado un término t, la función $\lfloor t \rfloor$ se define como sigue:*

$$\lfloor t \rfloor = \begin{cases} c(\lfloor t_1 \rfloor, \ldots, \lfloor t_k \rfloor) & \textit{si } t = c(t_1, \ldots, t_k) \textit{ y } c \in \mathcal{C} \\ y & \textit{en otro caso, donde } y \textit{ es una variable nueva.} \end{cases}$$

La siguiente definición formaliza un caso particular de grafo de dependencias entre términos $\mathcal{I}_\mathcal{R}$, "inducido" por un SRTC $\mathcal{R}$. Posteriormente, demostramos que $\mathcal{I}_\mathcal{R}$ es un grafo de prevención de bucles (i.e. cumple las condiciones de la Definición 4.3.1). Otro posible grafo de prevención de bucles diferente (pero menos preciso) se puede encontrar en [AFM95].

Definición 4.3.9 (grafo de dependencias $\mathcal{I}_\mathcal{R}$) *Sea $\mathcal{R}$ un SRTC. La siguiente transformación define un grafo dirigido $\mathcal{I}_\mathcal{R}$ de dependencias entre términos inducido por $\mathcal{R}$. Definimos la función auxiliar: $\bar{t} = f(\lfloor t_1 \rfloor, \ldots, \lfloor t_n \rfloor)$ si $t = f(t_1, \ldots, t_n)$. Para construir $\mathcal{I}_\mathcal{R}$, el algoritmo comienza con un estado $\langle \mathcal{R}, \emptyset \rangle$ y aplica las siguientes reglas de transición mientras éstas añadan nuevos arcos[2]:*

$$(1) \quad \frac{r \equiv (\lambda \to \rho \Leftarrow C) \ll \mathcal{R}}{\begin{array}{lll} \langle \mathcal{R}, \mathcal{I}_\mathcal{R} \rangle & \longmapsto & \langle \mathcal{R} - \{r\}, \mathcal{I}_\mathcal{R} \cup \{\lambda \stackrel{\mathcal{R}}{\to} \bar{t} \mid (t = \rho_{|u} \ \wedge \ u \in \bar{O}(\rho)) \ \vee \\ & & (t = e_{|u} \ \wedge \ e \in C \ \wedge \ u \in (\bar{O}(e) - \{\Lambda\}))\} \rangle \end{array}}$$

[2] Los nodos (términos) del grafo se consideran módulo renombramiento de variables.

$$(2) \quad \frac{(\lambda \xrightarrow{\mathcal{R}} \rho) \in \mathcal{I}_{\mathcal{R}} \ \wedge \ (\lambda' \xrightarrow{\mathcal{R}} \rho') \in \mathcal{I}_{\mathcal{R}} \ \wedge \ \rho \stackrel{?}{=} \lambda'}{\langle \mathcal{R}, \mathcal{I}_{\mathcal{R}} \rangle \ \longmapsto \ \langle \mathcal{R}, \mathcal{I}_{\mathcal{R}} \cup \{\rho \xrightarrow{u} \lambda'\} \rangle}$$

Dado un término $t \in \tau(\Sigma \cup V)$, definimos la función $\mathring{t}$ que asocia a t un nodo t' en el grafo, como sigue: $\mathring{t}= t'$ si 1) t' aparece en la parte izquierda de un arco $\xrightarrow{\mathcal{R}}$, y 2) t' unifica con $\bar{t}$. En el caso de que haya más de un término que cumpla las dos condiciones anteriores, se selecciona aquél que posea un camino en el grafo $\mathcal{I}_{\mathcal{R}}$ de mayor longitud (si no hay ninguno que cumpla las condiciones, la función está indefinida para t).

La terminación de este cálculo está asegurada, ya que el número de términos que aparecen en las reglas de $\mathcal{R}$ es finito. Informalmente, el algoritmo procede como sigue. Para cada regla $(\lambda \to \rho \Leftarrow C)$ de $\mathcal{R}$ y para cada término $f(t_1, \ldots, t_n)$ que aparezca en ρ o en C, la regla (1) añade un arco $\lambda \xrightarrow{\mathcal{R}} f(\lfloor t_1 \rfloor, \ldots, \lfloor t_n \rfloor)$ a $\mathcal{I}_{\mathcal{R}}$. La regla (2) añade un arco $\rho \xrightarrow{u} \lambda'$ entre la parte derecha ρ de un arco $\lambda \xrightarrow{\mathcal{R}} \rho$ de $\mathcal{I}_{\mathcal{R}}$ y la parte izquierda λ' de cada regla $\lambda' \xrightarrow{\mathcal{R}} \rho'$ con la que ρ unifica sintácticamente. En lo que sigue, $\to^*$ denota un camino en el grafo formado indistintamente por arcos $\xrightarrow{\mathcal{R}}$ ó $\xrightarrow{u}$. Por supuesto, teniendo en cuenta una estrategia φ particular, sería posible definir instancias más precisas del grafo, ya que no todos los subtérminos de las partes derechas de las cabezas y de las condiciones de las reglas pueden ser explotados por una determinada estrategia.

El grafo de términos $\mathcal{I}_{\mathcal{R}}$ es equivalente al grafo de símbolos más externos definido en [AFM95], cuando la función $\bar{t}$ se define como sigue: $\bar{t} = f$ si $t \equiv f(t_1, \ldots, t_n)$ (la función $\mathring{t}$ se define de idéntica forma). En general, $\mathcal{I}_{\mathcal{R}}$ permite representaciones más precisas de la estructura recursiva de un programa. Nuestra noción de grafo de prevención de bucles es similar a la de los grafos-U [WS90] de la programación lógica pura, en cuanto que éstos contienen un nodo por cada átomo (cada llamada) del programa y usan dos tipos de arcos: arcos de cláusulas y arcos de unificación. Existe un arco de cláusula de un átomo H a un átomo B_i si existe una cláusula en el programa de la forma $H \leftarrow B_1, \ldots, B_i, \ldots, B_n$, $n \geq 1$. Existe un arco de unificación entre un átomo B y un átomo H si B es un átomo del cuerpo de una cláusula, y unifica (módulo renombramiento) con la cabeza H de otra (o la misma) cláusula.

La siguiente proposición muestra que el grafo $\mathcal{I}_{\mathcal{R}}$ de dependencias entre términos inducido es, de hecho, un grafo de prevención de bucles.

Proposición 4.3.10 *Sea $\mathcal{R}$ un SRTC e $\mathcal{I}_{\mathcal{R}}$ el grafo de dependencias entre términos inducido por $\mathcal{R}$. Entonces, $\mathcal{I}_{\mathcal{R}}$ es un grafo de prevención de bucles para $\mathcal{R}$.*

DEMOSTRACIÓN. La demostración se basa en una medida de complejidad bien fundada sobre objetivos ecuacionales, que decrece a medida que se dan pasos de *narrowing*. Consideremos la derivación infinita:

$$\mathcal{D} \equiv \langle g_0, \theta_0 \rangle \leadsto_\varphi \langle g_1, \theta_1 \rangle \leadsto_\varphi \ldots$$

y asumamos, por reducción al absurdo, que en $\mathcal{I}_\mathcal{R}$ no existen bucles asociados a ninguno de los términos que aparecen en la derivación $\mathcal{D}$. Definimos $m(t)$ como la longitud del camino en el grafo $\mathcal{I}_\mathcal{R}$ que parte del nodo $\mathring{t}$ (si la función está indefinida, $m(t) = 0$). Asociamos a cada estado $\langle g_i, \theta_i \rangle$ en la derivación $\mathcal{D}$ un conjunto:

$$\mathcal{H}_i = \{\langle u, n \rangle \mid u \in \varphi(g_i),\ n = m(g_{i|u}\theta_i)\}$$

con $i \geq 0$. Definimos la complejidad $\mathcal{M}_i$ del objetivo $\langle g_i, \theta_i \rangle$ como el multiconjunto (finito) de números naturales formado por los segundos componentes de los pares de $\mathcal{H}_i$. Ya que $\mathcal{I}_\mathcal{R}$ no posee bucles asociados (usando la función $\mathring{t}$) a ningún término de $\mathcal{D}$, estos segundos componentes son siempre finitos para todo $i \geq 0$.

Definimos ahora un orden total bien fundado $<_{mul}$ sobre los multiconjuntos de complejidades, extendiendo de manera estándar el orden bien fundado $<$ sobre $\mathbb{N}$ al conjunto $M(\mathbb{N})$ de multiconjuntos finitos sobre $\mathbb{N}$. El conjunto $M(\mathbb{N})$ está bien fundado bajo el orden $<_{mul}$ ya que el conjunto $\mathbb{N}$ está bien fundado bajo el orden $<$ [DJ90, Klo92]. Sean $\mathcal{M}, \mathcal{M}'$ multiconjuntos de complejidades, $\mathcal{M} <_{mul} \mathcal{M}' \Leftrightarrow \exists X \subseteq \mathcal{M}, \exists X' \subseteq \mathcal{M}'$ tales que $\mathcal{M} = (\mathcal{M}' - X') \cup X$ y $\forall n \in X.\ \exists n' \in X'.\ n < n'$.

Por la definición de *narrowing*, en cada paso de derivación $\langle g_i, \theta_i \rangle \leadsto_\varphi \langle g_{i+1}, \theta_{i+1} \rangle$, la ocurrencia $u \in \varphi(g_i)$ seleccionada puede venir de las ocurrencias del objetivo inicial $\varphi(g_0)$, o de la parte derecha de la cabeza o de la condición de una regla del programa usada en algún paso previo. Por la definición de $\mathcal{I}_\mathcal{R}$, para todo $i \geq 0$, $\forall \langle u', n' \rangle \in (\mathcal{H}_{i+1} - \mathcal{H}_i)$ existe un arco $l \xrightarrow{\mathcal{R}} r \in \mathcal{R}_\mathcal{I}$ tal que $l \stackrel{?}{=} (g_{i|u}\theta_i)$ y $r \stackrel{?}{=} \overline{g_{i+1|u'}}$. Ya que existe un arco $r \xrightarrow{u} \lambda$, para todo λ tal que $r \stackrel{?}{=} \lambda$, entonces $m(g_{i|u}\theta_i) = 2 + m(g_{i+1|u'}\theta_{i+1})$ y, por tanto, $\forall \langle u', n' \rangle \in (\mathcal{H}_{i+1} - \mathcal{H}_i).\ n' < m(g_{i|u}\theta_i)$. Ahora, por la definición del orden $<_{mul}$, es inmediato que para todo $i \geq 0$, $\mathcal{M}_{i+1} <_{mul} \mathcal{M}_i$. Por la definición de *narrowing*, si existe un $k > 0$ tal que cada elemento de $\mathcal{M}_k$ es 0, entonces no existen más pasos de *narrowing* posibles. Por último, por el hecho de que el orden sobre multiconjuntos $<_{mul}$ sobre $M(\mathbb{N})$ está bien fundado, sólo es posible tener cadenas decrecientes y, por tanto, la derivación de *narrowing* termina. $\square$

El siguiente ejemplo ilustra el método de aproximación de un programa presentado en la Definición 4.3.3, usando como grafo de prevención de bucles el grafo de dependencias entre términos inducido por el programa.

Ejemplo 12 *Consideremos el siguiente programa $\mathcal{R}$:*

$$\begin{array}{rcl} h(0) & \rightarrow & 0 \\ f(0) & \rightarrow & 0 \\ f(c(x)) & \rightarrow & c(f(x)) \Leftarrow g(x) = x \\ g(c(x)) & \rightarrow & c(x) \end{array}$$

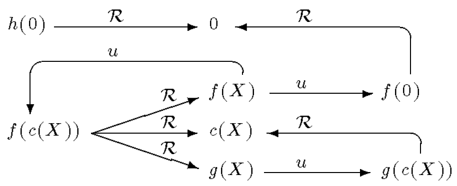

Figura 4.1: Grafo de dependencias entre términos $\mathcal{I}_{\mathcal{R}}$.

El grafo de dependencias entre términos $\mathcal{I}_{\mathcal{R}}$ inducido por $\mathcal{R}$ se muestra en la Figura 4.1. Existen solo dos bucles en el grafo, concretamente,

$$\langle f(x), f(x)\rangle, \langle f(c(x)), f(c(x))\rangle \in \mathcal{I}_{\mathcal{R}}^{+}.$$

Usando esta información, el programa abstracto $\mathcal{R}_{\mathcal{A}} = \mathcal{A}(\mathcal{R}, \mathcal{I}_{\mathcal{R}})$ es:

$$\begin{array}{rcl} h(0) & \rightarrow & 0 \\ f(0) & \rightarrow & 0 \\ f(c(x)) & \rightarrow & c(\perp) \Leftarrow g(x) = x \\ g(c(x)) & \rightarrow & c(x) \end{array}$$

donde el único término "simplificado" a $\perp$ ha sido $f(x)$.

Por el Teorema 4.3.7, dado un objetivo, existe una reducción por *narrowing* abstracto para cada derivación de *narrowing* concreto. Esta correspondencia se puede ver fácilmente en el siguiente ejemplo.

Ejemplo 13 *Consideremos de nuevo los programas $\mathcal{R}$ y $\mathcal{R}_{\mathcal{A}}$ del Ejemplo 12. Las figuras 4.2 y 4.3 muestran, respectivamente, el espacio de búsqueda de narrowing condicional básico y de narrowing básico abstracto para el objetivo inicial $\langle h(f(z)) = 0, \epsilon\rangle$. El conjunto de éxitos concreto $\mathcal{O}_{\mathcal{R}}^{\blacksquare}(g)$ coincide con el conjunto de éxitos abstracto $\Delta_{\mathcal{R}}^{\blacksquare}(g) = \{\{z/0\}\}$ (pese a que hemos pasado de un espacio de búsqueda infinito a uno finito).*

4.4 Análisis Composicional

En esta sección demostramos, bajo las mismas condiciones que en el caso concreto, la composicionalidad de la relación de *narrowing* abstracto. La utilidad de esta propiedad sobre la relación abstracta radica, al igual que en el caso concreto, en la posibilidad de definir un cálculo paralelo que sea equivalente al cálculo secuencial. Esto nos permite definir un análisis composicional equivalente al análisis definido en la Sección 4.2. En el capítulo siguiente mostraremos que la composicionalidad de la relación abstracta permite también incorporar incrementalidad en el análisis.

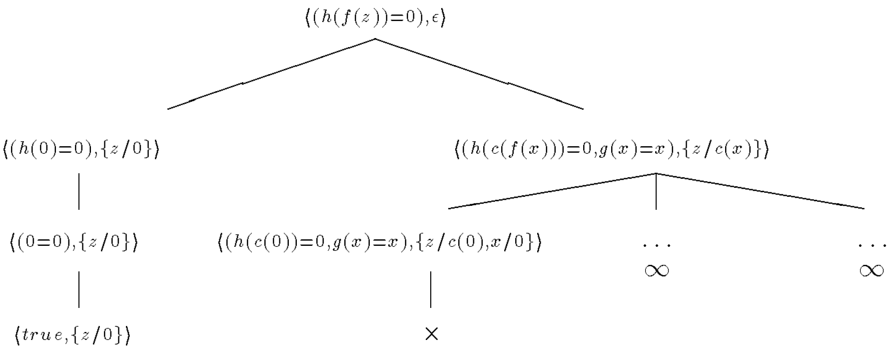

Figura 4.2: Narrowing Básico.

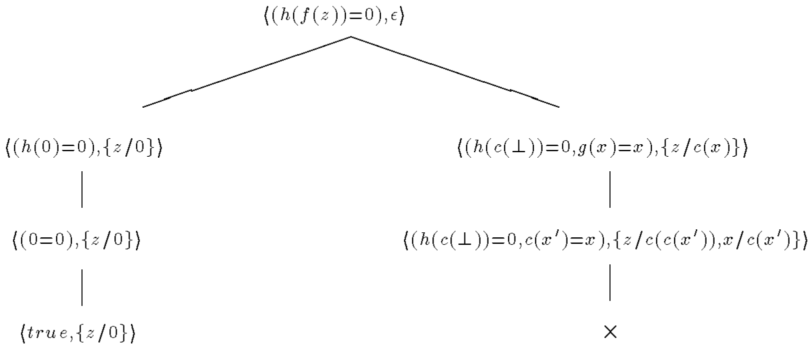

Figura 4.3: Narrowing Básico Abstracto.

Usando un argumento similar al de la Sección 3.2.1, la composicionalidad es una consecuencia del siguiente teorema.

Teorema 4.4.1 *Sea φ una estrategia de narrowing independiente del entorno. Dados dos objetivos ecuacionales $g_1, g_2 \in Goal_{\mathcal{A}}$, se cumple que:*

$$\begin{array}{l} \langle (g_1, g_2), \sigma\rangle \hookrightarrow_{\varphi}^{n} \langle g', \sigma \Uparrow_{\mathcal{A}} \theta\rangle \quad sii \\ \qquad \langle g_1, \sigma\rangle \hookrightarrow_{\varphi}^{n_1} \langle g_1', \sigma \Uparrow_{\mathcal{A}} \theta_1\rangle \ \ y \ \ \langle g_2, \sigma\rangle \hookrightarrow_{\varphi}^{n_2} \langle g_2', \sigma \Uparrow_{\mathcal{A}} \theta_2\rangle, \\ \qquad \theta = \theta_1 \Uparrow \theta_2 \not\equiv fail, \quad n = n_1 + n_2 \ \ y \ \ g' = (g_1', g_2'). \end{array}$$

Demostración. La demostración es perfectamente análoga a la del Teorema 3.2.6, teniendo en cuenta que:

- Si la relación concreta $\leadsto_{\varphi}$ es independiente del entorno, su versión abstracta $\hookrightarrow_{\varphi}$ también lo es (por el Lema 4.2.17).
- La relación abstracta $\hookrightarrow_{\varphi}$ es una aproximación correcta de la relación $\leadsto_{\varphi}^{\Uparrow}$ (ver la Definición 4.2.14).

□

Establecida la composicionalidad de la relación abstracta, mostramos los resultados de composicionalidad para la semántica abstracta del conjunto de éxitos $\Delta^{\varphi}_{\mathcal{R}_{\mathcal{A}}}$. La versión abstracta del Corolario 3.2.10 se puede formular como sigue.

Corolario 4.4.2 *Sea $\mathcal{R}$ un programa, $\mathcal{R}_{\mathcal{A}} \propto \mathcal{R}$ un programa abstracto que aproxima a $\mathcal{R}$ y φ una estrategia de narrowing independiente del entorno. Entonces, la semántica operacional abstracta $\Delta^{\varphi}_{\mathcal{R}_{\mathcal{A}}}$ es composicional.*

DEMOSTRACIÓN. Si la relación concreta $\leadsto_{\varphi}$ es independiente del entorno, por el Lema 4.2.17, la relación abstracta $\hookrightarrow_{\varphi}$ también es independiente del entorno. Ahora, el corolario se sigue como una consecuencia inmediata del Teorema 4.4.1, para $\sigma \equiv \epsilon$ y $g' \equiv g'_1 \equiv g'_2 \equiv \top$. □

El Corolario 4.4.2 implica que, cuando la estrategia de *narrowing* coincide con las estrategias básica, "innermost" o perezosa, el conjunto de éxitos abstractos se puede construir de forma composicional. Concretamente, las semánticas abstractas $\Delta^{\blacksquare}_{\mathcal{R}_{\mathcal{A}}}$, $\Delta^{\blacktriangleleft}_{\mathcal{R}_{\mathcal{A}}}$ y $\Delta^{\blacktriangleright}_{\mathcal{R}_{\mathcal{A}}}$ son composicionales.

El Teorema 4.4.1 permite definir un cálculo abstracto composicional, en el estilo del cálculo presentado en la Definición 3.2.14, o alguna otra variante en la línea de lo observado al final del capítulo anterior que deje mayor flexibilidad en la reconciliación de las derivaciones paralelas, como el cálculo que formalizamos en la siguiente definición.

Definición 4.4.3 *Sea $\mathcal{R}_{\mathcal{A}}$ un programa abstracto y φ una estrategia de narrowing. Definimos la relación de narrowing composicional abstracto como un sistema de transición $(State_{\mathcal{A}}, \rightarrowtail_{\varphi})$ cuya relación de transición $\rightarrowtail_{\varphi} \subseteq State_{\mathcal{A}} \times State_{\mathcal{A}}$ se define como la menor relación que satisface[3,4]:*

$$(1)\quad \frac{\begin{array}{l}\langle g_1, \kappa\rangle \rightarrowtail^{*}_{\varphi} \langle \top, \kappa \Uparrow_{\mathcal{A}} \kappa_1\rangle \;\wedge\; \langle g_2, \kappa\rangle \rightarrowtail^{*}_{\varphi} \langle \top, \kappa \Uparrow_{\mathcal{A}} \kappa_2\rangle \;\wedge \\ \kappa' = (\kappa_1 \Uparrow_{\mathcal{A}} \kappa_2) \;\wedge\; \kappa' \not\equiv fail\end{array}}{\langle (g_1, g_2), \kappa\rangle \rightarrowtail^{\shortparallel}_{\varphi} \langle \top, \kappa \Uparrow_{\mathcal{A}} \kappa'\rangle}$$

$$(2)\quad \frac{\langle e, \kappa\rangle \hookrightarrow_{\varphi} \langle g', \kappa'\rangle}{\langle e, \kappa\rangle \rightarrowtail_{\varphi} \langle g', \kappa'\rangle}$$

(3) seleccionar "don't care" (3.1) ó (3.2):

$$(3.1)\quad \frac{\langle g, \kappa\rangle \hookrightarrow_{\varphi} \langle g', \kappa'\rangle}{\langle g, \kappa\rangle \rightarrowtail_{\varphi} \langle g', \kappa'\rangle} \qquad (3.2)\quad \frac{\langle g, \kappa\rangle \rightarrowtail^{\shortparallel}_{\varphi} \langle g', \kappa'\rangle}{\langle g, \kappa\rangle \rightarrowtail_{\varphi} \langle g', \kappa'\rangle}$$

[3] Suponemos que en las reglas (3.1) y (3.2) g no está formada por una única ecuación, y que en la regla (1) g_1 y g_2 denotan secuencias de ecuaciones no vacías.

[4] Una implementación de la regla (3) podría seleccionar siempre la regla (3.1), o bien la regla (3.2), o bien cualquiera de ellas (pero no la otra) en cada paso.

Informalmente, el cálculo definido considera básicamente dos casos, dependiendo de si el objetivo g contiene una sola ecuación o más. En el primer caso (regla 2), la relación de *narrowing* composicional abstracto se comporta simplemente como la relación secuencial abstracta ($\hookrightarrow_{\varphi}$). En el caso de que g contenga más de una ecuación (regla 3), se puede reducir usando la relación de *narrowing* secuencial abstracto (regla 3.1), o bien se puede dividir en dos subobjetivos g_1 y g_2 que se resuelven independientemente (regla 3.2 y regla 1). Las siguientes proposiciones demuestran la corrección y completitud del nuevo cálculo composicional abstracto (para estrategias independientes del entorno).

Proposición 4.4.4 (corrección $\rightarrowtail_{\varphi}$)
Si $\langle g, \kappa\rangle \rightarrowtail_{\varphi}^{*} \langle \top, \kappa'\rangle$ *entonces* $\langle g, \kappa\rangle \hookrightarrow_{\varphi}^{*} \langle \top, \kappa'\rangle$.

Demostración. Procedemos con la prueba del siguiente resultado:

$$\text{si } \langle g, \kappa\rangle \rightarrowtail_{\varphi}^{\parallel} \langle \top, \kappa \Uparrow_{\mathcal{A}} \theta\rangle \text{ entonces } \langle g, \kappa\rangle \hookrightarrow_{\varphi}^{*} \langle \top, \kappa \Uparrow_{\mathcal{A}} \theta\rangle,$$

ya que la proposición se sigue de manera directa a partir de éste.
Si $\langle g, \kappa\rangle \rightarrowtail_{\varphi}^{\parallel} \langle \top, \kappa \Uparrow_{\mathcal{A}} \theta\rangle$ entonces, por la regla (1) de la Definición 4.4.3, existen dos subobjetivos g_1, g_2 tales que:

$$\begin{array}{l} \langle g_1, \kappa\rangle \rightarrowtail_{\varphi}^{*} \langle \top, \kappa \Uparrow_{\mathcal{A}} \theta_1\rangle \\ \langle g_2, \kappa\rangle \rightarrowtail_{\varphi}^{*} \langle \top, \kappa \Uparrow_{\mathcal{A}} \theta_2\rangle \end{array}$$

donde $g = (g_1, g_2)$ y $\theta = \theta_1 \Uparrow_{\mathcal{A}} \theta_2 \not\equiv fail$. Demostramos el resultado por inducción sobre el número n de aplicaciones recursivas de la regla (1) en las dos derivaciones anteriores.

$n = 0$. El resultado es inmediato.

Sea $n > 0$. Ya que $\langle g_1, \kappa\rangle \rightarrowtail_{\varphi}^{*} \langle \top, \kappa \Uparrow_{\mathcal{A}} \theta_1\rangle$ y $\langle g_2, \kappa\rangle \rightarrowtail_{\varphi}^{*} \langle \top, \kappa \Uparrow_{\mathcal{A}} \theta_2\rangle$ entonces, por las reglas (1) y (3) de la Definición 4.4.3, puede ocurrir uno de los siguientes casos:

$$\begin{cases} \langle g_1, \kappa\rangle \hookrightarrow_{\varphi}^{*} \langle g_1', \kappa \Uparrow_{\mathcal{A}} \kappa_1\rangle \rightarrowtail_{\varphi}^{\parallel} \langle \top, \kappa \Uparrow_{\mathcal{A}} \kappa_1 \Uparrow_{\mathcal{A}} \delta_1\rangle \equiv \langle \top, \kappa \Uparrow_{\mathcal{A}} \theta_1\rangle, & \text{y} \\ \langle g_2, \kappa\rangle \hookrightarrow_{\varphi}^{*} \langle g_2', \kappa \Uparrow_{\mathcal{A}} \kappa_2\rangle \rightarrowtail_{\varphi}^{\parallel} \langle \top, \kappa \Uparrow_{\mathcal{A}} \kappa_2 \Uparrow_{\mathcal{A}} \delta_2\rangle \equiv \langle \top, \kappa \Uparrow_{\mathcal{A}} \theta_2\rangle, & \text{o bien} \end{cases}$$
$$\begin{cases} \langle g_1, \kappa\rangle \hookrightarrow_{\varphi}^{*} \langle g_1', \kappa \Uparrow_{\mathcal{A}} \kappa_1\rangle \rightarrowtail_{\varphi}^{\parallel} \langle \top, \kappa \Uparrow_{\mathcal{A}} \kappa_1 \Uparrow_{\mathcal{A}} \delta_1\rangle \equiv \langle \top, \kappa \Uparrow_{\mathcal{A}} \theta_1\rangle, & \text{y} \\ \langle g_2, \kappa\rangle \hookrightarrow_{\varphi}^{*} \langle \top, \kappa \Uparrow_{\mathcal{A}} \theta_2\rangle, & \text{o bien} \end{cases}$$
$$\begin{cases} \langle g_1, \kappa\rangle \hookrightarrow_{\varphi}^{*} \langle \top, \kappa \Uparrow_{\mathcal{A}} \theta_1\rangle, & \text{y} \\ \langle g_2, \kappa\rangle \hookrightarrow_{\varphi}^{*} \langle g_2', \kappa \Uparrow_{\mathcal{A}} \kappa_2\rangle \rightarrowtail_{\varphi}^{\parallel} \langle \top, \kappa \Uparrow_{\mathcal{A}} \kappa_2 \Uparrow_{\mathcal{A}} \delta_2\rangle \equiv \langle \top, \kappa \Uparrow_{\mathcal{A}} \theta_2\rangle, & \end{cases}$$

y el número de aplicaciones de la regla (1) en las derivaciones etiquetadas con la relación $\rightarrowtail_{\varphi}^{\parallel}$ es menor estricto que n. Consideramos sólo el primer caso, ya que los otros dos son perfectamente análogos. Ya que $\langle g_1', \kappa \Uparrow_{\mathcal{A}} \kappa_1\rangle \rightarrowtail_{\varphi}^{\parallel} \langle \top, \kappa \Uparrow_{\mathcal{A}} \theta_1\rangle$ y

$\langle g_2', \kappa \Uparrow_{\mathcal{A}} \kappa_2 \rangle \rightarrowtail^{\parallel}_{\varphi} \langle \top, \kappa \Uparrow_{\mathcal{A}} \theta_2 \rangle$ entonces, por la hipótesis de inducción, existen las derivaciones:

$$\langle g_1', \kappa \Uparrow_{\mathcal{A}} \kappa_1 \rangle \hookrightarrow^{*}_{\varphi} \langle \top, \kappa \Uparrow_{\mathcal{A}} \theta_1 \rangle$$
$$\langle g_2', \kappa \Uparrow_{\mathcal{A}} \kappa_2 \rangle \hookrightarrow^{*}_{\varphi} \langle \top, \kappa \Uparrow_{\mathcal{A}} \theta_2 \rangle.$$

Ya que $\theta_1 \Uparrow_{\mathcal{A}} \theta_2 \not\equiv \mathit{fail}$ entonces, por el Teorema 4.4.1, existe la derivación:

$$\langle g, \kappa \rangle \equiv \langle (g_1, g_2), \kappa \rangle \hookrightarrow^{*}_{\varphi} \langle \top, \kappa \Uparrow_{\mathcal{A}} (\theta_1 \Uparrow_{\mathcal{A}} \theta_2) \rangle \equiv \langle \top, \kappa \Uparrow_{\mathcal{A}} \theta \rangle$$

de lo cual se sigue el resultado enunciado. □

Proposición 4.4.5 (completitud $\rightarrowtail_{\varphi}$)
Si $\langle g, \kappa \rangle \hookrightarrow^{*}_{\varphi} \langle \top, \kappa' \rangle$ entonces $\langle g, \kappa \rangle \rightarrowtail^{*}_{\varphi} \langle \top, \kappa' \rangle$.

Demostración. Probamos la proposición por inducción sobre la longitud n de la primera derivación.

Si $n = 1$ y $\langle g, \kappa \rangle \hookrightarrow_{\varphi} \langle \top, \theta \rangle$, entonces g debe estar formada por una única ecuación. Ahora, por la regla (2) de la Definición 4.4.3, demostramos el paso $\langle g, \kappa \rangle \rightarrowtail_{\varphi} \langle \top, \theta \rangle$.

Consideremos el caso inductivo. Si $n > 1$, entonces existe la derivación:

$$\langle g, \kappa \rangle \equiv \langle g_0, \kappa \Uparrow_{\mathcal{A}} \vartheta_0 \rangle \hookrightarrow_{\varphi} \langle g_1, \kappa \Uparrow_{\mathcal{A}} \vartheta_1 \rangle \hookrightarrow_{\varphi} \ldots \hookrightarrow_{\varphi} \langle g_n, \kappa \Uparrow_{\mathcal{A}} \vartheta_n \rangle \equiv \langle \top, \theta \rangle,$$

con ($\vartheta_0 \equiv \epsilon$). Podemos distinguir dos casos:

(1) Sea $g_0 \equiv e$. Por la hipótesis de inducción, existe la derivación $\langle g_1, \kappa \Uparrow_{\mathcal{A}} \vartheta_1 \rangle \rightarrowtail_{\varphi} \ldots \rightarrowtail_{\varphi} \langle \top, \theta \rangle$. Ya que $\langle e, \kappa \rangle \hookrightarrow_{\varphi} \langle g_1, \kappa \Uparrow_{\mathcal{A}} \vartheta_1 \rangle$ entonces, por la regla (2) de la Definición 4.4.3, existe $\langle e, \kappa \rangle \rightarrowtail_{\varphi} \langle g_1, \kappa \Uparrow_{\mathcal{A}} \vartheta_1 \rangle$.

(2) Sea $g_0 \equiv (g_0', g_0'')$, donde g_0' y g_0'' son secuencias de ecuaciones no vacías. Debemos probar que existe la derivación $\langle g, \kappa \rangle \rightarrowtail_{\varphi} \ldots \rightarrowtail_{\varphi} \langle \top, \theta \rangle$. De acuerdo a la regla (3) de la Definición 4.4.3, necesitamos demostrar que:

   (*a*) $\langle g, \kappa \rangle \equiv \langle g_0, \kappa \Uparrow_{\mathcal{A}} \vartheta_0 \rangle \hookrightarrow_{\varphi} \langle g_1, \kappa \Uparrow_{\mathcal{A}} \vartheta_1 \rangle \rightarrowtail_{\varphi} \ldots \rightarrowtail_{\varphi} \langle \top, \theta \rangle$, o bien

   (*b*) $\langle g, \kappa \rangle \rightarrowtail^{\parallel}_{\varphi} \langle \top, \theta \rangle$.

   Caso (*a*). Ya que $\langle g_1, \kappa \Uparrow_{\mathcal{A}} \vartheta_1 \rangle \hookrightarrow_{\varphi} \ldots \hookrightarrow_{\varphi} \langle \top, \theta \rangle$ entonces, por la hipótesis de inducción, existe la derivación $\langle g_1, \kappa \Uparrow_{\mathcal{A}} \vartheta_1 \rangle \rightarrowtail_{\varphi} \ldots \rightarrowtail_{\varphi} \langle \top, \theta \rangle$.

   Caso (*b*). Ya que $\langle g, \kappa \rangle \equiv \langle (g_0', g_0''), \kappa \rangle \hookrightarrow^{*}_{\varphi} \langle \top, \kappa \Uparrow_{\mathcal{A}} \vartheta_n \rangle \equiv \langle \top, \theta \rangle$ entonces, por el Teorema 4.4.1, existen las derivaciones:

$$\langle g_0', \kappa \rangle \hookrightarrow^{*}_{\varphi} \langle \top, \kappa \Uparrow_{\mathcal{A}} \theta_1 \rangle \;\; \text{y} \;\; \langle g_0'', \kappa \rangle \hookrightarrow^{*}_{\varphi} \langle \top, \kappa \Uparrow_{\mathcal{A}} \theta_2 \rangle$$

donde $\vartheta_n = \theta_1 \Uparrow_{\mathcal{A}} \theta_2 \not\equiv fail$. Entonces, por la hipótesis de inducción, existen las derivaciones:

$$\langle g_0', \kappa \rangle \rightarrowtail_\varphi^* \langle \top, \kappa \Uparrow_{\mathcal{A}} \theta_1 \rangle \;\; \text{y} \;\; \langle g_0'', \kappa \rangle \rightarrowtail_\varphi^* \langle \top, \kappa \Uparrow_{\mathcal{A}} \theta_2 \rangle.$$

Por tanto, por la regla (1) de la Definición 4.4.3,

$$\langle (g_0', g_0''), \kappa \rangle \rightarrowtail_\varphi^{\shortparallel} \langle \top, \kappa \Uparrow_{\mathcal{A}} (\theta_1 \Uparrow_{\mathcal{A}} \theta_2) \rangle \equiv \langle \top, \kappa \Uparrow_{\mathcal{A}} \vartheta_n \rangle \equiv \langle \top, \theta \rangle$$

lo que completa la prueba.

□

Usando estos resultados, la composicionalidad de la relación $\hookrightarrow_\varphi$ se puede extender directamente a la relación de *narrowing* composicional abstracto $\rightarrowtail_\varphi$. Introducimos, a continuación, el conjunto de éxitos abstracto generado mediante la relación $\rightarrowtail_\varphi$.

Definición 4.4.6 *Dado un programa abstracto $\mathcal{R}_{\mathcal{A}}$ y un objetivo abstracto $g \in Goal_{\mathcal{A}}$, denotamos el conjunto de éxitos composicional abstracto de g con respecto a $\mathcal{R}_{\mathcal{A}}$ usando la estrategia φ, como sigue:*

$$\nabla^{\varphi}_{\mathcal{R}_{\mathcal{A}}}(g) = \{\kappa_{\lceil Var(g)} \mid \langle g, \epsilon \rangle \rightarrowtail_\varphi^* \langle \top, \kappa \rangle\},$$

en la que las derivaciones $\langle g, \epsilon \rangle \rightarrowtail_\varphi^* \langle \top, \kappa \rangle$ se realizan usando las reglas del programa extendido $\mathcal{R}^+_{\mathcal{A}}$.

El siguiente resultado expresa formalmente la composicionalidad de la semántica abstracta $\nabla^{\varphi}_{\mathcal{R}_{\mathcal{A}}}$.

Corolario 4.4.7 *Sea $\mathcal{R}_{\mathcal{A}}$ un programa abstracto, g_1, g_2 objetivos abstractos y φ una estrategia de narrowing independiente del entorno. Entonces,*

$$\nabla^{\varphi}_{\mathcal{R}_{\mathcal{A}}}(g_1, g_2) = \nabla^{\varphi}_{\mathcal{R}_{\mathcal{A}}}(g_1) \Uparrow_{\mathcal{A}} \nabla^{\varphi}_{\mathcal{R}_{\mathcal{A}}}(g_2).$$

Demostración. Inmediato por el Corolario 4.4.2, la Proposición 4.4.4 y la Proposición 4.4.5. □

Como consecuencia de la Proposición 4.4.4 y la Proposición 4.4.5, el análisis para un objetivo dado (la semántica operacional abstracta del objetivo) se puede determinar explotando la composicionalidad AND de la relación de *narrowing* y su versión abstracta. En el siguiente capítulo mostramos cómo la composicionalidad de la semántica abstracta, tal y como se ha establecido en el Teorema 4.4.1, permite realizar los análisis de forma incremental.

4.5 Comparación con Otras Aproximaciones

En este capítulo hemos formalizado, en el marco de la interpretación abstracta, un método de análisis estático de programas lógico-funcionales que permite computar, de manera finita, una aproximación correcta de la semántica operacional del programa. El análisis definido es paramétrico respecto a 1) la estrategia de *narrowing* y 2) la técnica de abstracción de programas que construye los programas abstractos a partir de los concretos. Se demuestran la corrección y terminación del análisis para estrategias de *narrowing* independientes del entorno. Se introduce la noción genérica de "grafo de prevención de bucles", que da soporte a una técnica de transformación automática para obtener un programa abstracto que aproxima las derivaciones del programa concreto y que, además, permite asegurar la terminación del análisis. Por último, se formaliza un tipo particular de grafo de dependencias entre términos (inducido por un programa), y se demuestra que es una instancia de la definición genérica de grafo de prevención de bucles.

El análisis estático que hemos definido sigue las líneas del marco para el análisis de la insatisfacibilidad ecuacional presentado en [AFM95]. En [AFM95] se formalizan distintos análisis de insatisfaciblidad, tomando como semántica operacional concreta la relación de *narrowing* condicional básico. El marco para el análisis estático presentado en este trabajo es, por el contrario, paramétrico con respecto a distintas estrategias de *narrowing* y a un método correcto de abstracción del programa, lo que aumenta considerablemente su aplicabilidad y eficiencia. Más aún, los análisis de *narrowing* abstracto computan una descripción finita de la semántica operacional, lo que permite extraer distintos tipos de información a partir de ella (en el siguiente capítulo se presentan algunas instancias particulares del análisis).

Otras aproximaciones al análisis estático de programas lógico-funcionales son las siguientes. En [HW92] se presenta una extensión del marco para el análisis de programas lógicos, definido en [Bru91], al caso de programas lógico-funcionales. Esta aproximación ha sido utilizada, por ejemplo, en [HZ94], para desarrollar un análisis de la *groundness*, y en [SR92, SR93], para extraer información sobre el paralelismo implícito de los programas. Por otro lado, en [Boy93, BPM93] se define una aproximación al análisis basada en el uso de *grafos de dependencias*, siguiendo la aproximación formalizada en [DM85]. En [MKM+93] se esboza un intérprete abstracto, que permite transformar un programa en un conjunto de ecuaciones que reflejan correctamente la información sobre la demanda inducida por las reglas del programa. En [HL96], se define un análisis de la *groundness* a partir de una definición denotacional de *narrowing* perezoso, siguiendo la aproximación al análisis de programas lógicos presentada en [JS87].

Mucho más cercanos a nuestra técnica de análisis se encuentran los trabajos sobre *reescritura abstracta*, presentados en [BEØ93, BE95], en los que se define un método para aproximar conjuntos de formas normales para instancias básicas de términos con respecto a un programa CB. Se definen las nociones de descripción de un término y de programa abstracto, que se computa como el menor punto fijo de un operador de transformación de programas. Entonces, los términos son aproximados y normalizados usando las reglas del programa abstracto. Si los conjuntos de formas normales aproximadas no tienen elementos en común, los términos concretos no son $\mathcal{E}$-unificables.

El análisis basado en la reescritura abstracta de [BEØ93, BE95] es incomparable con el nuestro, en el sentido de que existen ejemplos para los que nuestro análisis permite obtener una descripción más precisa, y viceversa. Sin embargo, el análisis basado en la reescritura abstracta podría ser mejorado fácilmente si se reformulase como una instancia del marco presentado aquí. Haciendo uso de su técnica de abstracción de programas, pero extendiendo la relación de reescritura abstracta a una relación de *narrowing* abstracto, se eliminaría la necesidad de trabajar con términos básicos, lo cual aumentaría la precisión del análisis [AFV96a].

El análisis formalizado en este capítulo puede ser implementado de manera relativamente sencilla, ya que puede utilizarse el propio intérprete de *narrowing* condicional concreto para realizar las computaciones abstractas. El proceso más complejo consiste en la obtención del grafo de prevención de bucles asociado al programa, y en la posterior abstracción del programa en función de la información recogida por el grafo. El proceso completo se puede considerar como un proceso de compilación abstracta, y ha sido implementado en BIM-Prolog con buenos resultados (ver un ejemplo de su uso en la Sección 5.3). En el capítulo siguiente se desarrollan algunas de las principales aplicaciones del análisis definido, tales como:

- una estrategia que permite demostrar, en algunos casos, la insatisfacibilidad de un objetivo y, en general, eliminar derivaciones inútiles del árbol de búsqueda del *narrower* concreto;

- una serie de optimizaciones para un lenguaje lógico con restricciones ecuacionales, definido como instancia del esquema CLP [JL87]; y

- un análisis para los enlaces de las variables, que permite obtener información de *groundness* e independencia de forma simple y efectiva.

Capítulo 5

Aplicaciones del Análisis

En este capítulo, presentamos algunas de las posibles aplicaciones del análisis estático formalizado en el capítulo anterior. Los resultados de esta sección son, al igual que el análisis definido, paramétricos respecto a una estrategia de *narrowing* independiente del entorno.

En primer lugar, en la Sección 5.1, mostramos cómo el resultado del análisis (el conjunto de respuestas computadas abstractas para un programa y un objetivo dados) puede ser usado para extraer información útil acerca del programa. Concretamente, cuando el conjunto de respuestas abstractas es vacío, podemos afirmar que el objetivo original es insatisfacible. Por otro lado, cuando este conjunto no es vacío mostramos, en la Sección 5.2, que contiene información correcta y completa para determinar el tipo de sustituciones que pueden formar parte del conjunto de éxitos concreto. Este resultado nos permite formular un cálculo de *narrowing* refinado, en el que algunos estados pueden ser eliminados del árbol de búsqueda sin pérdida de completitud, concretamente, aquéllos en los que la sustitución en el entorno no es 'compatible' con alguna de las sustituciones del conjunto de éxitos abstracto.

En la Sección 5.3, empezamos recordando una instancia simple del marco de programación lógica con restricciones (CLP, *Constraint Logic Programming* [JL87]), en el que las restricciones son ecuaciones cuya satisfacibilidad debe comprobarse con respecto a una teoría ecuacional [AFL95]. Posteriormente, mostramos cómo los diferentes resultados obtenidos en los capítulos precedentes pueden combinarse para optimizar los mecanismos de comprobación de la satisfacibilidad de las restricciones ecuacionales en este marco. Ilustramos la técnica con los tiempos de ejecución obtenidos con una implementación preliminar del analizador. Por último, en la Sección 5.4, definimos un análisis simple para extraer información acerca de los enlaces de las variables en tiempo de ejecución.

5.1 Análisis de Insatisfacibilidad

La habilidad para detectar la insatisfacibilidad de un conjunto de ecuaciones (con respecto a una teoría de Horn ecuacional) resulta fundamental para eliminar derivaciones del árbol de búsqueda que no conducen a ninguna solución, evitando así computaciones innecesarias. El problema de la satisfacibilidad ecuacional es, en general, indecidible. Veamos cómo se puede utilizar la información del análisis introducido en el capítulo precedente para formular una aproximación decidible al problema.

Introducimos, en primer lugar, un subconjunto destacado de las respuestas parciales de un objetivo, denominado *respuestas parciales correctas*. Dicho conjunto está formado por aquellas sustituciones que aparecen en los estados intermedios de una derivación de éxito. La siguiente definición formaliza este concepto.

Definición 5.1.1 (conjunto de éxitos parciales) *Dado un objetivo g y un programa $\mathcal{R}$, definimos el conjunto de éxitos parciales $\mathcal{P}^{\varphi}_{\mathcal{R}}(g)$ como sigue:*

$$\mathcal{P}^{\varphi}_{\mathcal{R}}(g) = \{\sigma \mid \langle g, \epsilon\rangle \leadsto^*_{\varphi} \langle g', \sigma\rangle \leadsto^*_{\varphi} \langle \top, \theta\rangle\}.$$

Una propiedad importante del conjunto de éxitos parciales $\mathcal{P}^{\varphi}_{\mathcal{R}}(g)$, consiste en que toda respuesta computada del conjunto $\mathcal{O}^{\varphi}_{\mathcal{R}}(g)$ debe ser una instancia de alguna de las sustituciones de $\mathcal{P}^{\varphi}_{\mathcal{R}}(g)$. El resultado es inmediato ya que, a lo largo de una derivación (de éxito), la sustitución actual sólo puede instanciarse más. Dado que disponemos de una descripción de las respuestas computadas para el objetivo, es posible establecer una relación precisa entre las sustituciones (abstractas) del conjunto $\Delta^{\varphi}_{\mathcal{R}_{\mathcal{A}}}(g)$ y las sustituciones parciales (correctas) que pueden aparecer en una derivación concreta de éxito.

El siguiente teorema muestra que, cuando la relación de *narrowing* abstracto termina computando un conjunto vacío de soluciones, podemos establecer que el conjunto de ecuaciones inicial no tiene ningún $\mathcal{E}$-unificador. En el caso de que el conjunto de sustituciones abstractas no sea vacío, en la siguiente sección se muestra cómo se puede emplear para guiar las derivaciones de *narrowing* concreto.

Teorema 5.1.2 *Sea g un objetivo, $\mathcal{R}$ un programa y $\mathcal{R}_{\mathcal{A}} \propto \mathcal{R}$ un programa abstracto que aproxima a $\mathcal{R}$. Si $\Delta^{\varphi}_{\mathcal{R}_{\mathcal{A}}}(g) = \emptyset$, entonces g es insatisfacible en $\mathcal{R}$.*

DEMOSTRACIÓN. Por el Teorema 4.2.19, para toda sustitución concreta $\theta \in \mathcal{O}^{\varphi}_{\mathcal{R}}(g)$ debe existir una sustitución abstracta $\kappa \in \Delta^{\varphi}_{\mathcal{R}_{\mathcal{A}}}(g)$ tal que $\kappa \propto \theta$. Ya que $\Delta^{\varphi}_{\mathcal{R}_{\mathcal{A}}}(g) = \emptyset$, entonces $\mathcal{O}^{\varphi}_{\mathcal{R}}(g) = \emptyset$ y, por la completitud del *narrowing*, g es insatisfacible en $\mathcal{R}$. □

Usando este resultado, se pueden definir análisis de insatisfacibilidad ecuacional para las distintas estrategias de *narrowing*, cuya precisión dependerá de la forma en que se aproxime el programa original. Veamos un ejemplo que ilustra este resultado, usando la estrategia de *narrowing* condicional “innermost”.

Ejemplo 14 *Sea $\mathcal{R}$ el siguiente programa* (knapsack 0/1)*:*

$$\begin{array}{rcl} addweight(nil) & \rightarrow & 0 \\ addweight(x:xs) & \rightarrow & weight(x) + addweight(xs) \\ \\ weight(a) & \rightarrow & s(s(0)) \\ weight(b) & \rightarrow & s(0) \\ \\ x+0 & \rightarrow & x \\ x+s(y) & \rightarrow & s(x+y) \end{array}$$

El programa abstracto $\mathcal{R}_{\mathcal{A}}$:

$$\begin{array}{rcl} addweight(nil) & \rightarrow & 0 \\ addweight(x:xs) & \rightarrow & weight(x) + \bot \\ \\ weight(a) & \rightarrow & s(s(0)) \\ weight(b) & \rightarrow & s(0) \\ \\ x+0 & \rightarrow & x \\ x+s(y) & \rightarrow & s(\bot) \end{array}$$

es una aproximación correcta de $\mathcal{R}$, i.e. $\mathcal{R}_{\mathcal{A}} \propto \mathcal{R}$.
Dado el objetivo $g \equiv (addweight(x:xs) = 0)$, y considerando la relación de narrowing abstracto "innermost" $\hookrightarrow_{\blacktriangleleft}$, existen las siguientes derivaciones abstractas para g en el programa abstracto $\mathcal{R}_{\mathcal{A}}$:

$$\begin{array}{rcl} \langle \underline{addweight(x:xs)} = 0, \epsilon\rangle & \hookrightarrow_{\blacktriangleleft} & \langle \underline{weight(x)} + \bot = 0, \epsilon\rangle \\ & \hookrightarrow_{\blacktriangleleft} & \langle \underline{s(s(0)) + \bot} = 0, \{x/a\}\rangle \\ & \hookrightarrow_{\blacktriangleleft} & \langle s(s(0)) = 0, \{x/a\}\rangle \\ \langle \underline{addweight(x:xs)} = 0, \epsilon\rangle & \hookrightarrow_{\blacktriangleleft} & \langle \underline{weight(x)} + \bot = 0, \epsilon\rangle \\ & \hookrightarrow_{\blacktriangleleft} & \langle \underline{s(s(0)) + \bot} = 0, \{x/a\}\rangle \\ & \hookrightarrow_{\blacktriangleleft} & \langle s(\bot) = 0, \{x/a\}\rangle \\ \langle \underline{addweight(x:xs)} = 0, \epsilon\rangle & \hookrightarrow_{\blacktriangleleft} & \langle \underline{weight(x)} + \bot = 0, \epsilon\rangle \\ & \hookrightarrow_{\blacktriangleleft} & \langle \underline{s(0) + \bot} = 0, \{x/b\}\rangle \\ & \hookrightarrow_{\blacktriangleleft} & \langle s(0) = 0, \{x/b\}\rangle \\ \langle \underline{addweight(x:xs)} = 0, \epsilon\rangle & \hookrightarrow_{\blacktriangleleft} & \langle \underline{weight(x)} + \bot = 0, \epsilon\rangle \\ & \hookrightarrow_{\blacktriangleleft} & \langle \underline{s(0) + \bot} = 0, \{x/b\}\rangle \\ & \hookrightarrow_{\blacktriangleleft} & \langle s(\bot) = 0, \{x/b\}\rangle \end{array}$$

todas las cuales son derivaciones de fallo. Por tanto, el conjunto de respuestas computadas abstractas $\Delta^{\blacktriangleleft}_{\mathcal{R}_{\mathcal{A}}}(g)$ es vacío y, por el Corolario 5.1.2, el objetivo g es insatisfacible en el programa $\mathcal{R}$.

Además del anterior resultado de insatisfacibilidad, el Teorema 4.2.19 sugiere que el conjunto $\Delta^{\varphi}_{\mathcal{R}_{\mathcal{A}}}(g)$ también puede ser usado para detectar computaciones concretas inútiles, i.e. derivaciones que no llevan a ninguna solución. En base a estas ideas, en la Sección 5.2, se define un cálculo de *narrowing* refinado que comprueba en cada paso si la sustitución actual del estado es 'compatible' con alguna de las descripciones del conjunto de éxitos abstracto. En caso negativo, la derivación es desestimada, ya que no conduce a ninguna solución.

5.2 Narrowing Guiado por Narrowing Abstracto

Introducimos, en primer lugar, una relación de 'compatibilidad' $\diamond$ entre sustituciones abstractas. En lo que sigue, usaremos esta relación para realizar también comparaciones entre sustituciones abstractas y sustituciones concretas (ya que $Sub \subseteq Sub_{\mathcal{A}}$).

Definición 5.2.1 (relación de compatibilidad $\diamond$) *Sean $\kappa, \sigma \in Sub_{\mathcal{A}}$. Definimos la relación de compatibilidad entre sustituciones abstractas $\kappa \diamond \sigma$ como sigue:*

$$\kappa \diamond \sigma \;\Leftrightarrow\; \exists \theta \in Sub_{\mathcal{A}}.\, (\kappa \preceq \theta \;\wedge\; \sigma \preceq \theta).$$

Informalmente, dos sustituciones abstractas están en la relación $\diamond$ (i.e. son compatibles) si y sólo si tienen una instancia común, es decir, ambas son generalizaciones de una misma sustitución. Sin embargo, la Definición 5.2.1 no ofrece mucha intuición sobre cómo puede llevarse a cabo el test $\diamond$ sobre sustituciones abstractas. La siguiente proposición establece cómo puede realizarse dicho test en la práctica. Concretamente, la idea es tan simple como comprobar que la composición paralela (abstracta) de ambas sutituciones no sea $fail$.

Proposición 5.2.2 *Dadas dos sustituciones abstractas $\kappa, \sigma \in Sub_{\mathcal{A}}$, entonces:*

$$\kappa \diamond \sigma \;\Leftrightarrow\; (\kappa \Uparrow_{\mathcal{A}} \sigma) \not\equiv fail.$$

Demostración. El resultado se sigue de la siguiente cadena de equivalencias:

$$\begin{array}{lcl}
\kappa \diamond \sigma & \Leftrightarrow & \text{(por la Definición 5.2.1)} \\
\exists \theta.\, (\kappa \preceq \theta \;\wedge\; \sigma \preceq \theta) & \Leftrightarrow & \text{(por la Definición 4.2.5)} \\
\exists \theta.\, (\theta \;\Rightarrow\; [\![\kappa]\!] \;\wedge\; \theta \;\Rightarrow\; [\![\sigma]\!]) & \Leftrightarrow & \\
\exists \theta.\, (\theta \in unif([\![\kappa]\!]) \;\wedge\; \theta \in unif([\![\sigma]\!])) & \Leftrightarrow & \\
\exists \theta.\, (\theta \in unif([\![\kappa]\!]) \cap unif([\![\sigma]\!])) & \Leftrightarrow & \\
\exists \theta.\, \theta \in unif([\![\kappa \cup \sigma]\!]) & \Leftrightarrow & \text{(por la Proposición 4.2.10)} \\
mgu_{\mathcal{A}}(\kappa \cup \sigma) \not\equiv fail. & &
\end{array}$$

$\square$

Partiendo del hecho de que cada sustitución en una derivación de éxito concreta (i.e cada respuesta parcial correcta) es más general que alguna de las respuestas computadas, mostramos ahora cómo usar la información del análisis para mejorar el rendimiento del intérprete. Esta propiedad es auxiliar y resulta útil para plantear seguidamente las optimizaciones.

Teorema 5.2.3 *Sea g un objetivo, $\mathcal{R}$ un programa y $\mathcal{R}_{\mathcal{A}} \propto \mathcal{R}$ un programa abstracto que aproxima a $\mathcal{R}$. Entonces, para toda respuesta parcial correcta $\theta \in \mathcal{P}^{\varphi}_{\mathcal{R}}(g)$, existe una sustitución abstracta $\kappa \in \Delta^{\varphi}_{\mathcal{R}_{\mathcal{A}}}(g)$ tal que $\kappa \diamond \theta$.*

Demostración. Si $\theta \in \mathcal{P}^{\varphi}_{\mathcal{R}}(g)$, entonces existe una derivación:

$$\langle g, \epsilon\rangle \rightsquigarrow^*_{\varphi} \langle g', \theta\rangle \rightsquigarrow^*_{\varphi} \langle \top, \sigma\rangle$$

tal que $\sigma \in \mathcal{O}^{\varphi}_{\mathcal{R}}(g)$ y $\theta \leq \sigma$. Por la Definición 4.2.5, $\theta \preceq \sigma$ (considerando θ y σ sustituciones abstractas, ya que $Sub \subseteq Sub_{\mathcal{A}}$). Por el Teorema 4.2.19, debe existir una sustitución $\kappa \in \Delta^{\varphi}_{\mathcal{R}_{\mathcal{A}}}(g)$ tal que $\kappa \preceq \sigma$. Por tanto, para toda $\theta \in \mathcal{P}^{\varphi}_{\mathcal{R}}(g)$, existe una sustitución abstracta $\kappa \in \Delta^{\varphi}_{\mathcal{R}_{\mathcal{A}}}(g)$ tal que:

- $\theta \preceq \sigma$
- $\kappa \preceq \sigma$

y, en consecuencia, por la Definición 5.2.1, se cumple que $\kappa \diamond \theta$. $\square$

El siguiente resultado se sigue de forma inmediata del teorema anterior, y representa la base de nuestra optimización.

Corolario 5.2.4 *Sea g un objetivo, $\mathcal{R}$ un programa y $\mathcal{R}_{\mathcal{A}} \propto \mathcal{R}$ un programa abstracto que aproxima a $\mathcal{R}$. Sea $\langle g', \theta\rangle$ un estado en el árbol de derivaciones de narrowing para $\mathcal{R} \cup \{g\}$, usando la estrategia φ. Si no existe ninguna sustitución $\kappa \in \Delta^{\varphi}_{\mathcal{R}_{\mathcal{A}}}(g)$ tal que $\kappa \diamond \theta$, entonces no existe ninguna derivación de éxito a partir del estado $\langle g', \theta\rangle$.*

Demostración. La prueba se realiza por reducción al absurdo. Asumamos que existe una derivación de éxito partiendo de $\langle g', \theta\rangle$. Entonces, por el Teorema 5.2.3, existe una sustitución $\kappa \in \Delta^{\varphi}_{\mathcal{R}_{\mathcal{A}}}(g)$ tal que $\kappa \diamond \theta$, lo que contradice la hipótesis. $\square$

Ya que los subárboles que se originan en los estados que satisfacen las condiciones del Corolario 5.2.4 no contienen ninguna solución, es inútil explorarlos. Este resultado sugiere un método para reducir el tamaño del árbol de búsqueda manteniendo la completitud del cálculo. Concretamente, la idea es definir un cálculo de *narrowing* 'refinado' en el que se eliminen aquellos estados que satisfagan las condiciones del Corolario 5.2.4. Así, presentamos ahora una versión refinada de la relación de *narrowing* genérico, introducida en la Definición 2.3.1, que hace uso del conjunto de soluciones abstractas $\Delta^{\varphi}_{\mathcal{R}_{\mathcal{A}}}(g)$ obtenido mediante el análisis.

Definición 5.2.5 (narrowing con estrategia refinado $\looparrowright_\varphi$) *Sea g_0 un objetivo ecuacional, $\mathcal{R}$ un programa y $\mathcal{R}_\mathcal{A} \propto \mathcal{R}$ un programa abstracto que aproxima a $\mathcal{R}$. Definimos la relación de narrowing genérico refinado $\looparrowright_\varphi$ como la menor relación que satisface[1]:*

$$\frac{\begin{array}{l} u \in \varphi(g) \ \wedge \ r \equiv (\lambda \to \rho \Leftarrow C) \ll \mathcal{R} \ \wedge \ \sigma = mgu(\{g_{|u}\theta = \lambda\}) \\ \wedge \ \sigma_r = restrict_\varphi(\sigma, r) \ \wedge \ \exists \kappa \in \Delta^\varphi_{\mathcal{R}_\mathcal{A}}(g_0).\ \kappa \diamond \theta\sigma \end{array}}{\langle g, \theta\rangle \ \looparrowright_\varphi \ \langle (C, g[\rho]_u)\sigma_r, \theta\sigma\rangle}$$

En el cálculo, únicamente se realiza el análisis para el objetivo inicial g_0 y se poda el árbol de búsqueda de acuerdo con la información obtenida en el conjunto $\Delta^\varphi_{\mathcal{R}_\mathcal{A}}(g_0)$. Otra opción, podría consistir en calcular el conjunto $\Delta^\varphi_{\mathcal{R}_\mathcal{A}}(g)$ para los objetivos g de todos los estados $\langle g, \theta\rangle$ en el árbol concreto, y proseguir o no con la derivación en función del test. En este caso, las posibilidades de eliminar derivaciones inútiles es mucho mayor pero, por supuesto, el coste asociado resulta excesivo.

El siguiente ejemplo ilustra cómo se comporta el cálculo refinado de la Definición 5.2.5, cuando la estrategia φ se corresponde con la estrategia de *narrowing* básico.

Ejemplo 15 *Consideremos de nuevo los programa $\mathcal{R}$, $\mathcal{R}_\mathcal{A}$ y el objetivo inicial $g \equiv \langle h(f(z)) = 0, \epsilon\rangle$ del Ejemplo 13. Mientras que el espacio de búsqueda del narrowing básico para g es infinito, el cálculo refinado explora el árbol representado en la Figura 5.1. Nótese que el subárbol que se originaba en el descendiente de la derecha de la raíz en la Figura 4.2 ha sido eliminado. Además, la insatisfacibilidad del subárbol eliminado no puede detectarse con ninguna de las estrategias de simplificación estándar, como las técnicas basadas en el uso de grafos de detección de bucles [CR91], reglas de unificación [Fri85], compatibilidad de operadores [DS87], o normalización impaciente [ALN87, DS87].*

Cerramos este apartado demostrando, en el siguiente corolario, la corrección y completitud del nuevo cálculo refinado.

Corolario 5.2.6 *Dado un objetivo g, un programa $\mathcal{R}$ y un programa abstracto $\mathcal{R}_\mathcal{A} \propto \mathcal{R}$ que aproxima a $\mathcal{R}$, el conjunto de respuestas computadas $\{\sigma_{\lceil Var(g)} \mid \langle g, \epsilon\rangle \looparrowright^*_\varphi \langle \top, \sigma\rangle\}$ para g en $\mathcal{R}$ coincide (módulo renombramiento de variables) con el conjunto de éxitos $\mathcal{O}^\varphi_\mathcal{R}(g)$.*

DEMOSTRACIÓN. La corrección del nuevo cálculo es inmediata, ya que la relación $\looparrowright_\varphi$ está incluida en la correspondiente relación $\leadsto_\varphi$. La completitud se preserva como consecuencia del Teorema 5.2.3, ya que para toda derivación de éxito de la forma:

$$\langle g, \epsilon\rangle \leadsto^*_\varphi \langle g', \theta\rangle \leadsto^*_\varphi \langle \top, \sigma\rangle,$$

[1]Escribiremos $\looparrowright_{\varphi_{[u,r]}}$ cuando sea necesario distinguir la ocurrencia y la regla usadas.

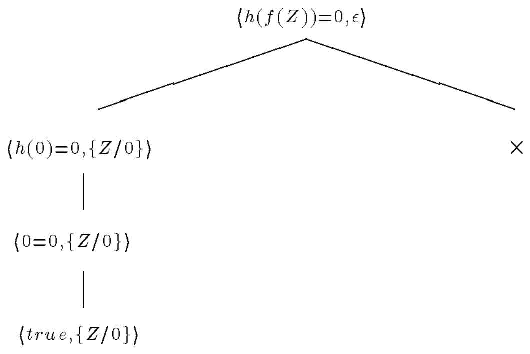

Figura 5.1: Narrowing básico refinado.

existe una sustitución $\kappa \in \Delta^{\varphi}_{\mathcal{R}_{\mathcal{A}}}(g)$ tal que $\kappa \diamond \theta$ y, por tanto, la derivación $\langle g, \epsilon \rangle \looparrowright^{*}_{\varphi}$ $\langle \top, \sigma \rangle$ también se puede probar en el cálculo refinado. $\square$

5.3 Satisfacción Perezosa de Restricciones Ecuacionales

Durante la última década, el paradigma de programación lógica tradicional ha sido generalizado al marco de la programación lógica con restricciones (CLP, *Constraint Logic Programming* [JL87]). CLP es un esquema de lenguajes de programación lógica, que extiende la programación lógica pura para incluir restricciones. En contraste con la programación lógica, en CLP el dominio de computación se deja sin especificar, aunque sujeto al cumplimiento de ciertas restricciones. Cada instancia CLP(X) del esquema es un lenguaje de programación, obtenido mediante la especificación de una estructura X de computación. La estructura define el dominio de discurso subyacente, así como las operaciones y relaciones sobre este dominio, proporcionando así una interpretación semántica del mismo. Por ejemplo, la programación lógica pura es una instancia de CLP, en la que la correspondiente estructura tiene como dominio el universo de Herbrand sobre un alfabeto finito, y el único símbolo de predicado para las restricciones es la igualdad (sintáctica).

Una de las diferencias fundamentales entre el paradigma de programación lógica y el marco CLP, consiste en la sustitución del concepto de unificación por la noción (más general) de resolución de restricciones [JL87]. Informalmente, un programa CLP(X) consta de un conjunto de cláusulas de la forma:

$$
\begin{array}{lll}
H \Leftarrow c. & & \texttt{\% hechos} \\
H \Leftarrow c \,\Box\, B_1, \ldots, B_n. & & \texttt{\% reglas}
\end{array}
$$

donde H y $B_1, \ldots, B_n$ son átomos, y c es una restricción sobre el dominio X. La posibilidad de incluir restricciones en los objetivos y en los cuerpos de las cláusulas incrementa sustancialmente la potencia expresiva de los programas.

La semántica operacional de este tipo de lenguajes se define, generalmente, mediante una regla de resolución como la que sigue[2]:

$$\frac{H \Leftarrow c_0 \,\Box\, \widetilde{B} \ll \mathcal{P} \;\wedge\; \widetilde{c} = (c_0, H = A_1) \;\wedge\; c \overset{\widetilde{c}}{\longrightarrow}_{CS} c, \widetilde{c}}{\Leftarrow c \,\Box\, A_1, \ldots, A_n \longrightarrow_{CLP} \Leftarrow c, \widetilde{c} \,\Box\, \widetilde{B}, A_2, \ldots, A_n}$$

Informalmente, es posible realizar un paso de resolución CLP siempre y cuando la restricción acumulada $(c, \widetilde{c})$ sea satisfacible, lo cual denotamos por $c \overset{\widetilde{c}}{\longrightarrow}_{CS} c, \widetilde{c}$. Nótese que la unificabilidad entre el átomo seleccionado A_1 y la cabeza de la cláusula H se comprueba al tiempo que se verifica la satisfacibilidad de la nueva restricción $(c, \widetilde{c})$, ya que la ecuación $H = A_1$ ha sido previamente añadida a la restricción $\widetilde{c}$. La relación $\longrightarrow_{CS}$ es la responsable de verificar la satisfacibilidad de la restricción acumulada (la etiqueta "CS" toma las iniciales de *Constraint Solver*).

El esquema CLP ofrece una visión unificada de varias extensiones potentes de la programación lógica pura, como la inclusión de características de orientación a objetos [AN86], aritmética real [JM87] o términos infinitos y desigualdades [JLM86]. En el siguiente apartado, presentamos una instancia particular del esquema CLP que integra, de manera simple y elegante, los paradigmas de programación lógica y funcional.

El Lenguaje CLP($\mathcal{H}/\mathcal{E}$)

Describimos brevemente una instancia del esquema CLP especializada en resolver ecuaciones con respecto a una teoría de Horn ecuacional $\mathcal{E}$ [Alp91, AFL95]. La estructura viene determinada por la partición más fina inducida por $\mathcal{E}$ sobre el universo de Herbrand $\mathcal{H}$. El símbolo $=$ es el único símbolo de predicado permitido en las restricciones, y se interpreta como la igualdad semántica sobre el dominio. Representamos dicha estructura por $\mathcal{H}/\mathcal{E}$, y denotamos por CLP($\mathcal{H}/\mathcal{E}$) a la correspondiente instancia del esquema CLP.

Informalmente, un programa CLP($\mathcal{H}/\mathcal{E}$) puede verse como la unión de dos conjuntos de cláusulas. Por un lado, las cláusulas CLP, que poseen la forma estándar vista anteriormente y cuyas restricciones consisten en secuencias de ecuaciones y, por otro lado, las cláusulas que definen la teoría de Horn ecuacional. Como es habitual, estas últimas pueden orientarse como un conjunto de reglas de reescritura condicionales (i.e. un SRTC). En este contexto, la relación que comprueba la satisfacibilidad de la restricción acumulada puede definirse como sigue.

[2]Denotamos con $\tilde{c}$ y $\widetilde{B}$ una secuencia de ecuaciones y una secuencia de átomos, respectivamente.

Definición 5.3.1 ($\rightarrow_{CS}$) *Sea $\mathcal{R}$ un SRTC y φ una estrategia de narrowing. Definimos la relación $\rightarrow_{CS}$ como la menor relación que satisface:*

$$\frac{\mathcal{O}^{\varphi}_{\mathcal{R}}(c, \widetilde{c}) \neq \emptyset}{c \ \stackrel{\tilde{c}}{\rightarrow}_{CS} \ c, \widetilde{c}}$$

En otras palabras, se hace uso directamente de (una instancia de) la relación de *narrowing* genérico con estrategia como mecanismo de comprobación de la satisfacibilidad de las restricciones. Cuando el conjunto de respuestas computadas es vacío, podemos afirmar que la nueva restricción $(c, \widetilde{c})$ es insatisfacible y, por tanto, la derivación CLP debe terminar con fallo. En caso contrario (basta con que exista una respuesta computada), queda demostrada su satisfacibilidad, y la restricción acumulada pasa a formar parte del nuevo objetivo CLP derivado.

Es evidente, no obstante, que el modelo de computación presentado posee dos graves inconvenientes:

- En primer lugar, el mecanismo de resolución de restricciones es extremadamente ineficiente. Esto se debe, principalmente, a que no se realizan las computaciones de manera incremental, es decir, en cada paso de resolución CLP se comprueba la satisfacibilidad de la nueva restricción acumulada $(c, \widetilde{c})$, sin explotar el hecho de que la restricción previa c se sabía satisfacible (ya que se demostró en el paso anterior).

- En segundo lugar, tenemos el problema de la (posible) no terminación de *narrowing*. Concretamente, cuando en el intento de demostrar la satisfacibilidad de una restricción el algoritmo de *narrowing* no termina, la correspondiente derivación CLP entra en ciclo sin computar ninguna respuesta.

En los siguientes apartados, hacemos uso de los distintos resultados obtenidos hasta el momento para mejorar los cálculos. Concretamente, la composicionalidad de la semántica nos permite introducir incrementalidad en el proceso de comprobación de la satisfacibilidad de las restricciones. Independientemente, mostramos cómo el uso de la relación de *narrowing* aproximado se puede utilizar para formalizar un mecanismo de resolución CLP perezoso, evitando así el problema de la posible no terminación de *narrowing*. Por último, se presenta una solución que integra ambas optimizaciones.

Resolución Incremental

En el contexto del marco CLP, la incrementalidad consiste en demostrar la satisfacibilidad de una secuencia de restricciones, transformando la solución existente para cada restricción previamente resuelta en una solución para la siguiente restricción [HP91]. Asumimos que las restricciones crecen monótonamente a lo largo de la derivación CLP y la cuestión es, por tanto, cómo realizar eficientemente el test de satisfacibilidad para las restricciones acumuladas en los sucesivos pasos. La siguiente definición introduce la noción de *problema de satisfacción de restricciones incremental.*

Definición 5.3.2 *Sean $c_0, \widetilde{c}_1, c_1, \ldots, \widetilde{c}_n, c_n$ restricciones, donde $c_i = (c_{i-1}, \widetilde{c}_i)$. Un problema de satisfacción de restricciones incremental consiste en comprobar (eficientemente) la satisfacibilidad de c_i, haciendo uso de la información sobre las computaciones previas para $c_0, \ldots, c_{i-1}$, $i = 1, \ldots, n$.*

La composicionalidad de la semántica $\mathcal{O}^{\varphi}_{\mathcal{R}}$ (para estrategias independientes del entorno) sugiere una solución simple y elegante al problema de la satisfacción de restricciones incremental. La idea básica consiste en, dadas las restricciones $c_0, \ldots, c_n$ computadas a lo largo de una derivación CLP, utilizar los resultados de composicionalidad de la semántica operacional para calcular, en cada paso, únicamente el conjunto de éxitos de la nueva restricción c_i, combinándolo con los anteriores mediante el operador de composición paralela $\Uparrow$. Para ello, es necesario almacenar junto a cada restricción el correspondiente conjunto de respuestas computadas, lo que nos lleva a la siguiente definición de iCS-estado (la etiqueta "iCS" toma las iniciales de *incremental Constraint Solver*).

Definición 5.3.3 (iCS-estado) *Un iCS-estado es un par $\langle c, \Theta \rangle$, donde c es una restricción (i.e. una secuencia de ecuaciones) y Θ es el conjunto de respuestas computadas asociado (es decir, $\mathcal{O}^{\varphi}_{\mathcal{R}}(c)$). El iCS-estado vacío se denota por el par $\langle \emptyset, \emptyset \rangle$.*

Introducimos ahora la nueva relación incremental que verifica la satisfacibilidad de las restricciones.

Definición 5.3.4 ($\rightarrow_{iCS}$) *Sea $\mathcal{R}$ un programa y φ una estrategia de narrowing. Definimos la relación $\rightarrow_{iCS}$ como la menor relación que satisface:*

$$\frac{\Theta' = \Theta \Uparrow \mathcal{O}^{\varphi}_{\mathcal{R}}(\widetilde{c}) \;\wedge\; \Theta' \neq \emptyset}{\langle c, \Theta \rangle \xrightarrow{\widetilde{c}}_{iCS} \langle (c, \widetilde{c}), \Theta' \rangle}$$

El nuevo procedimiento comienza con el iCS-estado vacío $\langle \emptyset, \emptyset \rangle$ y, en cada paso, calcula únicamente el conjunto de respuestas computadas para la nueva restricción, componiéndolo a continuación con el conjunto de respuestas de la restricción (acumulada) anterior. La relación $\rightarrow_{CLP}$ se extiende de manera natural para trabajar con iCS-estados en lugar de restricciones:

$$\frac{H \Leftarrow c_0 \,\square\, \widetilde{B} \ll \mathcal{P} \;\wedge\; \widetilde{c} = (c_0, H = A_1) \;\wedge\; \langle c, \Theta \rangle \xrightarrow{\widetilde{c}}_{iCS} \langle (c, \widetilde{c}), \Theta' \rangle}{\Leftarrow \langle c, \Theta \rangle \,\square\, A_1, \ldots, A_n \rightarrow_{CLP} \Leftarrow \langle (c, \widetilde{c}), \Theta' \rangle \,\square\, \widetilde{B}, A_2, \ldots, A_n}$$

Nótese que, si a lo largo de una derivación CLP el nuevo conjunto de sustituciones Θ' en un estado es $\emptyset$, la restricción acumulada es insatisfacible y podemos detener la derivación CLP con fallo. La corrección y completitud del nuevo cálculo se enuncia en el siguiente lema, y es una consecuencia inmediata del Teorema 3.2.6.

Lema 5.3.5 *Sea $\mathcal{R}$ un programa y φ una estrategia de narrowing. Dadas las restricciones $c, \widetilde{c}$, se cumple:*

$$c \xrightarrow{\tilde{c}}_{CS} c, \widetilde{c} \;\Leftrightarrow\; \langle c, \Theta\rangle \xrightarrow{\tilde{c}}_{iCS} \langle (c, \widetilde{c}), \Theta'\rangle,\; \Theta' = \Theta \Uparrow \mathcal{O}^{\varphi}_{\mathcal{R}}(\widetilde{c}).$$

Respecto a la mejora obtenida con el nuevo mecanismo, podemos considerar dos casos. Si deseamos obtener todas las soluciones para las restricciones, el nuevo cálculo es generalmente más eficiente que el anterior, ya que no es necesario recomputar todas las soluciones para la restricción acumulada de entrada. Por otro lado, si es suficiente con obtener una única solución para las restricciones (ya que con ello queda demostrada su satisfacibilidad), el mecanismo original es (en algunos casos) más eficiente y, además, el riesgo de no terminación es menor. Sin embargo, en ambos casos el riesgo de no terminación debido a la relación de *narrowing* sigue siendo demasiado alto.

Resolución Perezosa

En este apartado, presentamos una solución al problema de la no terminación del mecanismo de resolución de restricciones. Concretamente, la idea es utilizar la relación de *narrowing* aproximado, cuya terminación está garantizada, para definir un mecanismo de resolución CLP perezoso.

Informalmente, el nuevo procedimiento de resolución CLP realiza los siguientes pasos. Mientras el conjunto de átomos en el objetivo no es vacío, simplemente se comprueba en cada paso la (posible) satisfacibilidad de la restricción acumulada, haciendo uso del análisis estático definido por la relación $\hookrightarrow_{\varphi}$. Por la corrección del análisis, podemos asegurar que si el conjunto de éxitos abstracto es vacío, entonces la restricción es insatisfacible (y podemos abortar la derivación CLP de forma segura). Por otro lado, si llegamos a un objetivo donde la secuencia de átomos es vacía sin haber demostrado la insatisfacibilidad de la restricción acumulada, entonces podemos hacer uso de la relación de *narrowing* concreto para determinar la satisfacibilidad de la restricción final (así como las correspondientes respuestas computadas). La siguiente definición formaliza el nuevo cálculo para la verificación de las restricciones.

Definición 5.3.6 ($\rightarrow_{CA}$) *Sea $\mathcal{R}$ un programa, $\mathcal{R}_{\mathcal{A}} \propto \mathcal{R}$ un programa abstracto que aproxima a $\mathcal{R}$ y φ una estrategia de narrowing. Definimos la relación $\rightarrow_{CA}$ como la menor relación que satisface:*

$$\frac{\Delta^{\varphi}_{\mathcal{R}_{\mathcal{A}}}(c, \widetilde{c}) \neq \emptyset}{c \xrightarrow{\tilde{c}}_{CA} c, \widetilde{c}}$$

La nueva relación $\to_{CA}$ (la etiqueta "CA" toma las iniciales de *Constraint Analyzer*) es ahora la encargada de verificar la satisfacibilidad de la restricción acumulada. La (posible) satisfacibilidad de las restricciones se demuestra haciendo uso de la relación de *narrowing* abstracto, cuya terminación está siempre garantizada. El nuevo mecanismo de resolución CLP, definido por la siguiente regla de transición:

$$\frac{H \Leftarrow c_0 \,\Box\, \widetilde{B} \ll \mathcal{P} \ \wedge\ \widetilde{c} = (c_0, H = A_1) \ \wedge\ c \xrightarrow{\widetilde{c}}_{CA} c, \widetilde{c}}{\Leftarrow c \,\Box\, A_1, \ldots, A_n \to_{CLP} \ c, \widetilde{c} \,\Box\, \widetilde{B}, A_2, \ldots, A_n}$$

se dice perezoso, ya que en cada paso de computación CLP sólo se comprueba una aproximación a la satisfacibilidad de la restricción acumulada. Si se alcanza un objetivo CLP de la forma $\Leftarrow c \,\Box$, en el que la secuencia de átomos es vacía, entonces se debe hacer uso de la relación de *narrowing* concreto para verificar la satisfacibilidad de la restricción final. Esta última verificación, sin embargo, puede no terminar (e, incluso, resultar muy ineficiente). En este caso, ya no es posible demorar más el cómputo de las soluciones concretas para la restricción final (en caso de que sea satisfacible), con lo que la única posibilidad consiste en utilizar una estrategia de *narrowing* lo más optimizada posible. Dicha computación final puede efectuarse haciendo uso de la relación de *narrowing* guiado por el análisis. La corrección del nuevo mecanismo de satisfacción de restricciones es inmediato por la corrección de la relación de *narrowing* abstracto (Teoremas 4.2.19 y 4.3.7).

Resolución Perezosa Incremental

Presentamos ahora un refinamiento del procedimiento de resolución CLP que integra todas las optimizaciones anteriores. Informalmente, la idea consiste en tomar como base el cálculo perezoso del apartado anterior e incorporar las dos optimizaciones ya comentadas: en primer lugar, se calcula incrementalmente el conjunto de éxitos abstracto para la restricción acumulada y, en segundo lugar, se hace uso de la información del análisis (el conjunto de éxitos abstracto) para calcular de forma eficiente las respuestas computadas para la restricción final (haciendo uso de la relación de *narrowing* guiado por *narrowing* abstracto).

El nuevo modelo trabaja con estados (c, Θ) similares a los presentados en la Definición 5.3.3, con la única diferencia de que aquí Θ denota un conjunto de sustituciones *abstractas*. La siguiente definición formaliza el nuevo procedimiento de resolución CLP.

Definición 5.3.7 (resolución CLP perezosa incremental) *Sea $\mathcal{P}$ un programa CLP($\mathcal{H}/\mathcal{E}$), $\mathcal{R}$ un SRTC, $\mathcal{R}_{\mathcal{A}} \propto \mathcal{R}$ un programa abstracto que aproxima a $\mathcal{R}$ y φ una estrategia de narrowing. El procedimiento de resolución CLP perezoso incremental se define mediante el siguiente sistema de transición jerárquico:*

$$(1)\qquad \frac{\Theta' = \Theta \Uparrow_{\mathcal{A}} \Delta^{\varphi}_{\mathcal{R}_{\mathcal{A}}}(\widetilde{c}) \ \wedge\ \Theta' \not\equiv \emptyset}{\langle c, \Theta\rangle \overset{\widetilde{c}}{\rightarrow}_{iCA} \langle (c,\widetilde{c}), \Theta'\rangle}$$

$$(2)\qquad \frac{H \Leftarrow c_0 \ \Box\ \widetilde{B} \ll \mathcal{P} \ \wedge\ \widetilde{c} = (c_0, H = A_1) \ \wedge\ \langle c, \Theta\rangle \overset{\widetilde{c}}{\rightarrow}_{iCA} \langle (c,\widetilde{c}), \Theta'\rangle}{\langle c, \Theta\rangle \ \Box\ A_1, \ldots, A_n \rightarrow_{CLP} \ \langle (c,\widetilde{c}), \Theta'\rangle \ \Box\ \widetilde{B}, A_2, \ldots, A_n}$$

La principal diferencia con respecto al procedimiento perezoso introducido en la sección anterior, consiste en que ahora el conjunto de soluciones abstractas se obtiene de forma incremental (regla 1) haciendo uso de la relación $\rightarrow_{iCA}$ (la etiqueta "iCA" toma las iniciales de *incremental Constraint Analyzer*). Por otra parte, disponemos del conjunto de éxitos abstracto para la restricción final, lo que nos permite hacer uso de la relación de *narrowing* guiado por *narrowing* abstracto. Concretamente, si mediante la aplicación de la regla (2) se alcanza un estado terminal de la forma $\Leftarrow \langle c, \Theta\rangle \ \Box$, las soluciones de la computación CLP vienen determinadas por el conjunto:

$$\{\sigma \mid \langle c, \epsilon\rangle \looparrowright^*_{\varphi} \langle \top, \sigma\rangle\}.$$

De esta forma, se consigue reforzar la terminación del cálculo. Se ha implementado un prototipo de este método de análisis incremental en Prolog con buenos resultados. A continuación, se presentan algunos tiempos de ejecución para un programa ejemplo sencillo.

Ejemplo 16 *Sea $\mathcal{R}$ el siguiente programa (canónico por niveles):*

```
X + 0        →   X          ⇐
X + s(Y)     →   s(X + Y)   ⇐
parity(X)    →   even       ⇐   X = Y + Y
```

donde la función 'parity' se evalúa a 'even' cuando su argumento es par. Una posible aproximación correcta de dicho programa es el programa abstracto $\mathcal{R}_{\mathcal{A}}$:

```
X + 0        →   X      ⇐
X + s(Y)     →   s(⊥)   ⇐
parity(X)    →   even   ⇐   X = Y + Y
```

En la Tabla 5.1 se muestran los tiempos de ejecución para 5 posibles conjuntos de restricciones que se podrían generar durante una computación CLP. Se ha tomado, como base para analizar las restricciones ecuacionales, la relación de narrowing básico abstracto. La columna **CA** *muestra el tiempo de ejecución usando la relación $\rightarrow_{CA}$, que debe (re-)analizar en cada paso la restricción completa. La columna* **iCA** *muestra*

restricciones	CA	iCA	AbsNwing	APCom	Inc
`Z = 0, parity(X) = even`	*0.52*	*0.54*	*0.42*	*0.00*	
`Y + Z =` s^2`(0)`	*3.38*	*1.62*	*0.62*	*0.86*	
`Y = X + Z`	**10.62**	**5.24**	*0.44*	*4.76*	*2*
`parity(X) = even`	*0.42*	*0.42*	*0.30*	*0.00*	
`parity(Y) = even`	*3.88*	*2.80*	*0.32*	*1.20*	
`X + Y =` s^2`(0)`	**18.24**	**3.44**	*0.44*	*2.92*	*5.3*
`parity(X) = even`	*0.40*	*0.40*	*0.28*	*0.00*	
`Y = s(0), Z = X + Y`	*3.78*	*1.52*	*0.46*	*0.66*	
`X + Z =` s^3`(0)`	**26.36**	**4.44**	**fail**		*5.9*
`X + Y =` s^4`(0)`	*0.72*	*0.72*	*0.54*	*0.00*	
`parity(X) = even`	*3.66*	*2.10*	*0.32*	*1.46*	
`parity(Y) = even`	**19.48**	**3.64**	*0.30*	*2.62*	*5.34*
`parity(X) = parity(Y)`	*3.62*	*3.66*	*2.30*	*0.00*	
`X + Y = s(Z)`	*23.18*	*12.00*	*0.46*	*9.52*	
`Z = 0`	**20.26**	**1.14**	**fail**		*17.8*

CA *Analizador de Restricciones*
iCA *Analizador de Restricciones Incremental*
AbsNwing *Narrowing Abstracto*
APCom *Composición Paralela Abstracta*

Tabla 5.1: Tiempos del analizador de restricciones Incremental vs. No Incremental (en segundos, BIM-Prolog, SUN 3/80).

el tiempo de ejecución usando la relación $\to_{iCA}$, que obtiene de forma incremental el nuevo conjunto de éxitos abstracto. La columna **Inc** *(CA/iCA) muestra el incremento de eficiencia obtenido por el análisis incremental. Por último, los tiempos de la columna* **iCA** *son el resultado de la suma del tiempo para calcular el conjunto de éxitos abstracto para la nueva restricción (columna* **AbsNwing***), el tiempo de componer el nuevo conjunto de sustituciones abstractas con el anterior, haciendo uso del operador de composición paralela abstracto (columna* **APCom***), más un tiempo extra usado para realizar algunas simplificaciones relevantes únicamente para la implementación.*

Para terminar este punto, mencionar que no todas las posibilidades para incrementar la eficiencia han sido explotadas. Por ejemplo, dado que se han introducido (y demostrado correctas) las versiones composicionales tanto del cálculo de *narrowing* concreto como de su versión aproximada, se puede incorporar de manera directa paralelismo en el proceso de cómputo formalizado en la Definición 5.3.7. Es importante destacar que en las computaciones CLP las cláusulas del programa no son instanciadas al realizar pasos de computación, ya que el test de unificación se realiza incorporando la correspondiente ecuación $H = A$ a la nueva restricción. Por tanto, es posible calcular los conjuntos de éxitos abstractos de las condiciones contenidas en las cláusulas en tiempo de compilación, permitiendo así minimizar las computaciones a realizar en tiempo de ejecución. Concretamente, sólo sería necesario calcular las soluciones a las ecuaciones $H = A$, y hacer uso del operador de composición paralela abstracto para ir componiendo los distintos conjuntos de sustituciones abstractas.

Todo esto indica que la implementación de una instancia del marco CLP que integre la programación lógica y funcional es un proyecto viable. Más aún, si se utiliza el mecanismo de resolución CLP perezoso, las optimizaciones descritas anteriormente y una estrategia de *narrowing* eficiente (e.g. *narrowing* condicional perezoso), el modelo resultante tiene buenas oportunidades para explotar el paralelismo inherente de los programas, así como para realizar las computaciones de manera eficiente.

5.4 Análisis de Enlaces de Variables

El análisis estático formalizado en el Capítulo 4 permite obtener una aproximación correcta del conjunto de éxitos de un objetivo. Esta información es significativa para aquellos tipos de análisis en los que se desea verificar propiedades que se satisfacen en todas las derivaciones de éxito. Como ejemplo, vamos a desarrollar en esta sección dos análisis simples, que nos permiten mostrar cómo el conjunto de sustituciones abstractas (computado por *narrowing* abstracto) puede ser usado para extraer información precisa sobre los enlaces de las variables del objetivo durante la ejecución.

La información sobre los enlaces de las variables puede resultar útil para toda una serie de optimizaciones de programas. Por ejemplo, en la implementación de un modelo de paralelismo AND independiente [HR95], las subexpresiones sólo pueden ser ejecutadas en paralelo si son independientes, es decir, si sus variables están ligadas a términos que no comparten variables [JL89, MH89]. Dichas comprobaciones de independencia entre subexpresiones pueden evitarse en tiempo de ejecución si se dispone, en tiempo de compilación, de la información adecuada sobre los enlaces de las variables.

Vamos a definir dos tipos de análisis a partir de la información generada por la semántica abstracta. El primero, permite extraer información sobre aquellas variables del objetivo que se instanciarán, en todas las derivaciones de éxito, a un término básico (*groundness analysis*). El segundo, permite extraer información sobre la independencia de las variables del objetivo, es decir, dadas dos variables del objetivo, nos permite inferir si en todas las derivaciones de éxito dichas variables se instanciarán a términos que no comparten variables.

Decimos que una variable x es *básica bajo una sustitución* θ si y sólo si $Var(x\theta) = \emptyset$. Un par de variables $x, y \in V$ son *independientes bajo* θ si y sólo si los términos a los que dichas variables se instancian por θ no comparten variables, es decir, si y sólo si $Var(x\theta) \cap Var(y\theta) = \emptyset$. En la siguiente definición se introduce la función *ground*, que toma una sustitución abstracta κ y devuelve el conjunto de variables básicas bajo dicha sustitución κ.

Definición 5.4.1 *Dada una sustitución abstracta $\kappa \in Sub_{\mathcal{A}}$, definimos la función $ground: Sub_{\mathcal{A}} \to \wp V$ como sigue*[3]*:*

$$ground(\kappa) = \{x \mid Var(x\kappa) = \emptyset\}.$$

La siguiente definición formaliza la función *indep*, que toma una sustitución abstracta κ y devuelve el conjunto de variables independientes bajo κ.

Definición 5.4.2 *Dada una sustitución abstracta $\kappa \in Sub_{\mathcal{A}}$, definimos la función $indep: Sub_{\mathcal{A}} \to \wp(V \times V)$ como sigue:*

$$indep(\kappa) = \{(x, y) \mid x \neq y \ \wedge \ Var(x\kappa) \cap Var(y\kappa) = \emptyset\}.$$

Lógicamente, los enlaces para las variables de un objetivo dado dependen de la estrategia de *narrowing* empleada. Sea $\mathcal{R}$ un programa, g un objetivo ecuacional y φ una estrategia de *narrowing*. El conjunto de variables del objetivo g que se instanciarán a términos básicos en todas las derivaciones de éxito de *narrowing* es:

$$\bigcap_{\theta \in \mathcal{O}^{\varphi}_{\mathcal{R}}(g)} ground(\theta).$$

La información sobre la independencia de las variables del objetivo se puede extraer del siguiente conjunto:

$$\bigcap_{\theta \in \mathcal{O}^{\varphi}_{\mathcal{R}}(g)} indep(\theta).$$

En la siguiente definición, mostramos cómo obtener aproximaciones correctas de dichos conjuntos, a partir de las respuestas aproximadas obtenidas por *narrowing* abstracto.

Definición 5.4.3 *Sea $\mathcal{R}$ un programa, $\mathcal{R}_{\mathcal{A}} \propto \mathcal{R}$ un programa abstracto que aproxima a $\mathcal{R}$ y g un objetivo ecuacional. Dada una estrategia de narrowing φ, definimos la función $SGV_{\varphi}: Goal \to \wp V$ como sigue:*

$$SGV_{\varphi}(g) = \bigcap_{\kappa \in \Delta^{\varphi}_{\mathcal{R}_{\mathcal{A}}}(g)} ground(\kappa).$$

La función $SIV_{\varphi}: Goal \to \wp(V \times V)$ se define como sigue:

$$SIV_{\varphi}(g) = \bigcap_{\kappa \in \Delta^{\varphi}_{\mathcal{R}_{\mathcal{A}}}(g)} indep(\kappa).$$

[3]Nótese que en la Definición 4.2.1 se introduce $\perp$ como un nuevo símbolo de *variable*. Así, la función $Var(t)$ devuelve también las ocurrencias de $\perp$ que aparezcan en t.

El siguiente ejemplo ilustra la extracción de dicha información, usando la relación de *narrowing* básico abstracto.

Ejemplo 17 *Consideremos el siguiente programa $\mathcal{R}$, que computa el número de elementos de una lista:*

$$\begin{array}{rcl} noe([\,]) & \to & 0 \\ noe([h|t]) & \to & s(noe([t])) \end{array}$$

y el siguiente programa abstracto $\mathcal{R}_\mathcal{A}$ que lo aproxima de forma correcta:

$$\begin{array}{rcl} noe([\,]) & \to & 0 \\ noe([h|t]) & \to & s(\bot) \end{array}$$

La semántica abstracta para el objetivo inicial:

$$g \equiv (noe(l) = s(0), noe(ll) = x, x = 0, l = [a|ll])$$

usando la estrategia de narrowing básico es:

$$\Delta^{\blacksquare}_{\mathcal{R}_\mathcal{A}}(g) = \{\{x/0, l/[a], ll/[\,]\}\}.$$

Entonces, $SGV_\blacksquare(g) = \{x, l, ll\}$. El análisis muestra que, en los estados finales de cualquier derivación concreta de éxito para $\mathcal{R} \cup \{g\}$, todas las variables de g se encuentran instanciadas a un término básico.

La siguiente proposición establece la corrección de las funciones SGV_φ y SIV_φ, como consecuencia de la corrección del análisis basado en *narrowing* abstracto.

Proposición 5.4.4 *Sea $\mathcal{R}$ un programa, $\mathcal{R}_\mathcal{A} \propto \mathcal{R}$ un programa abstracto que aproxima a $\mathcal{R}$ y g un objetivo ecuacional. Dada una estrategia de narrowing φ, entonces se cumple:*

1. *Si $x \in SGV_\varphi(g)$ entonces, en toda derivación de éxito para $\mathcal{R} \cup \{g\}$, la variable x se instancia a un término básico.*

2. *Si $(x, y) \in SIV_\varphi(g)$ entonces, en toda derivación de éxito para $\mathcal{R} \cup \{g\}$, las variables x e y permanecen independientes.*

Demostración.

1. Sea $x \in SGV_\varphi(g)$. Entonces $\forall \kappa \in \Delta^\varphi_{\mathcal{R}_\mathcal{A}}(g).\ Var(x\kappa) = \emptyset$. Por el Teorema 4.2.19, para toda sustitución de respuesta computada $\theta \in \mathcal{O}^\varphi_\mathcal{R}(g)$ existe una sustitución abstracta $\kappa \in \Delta^\varphi_{\mathcal{R}_\mathcal{A}}(g)$ tal que $\kappa \propto \theta$. Por la Definición 4.2.5, es inmediato que si $\kappa \propto \theta$ entonces se cumple $\forall x \in V.\ (Var(x\kappa) = \emptyset \;\Rightarrow\; Var(x\theta) = \emptyset)$. Por tanto, $\forall \theta \in \mathcal{O}^\varphi_\mathcal{R}(g).\ Var(x\theta) = \emptyset$, lo que completa la prueba.

2. Sea $(x, y) \in SIV_{\varphi}(g)$. Entonces $\forall \kappa \in \Delta^{\varphi}_{\mathcal{R}_{\mathcal{A}}}(g).\ Var(x\kappa) \cap Var(y\kappa) = \emptyset$. De forma análoga al caso anterior, para toda sustitución de respuesta computada $\theta \in \mathcal{O}^{\varphi}_{\mathcal{R}}(g)$ existe una sustitución abstracta $\kappa \in \Delta^{\varphi}_{\mathcal{R}_{\mathcal{A}}}(g)$ tal que $\kappa \propto \theta$. Por la Definición 4.2.5, resulta inmediato que si $\kappa \propto \theta$ entonces se cumple $\forall x, y \in V.\ (Var(x\kappa) \cap Var(y\kappa) = \emptyset \quad \Rightarrow \quad Var(x\theta) \cap Var(y\theta) = \emptyset)$. Por tanto, $\forall \theta \in \mathcal{O}^{\varphi}_{\mathcal{R}}(g).\ Var(x\theta) \cap Var(y\theta) = \emptyset$, lo que concluye la prueba.

$\square$

Parte II

Transformación de Programas

Capítulo 6

Fundamentos de la EP de Programas

El objetivo general de las técnicas de transformación de programas es derivar programas "mejores" pero semánticamente equivalentes al programa original. La evaluación parcial (EP) es una técnica de transformación de programas que consiste en la especialización de un programa respecto a ciertos datos de entrada, conocidos en tiempo de compilación [Fut71]. El programa resultante puede ser ejecutado más eficientemente que el programa original ya que, usando el conjunto de datos conocidos (parcialmente), es posible evitar algunas computaciones en tiempo de ejecución que se realizarán, una única vez, durante el proceso de compilación. Por otro lado, las técnicas de EP han demostrado también su utilidad para la optimización del proceso de compilación, así como en el campo de la generación automática de compiladores [JGS93], permitiendo la obtención de un compilador para el lenguaje a partir de su intérprete (por evaluación parcial del propio evaluador respecto a éste).

Las técnicas convencionales de EP para programas funcionales se basan en la reducción de expresiones y en la propagación de constantes, mientras que las técnicas empleadas para la deducción parcial de programas lógicos explotan la propagación de parámetros basada en unificación. Han habido, hasta la fecha, pocos intentos de estudiar las relaciones entre las técnicas usadas en programación funcional y programación lógica [GS94]. Pensamos que un tratamiento unificado del problema establece las bases para una comparación precisa, y podría generar ideas para nuevos desarrollos en ambos campos.

Este capítulo se ha organizado como sigue. En la Sección 6.1, se describen algunos antecedentes de la EP, se introducen las motivaciones para el presente trabajo y se relaciona éste con otros trabajos previos existentes en la literatura sobre el tema. En la Sección 6.2 mostramos que, en el contexto de los lenguajes lógico–funcionales, la especialización de programas se puede basar directamente en el mecanismo operacional de *narrowing*. Debido al tratamiento que *narrowing* hace de las variables lógicas y, en particular, a la propagación bidireccional de parámetros debida a la unificación, nuestros resultados son más potentes (y las pruebas más simples) que los de otras aproximaciones comparables previas. Concretamente, demostramos para nuestro método un resultado de corrección fuerte que asegura la equivalencia del programa original y el transformado respecto a las respuestas computadas y no sólo respecto al conjunto de éxitos básicos. Desarrollamos los conceptos que sirven de base a la EP de programas lógico-funcionales y formalizamos un esquema de especialización basado en el desplegado de árboles de *narrowing* para el objetivo a evaluar. También formulamos las nociones de cierre e independencia que resultan esenciales para obtener todas las respuestas sin producir soluciones adicionales. Por último, en la Sección 6.3, establecemos, para la relación de *narrowing* condicional (bajo las condiciones anteriormente introducidas), la corrección y completitud fuertes del método de EP aquí propuesto.

6.1 Introducción

En general, el objetivo de las técnicas de EP es construir, dado un programa y algún tipo de restricción sobre su uso (e.g. información sobre algunos de los valores de los parámetros de entrada), un programa especializado o "residual" más eficiente, pero equivalente al original cuando se usa de acuerdo a la restricción impuesta [JGS93].

Una de las razones principales que suscitan el interés por las técnicas de EP es la posibilidad de conseguir implementaciones eficientes, por medios puramente automáticos, de programas generales y altamente parametrizables. La idea básica consiste en permitir la escritura de programas de propósito general, usualmente poco eficientes, y utilizar las técnicas de EP para generar programas especializados (de propósito concreto) cuya ejecución sea más eficiente [JGS93].

En el campo de las matemáticas, es bien conocido el proceso que permite generar una función unaria a partir de una función binaria, cuando se establece un valor fijo para uno de sus argumentos. Esta operación recibe el nombre de proyección. La EP consiste en una transformación muy similar, pero operando sobre programas en lugar de funciones. Una formulación simple, pero que describe con claridad las características de la EP de programas, es la siguiente [EJ88]. Sea $P = \{p_i\}_{i \in I}$ un dominio de programas, $D = \{d_j\}_{j \in J}$ un dominio de datos, y $Sem : P \times D^* \to D^*$ la función semántica asociada a un programa (con sus datos de entrada). Entonces, la función de EP $Part : P \times D^* \to P$ debe cumplir la siguiente propiedad:

$$Sem(p_0, (d_1, d_2)) = Sem(Part(p_0, d_1), d_2)$$

donde p_0 es un programa cuyos datos de entrada son (d_1, d_2) y $Part(p_0, d_1)$ es un programa "especializado" respecto a los datos de entrada d_1. Así, la semántica del programa original se debe preservar cuando se emplea el programa especializado sobre los datos desconocidos en el momento de la EP.

Si bien la idea intuitiva acerca de qué es un programa parcialmente evaluado resulta simple, existen toda una serie de cuestiones prácticas que deben abordarse para la definición de un método de EP:

- Las técnicas básicas de transformación empleadas en el método.
- La terminación del proceso completo.
- La preservación de la semántica (operacional) del programa original.
- La efectividad del proceso, i.e. la disminución del tiempo de ejecución, para una determinada clase de programas, sin que se produzca un aumento inaceptable del tamaño del código.

En el resto de este trabajo, consideramos cada una de estas cuestiones y desarrollamos nuestra solución particular, que contempla todos los aspectos mencionados.

Se puede considerar que el origen de la EP de programas se remonta a los años 50–60. No es estraño, pues, que exista una gran cantidad de literatura relacionada con el tema. En el siguiente apartado se resumen, sin ánimo de ser exhaustivos, los principales antecedentes de las técnicas de EP de programas lógicos y funcionales.

6.1.1 Antecedentes

La especialización de programas no es precisamente una idea reciente. Fue introducida, hace más de 40 años, por Kleene en su formulación del teorema s-m-n [Kle52], que demuestra básicamente la corrección de la transformación, y es una de las contribuciones más importantes de Kleene a la teoría de las funciones recursivas. Sin embargo, las cuestiones de eficiencia eran irrelevantes en las investigaciones de Kleene y, de hecho, su construcción obtenía programas especializados *menos* eficientes que los originales.

Las distintas técnicas actuales de EP no tuvieron un origen común, sino que surgieron varias técnicas muy similares en diferentes países y de manera independiente [Ers88]. Por un lado, encontramos los trabajos de Y. Futamura [Fut71, Fut83, Fut88], en los que se formulan las hoy llamadas "proyecciones de Futamura". Este trabajo es un precedente claro de las técnicas de EP, en el que se muestra cómo obtener compiladores, generadores de compiladores y generadores de generadores de compiladores a partir de intérpretes. Una de las aportaciones más relevantes del trabajo de Futamura fue el descubrimiento de que la especialización del propio evaluador parcial, tomando un intérprete como dato conocido en el proceso de especialización, generaba (esencialmente) como programa residual un compilador.

Por otro lado, los trabajos de V. Turchin y su grupo relacionados con la *supercompilación*, juegan también un papel importante dentro del desarrollo de la EP. Sus primeros trabajos datan de mediados de la década de los 70, e.g. [Tur74, Tur77], pero no fue hasta principios de los 80 cuando sus trabajos comenzaron a ser conocidos por la comunidad internacional [Tur79, Tur80a, Tur80b]. Aparte del problema del idioma (los primeros trabajos sólo están disponibles en ruso), otro inconveniente añadido para la difusión de los trabajos de Turchin fue su orientación exclusiva hacia el lenguaje de programación Refal [Tur89], un lenguaje funcional poco estándar según los parámetros occidentales. La técnica de supercompilación de Turchin está considerada, actualmente, como una de las más potentes para la especialización de programas funcionales [Jon94, SGJ94, Sør94].

Otro de los trabajos pioneros en el tema se debe a L.A. Lombardi y B. Raphael, cuya "computación incremental" para el lenguaje LISP [Lom67, LR64] se puede entender como una forma de EP de programas funcionales. De la misma época y sobre el mismo tema es también el trabajo de J. McCarthy [McC64]. Cabe destacar también los trabajos de Burstall y Darlington [BD77], que definen un conjunto de reglas de transformación de programas que son la base de una buena parte de las técnicas de EP, siendo las más populares las transformaciones de plegado y desplegado. Aunque muchas de las técnicas de EP no se describen explícitamente en términos de las reglas de [BD77], éstas pueden ser reformuladas, de hecho, en términos de transformaciones de plegado/desplegado. Mencionamos finalmente los trabajos de A.P. Ershov [Ers77, Ers78a, Ers78b], sobre el principio de computación parcial, y los trabajos de L. Beckman [Bec76] y A. Haraldsson [Har77, Har78], que hacen uso de la EP para optimizar la compilación de programas Lisp.

Si bien la EP ha sido aplicada intensivamente en el campo de la programación funcional durante más de tres décadas (ver, por ejemplo, [CD93, JGS93, SS88] para una lista exhaustiva de referencias), ésta también ha atraído un interés considerable en la comunidad de programación lógica, donde usualmente recibe el nombre de *deducción parcial*. El primer trabajo en el que se formula una técnica para la EP de programas lógicos se debe a Komorowski [Kom82]. Informalmente, un evaluador parcial $\mathcal{P}$ para un lenguaje lógico es una función que, dado un programa P y un objetivo G, deriva un programa residual más eficiente $P_G = \mathcal{P}(P, G)$ que computa las mismas respuestas para G (y cualquiera de sus instancias) que P.

No es hasta 1987, sin embargo, cuando Lloyd y Shepherdson establecen de manera formal y rigurosa los fundamentos de la deducción parcial de programas lógicos [LS87, LS91]. Pese a que no se define un procedimiento concreto para efectuar la deducción parcial del programa, Lloyd y Shepherdson muestran los requerimientos básicos para la corrección y completitud del programa transformado; estos requerimientos son las condiciones de independecia y cierre (*closedness*), las cuales garantizan, respectivamente, que el programa transformado no produce respuestas adicionales, y que todas las llamadas que pueden ocurrir durante la ejecución están cubiertas por alguna cláusula del programa.

Dado el interés que las técnicas de especialización han suscitado en ambas comunidades, parece razonable pensar que podrían aportar también resultados interesantes en el campo de los programas lógico–funcionales. En el siguiente apartado, se desarrollan más extensamente las ideas que han motivado el presente trabajo.

6.1.2 Motivación

Como ya se ha apuntado, las transformaciones de plegado y desplegado, introducidas originalmente por Burstall y Darlington en [BD77] para programas funcionales, son explotadas en mayor o menor medida por las distintas técnicas de EP. La técnica de desplegado consiste, esencialmente, en la sustitución de una llamada a función por su respectiva definición, aplicando la correspondiente sustitución. La técnica de plegado es la transformación inversa, es decir, el reemplazamiento de cierto fragmento de código por una llamada a función equivalente. En el caso de los programas funcionales, las técnicas de plegado/desplegado sólo involucran emparejamiento (*pattern-matching*). Esta aproximación a la transformación de programas basada en las operaciones de plegado/desplegado, fue adaptada por primera vez al caso de los programas lógicos por Tamaki y Sato [TS84], reemplazando el emparejamiento por la unificación en las reglas de transformación. El desplegado de un programa lógico consiste, por tanto, en aplicar un paso de resolución a un subobjetivo en el cuerpo de una cláusula de todas las formas posibles. Gracias al mecanismo de unificación, la EP de programas lógicos permite propagar información sintáctica sobre los datos (parciales) de entrada, tales como la estructura de los términos, y no sólo valores constantes (como en el caso del desplegado de programas funcionales). De esta forma, se obtiene un mecanismo más potente que en el caso de los programas funcionales.

Dado que las operaciones de plegado/desplegado sobre programas funcionales sólo involucran emparejamiento, y puesto que las llamadas a función en el cuerpo de las reglas del programa pueden contener variables, las transformaciones de Burstall y Darlington hacen uso de una operación previa de *instanciación* [BD77]. De esta forma, si una llamada a función no empareja con ninguna de las funciones definidas en el programa, sus variables pueden instanciarse de forma que empareje con alguna de ellas (ver, por ejemplo, la Sección 17.3 de [JGS93] y, en particular, el ejemplo de EP de la función de Ackermann). Sin embargo, hemos encontrado en la literatura pocas técnicas de transformación automáticas basadas en las transformaciones de instanciación + desplegado. Un ejemplo del uso de unificación en la técnica de desplegado se puede encontrar en [DR93], donde se utiliza para formular una aproximación a la síntesis de programas funcionales.

El mecanismo de ejecución de los programas lógico-funcionales (*narrowing*) utiliza la unificación para el paso de parámetros en las llamadas a función. De esta forma, se consigue de forma automática el efecto de la regla de instanciación. La especialización de programas basada en el mecanismo operacional de *narrowing* aporta una visión unificada de los mecanismos de ejecución y transformación, que nos permite desarrollar un marco simple y efectivo para la EP de programas lógico–funcionales. Siendo *narrowing* un mecanismo operacional bien estudiado, podemos hacer un uso inmediato de sus bien conocidas propiedades, sin tener que partir de cero para caracterizar el comportamiento de los métodos desarrollados (como sucede con los métodos convencionales de EP de los lenguajes funcionales, generalmente definidos *ad-hoc* para realizar la transformación). Más aún, nuestro trabajo demuestra que son posibles varias optimizaciones del esquema que son únicas al mecanismo de computación de los programas lógico–funcionales (como, por ejemplo, la inclusión de un proceso de simplificación determinista durante el proceso de EP). Debido a la posibilidad de explotar su componente funcional, la EP de un programa lógico-funcional puede resultar más efectiva que la EP de un programa lógico equivalente.

6.1.3 Trabajos Relacionados

No existen en la literatura muchos trabajos relacionados con la especialización de programas lógico–funcionales. Hemos encontrado, sin embargo, dos excepciones notables. En [LS75], Levi y Sirovich definen un procedimiento de EP para el lenguaje funcional TEL, usando un mecanismo de ejecución simbólica basado en unificación que puede llegar a entenderse como una forma de *narrowing* perezoso. En [DP88a], Darlington y Pull muestran cómo la unificación permite integrar los pasos de desplegado e instanciación (introducidos en el marco de transformación de programas definido en [BD77]), obteniendo así la habilidad del *narrowing* para tratar con variables lógicas. También se esboza un evaluador parcial para el lenguaje funcional HOPE (extendido con unificación). Sin embargo, en ninguno de estos trabajos se han abordado cuestiones de control, terminación o equivalencia semántica.

Los trabajos sobre supercompilación [Tur86] son, de entre la extensa literatura sobre transformación de programas, los más cercanos a nuestro trabajo. La supercompilación (*supervised compilation*) es una técnica de transformación para programas funcionales que consta de tres elementos básicos: *driving*, generalización y generación de programas residuales. La supercompilación no especializa el programa original, sino que construye un programa nuevo para la (especialización de la) llamada a función inicial, usando el mecanismo de *driving* [GS94]. El procedimiento de *driving* puede considerarse un mecanismo de transformación de funciones basado en unificación. Concretamente, hace uso de un tipo de maquinaria de evaluación similar a *narrowing* (perezoso) para construir "árboles de estados" (posiblemente infinitos) para un programa de entrada y una llamada dada. En los trabajos de Turchin sobre supercompilación se usa la siguiente terminología [Rom91]: "emparejamiento generalizado" para la unificación, "contracciones" para las sustituciones de *narrowing* y, muy a menudo en los textos, "driving" para el propio mecanismo de *narrowing*, es decir, la operación de instanciación de una llamada a función para todos los casos posibles, seguida por el desplegado de las distintas ramas. Gracias al procedimiento de *driving*, el supercompilador puede conseguir la misma cantidad de propagación de información (basada en unificación) y especialización del programa que en la EP de programas lógicos [GS94].

Los trabajos de Turchin describen el supercompilador para Refal (*Recursive Function Algorithmic Language* [Tur89]), un lenguaje funcional basado en emparejamiento con una noción de patrones poco usual. La semántica de Refal se da en términos de un intérprete de reescritura (con una estrategia de evaluación impaciente), mientras que el supercompilador (en el que se encuentra incorporado el procedimiento de *driving*) posee una estrategia de evaluación perezosa. La supercompilación subsume, entre otros, el procedimiento de deforestación [Wad88], la evaluación parcial y otras transformaciones estándar de programas funcionales [SGJ94]. Por ejemplo, la supercompilación es capaz de soportar ciertos tipos de demostración de teoremas, síntesis de programas e inversión de programas (ver [Sør94] para una exposición detallada). La supercompilación puede mejorar un programa incluso si todos los argumentos de las llamadas a función son variables, eliminando las redundancias debidas a los anidamientos funcionales o a las variables repetidas. En [Jon94], Jones formaliza la metodología de Turchin basada en *driving* sobre unos sólidos fundamentos semánticos, que no están ligados a ningún lenguaje de programación o estructura de datos particular.

El proceso de *driving* puede ser infinito y, en general, no preserva la semántica del programa, ya que puede extender el dominio de las funciones (ver e.g. [GS94, JGS93, SGJ94]). En [SG95, Tur88] se estudian distintas técnicas para asegurar la terminación del proceso de supercompilación. La idea de [Tur88] consiste en supervisar la construcción del árbol y, bajo determinadas condiciones, realizar una "vuelta atrás", i.e. plegar las configuraciones a uno de los estados previos, obteniendo de esta forma un grafo finito. A menudo, es necesario disponer de una operación de *generalización* para que el plegado a una configuración anterior sea posible. En [SG95] se estudia la terminación de la supercompilación *positiva*, una versión simplificada del algoritmo de Turchin en la que no se considera la propagación de información negativa en el proceso de especialización. En dicho trabajo, la terminación se garantiza siguiendo un método comparable a la aproximación general de Martens y Gallagher para asegurar la terminación global del proceso de deducción parcial [MG95].

En [GS94], Glück y Sørensen estudian la correspondencia entre la EP de programas lógicos (deducción parcial) y *driving*, estableciendo las relaciones entre la aplicación de *driving* sobre un programa funcional y la construcción de un árbol SLD para un programa Prolog similar. Los autores no reflejan la relación intrínseca existente entre los mecanismos de *driving* y *narrowing*. Pensamos que explotar esta correspondencia lleva a un mejor entendimiento de cómo el procedimiento de *driving* consigue sus efectos, así como facilita la respuesta a muchas preguntas sobre la corrección y terminación de la transformación. Nuestro trabajo se puede ver como una nueva formulación del mecanismo esencial de *driving* en términos más simples y familiares a la comunidad de programación lógica. Además, se libera al lenguaje de las fuertes restricciones sintácticas impuestas en [GS94, SGJ94], para no complicar en exceso la formulación del algoritmo de *driving*.

Otra aproximación relevante a las técnicas de transformación de programas se ha presentado recientemente en [San95b, San95a]. En general, las transformaciones de plegado/desplegado, definidas en [BD77], no garantizan ni la mejora en eficiencia ni la corrección total de la transformación. En [San95b], se introduce una condición (semántica) para la corrección total de las transformaciones sobre programas funcionales de orden superior. El principal resultado técnico consiste en que, si los pasos locales de transformación se realizan siguiendo un cierto criterio, entonces se puede asegurar la corrección total de la transformación. En [San95a], se muestra cómo el resultado anterior permite demostrar la corrección total de algunas de las principales técnicas automáticas de transformación de programas, incluyendo la deforestación [Wad88] y la supercompilación [Tur86].

6.2 EP de Programas Lógico–Funcionales

En esta sección, introducimos los conceptos fundamentales necesarios para la EP de programas lógico–funcionales, y demostramos la corrección y completitud de la transformación. Nuestra formalización se define siguiendo una aproximación similar al marco teórico presentado en [LS91] para la deducción parcial de programas lógicos.

En programación lógica, la idea de la EP de programas consiste, básicamente, en realizar los siguientes pasos [LS91]. Dado un programa P y un objetivo atómico G:

1. Construir un árbol SLD (finito) para $P \cup \{G\}$, conteniendo al menos un nodo distinto de la raíz.

2. Recoger los subobjetivos G_i y las correspondientes sustituciones θ_i de cada hoja no fallada, formando el conjunto de cláusulas $\{G\theta_i \leftarrow G_i\}$, llamadas resultantes.

Intuitivamente, los resultantes son respuestas condicionales al objetivo inicial. El motivo por el cual se exige que G sea atómico es que, en caso contrario, los resultantes obtenidos no serían cláusulas definidas (al contener más de un átomo en la cabeza).

Bajo ciertas condiciones, el conjunto de resultantes obtenido se puede considerar como una versión especializada del programa inicial para resolver el objetivo G (y cualquiera de sus instancias).

Cuando consideramos sistemas de reescritura condicionales como programas y una semántica operacional basada en *narrowing* condicional, no resulta inmediato decidir cuál puede ser la noción de *resultante* más apropiada. Sin embargo, para el caso de programas no condicionales (y una relación de *narrowing* definida sobre términos), la siguiente aproximación viene en mente de forma inmediata. Dado un SRT $\mathcal{R}$ y un término t, la EP de $\mathcal{R}$ con respecto a t consiste en:

1. Construir un árbol de *narrowing* (finito) para el término t con respecto al programa $\mathcal{R}$.

2. Recoger los términos t_i y las correspondientes sustituciones θ_i de cada hoja, formando el conjunto de reglas de reescritura $\{t\theta_i \to t_i\}$.

El siguiente ejemplo ilustra esta primera aproximación al problema.

Ejemplo 18 *Sea $\mathcal{R}$ el siguiente programa no condicional que define la suma de los números naturales:*

$$\begin{array}{rcl} x+0 & \to & x \\ x+s(y) & \to & s(x+y) \end{array}$$

La especialización del programa $\mathcal{R}$ con respecto al término $x+s(s(0))$ se genera siguiendo los siguientes pasos. Primero, se desarrolla un árbol de narrowing (finito) para dicho término, cuya construcción se detiene utilizando algún criterio. En este ejemplo, sólo es posible la derivación:

$$x+s(s(0)) \leadsto_{\{y/s(0)\}} s(x+s(0)) \leadsto_{\{y'/0\}} s(s(x+0)) \leadsto_{\epsilon} s(s(x))$$

con sustitución asociada (restringida a las variables del término de entrada) ϵ. A continuación, construimos el resultante asociado a dicha derivación: $\{x+s(s(0)) \to s(s(x))\}$. Entonces, el programa especializado $\mathcal{R}'$ consta únicamente de la siguiente regla:

$$x+s(s(0)) \quad \to \quad s(s(x)).$$

Resulta inmediato comprobar que, en este ejemplo, las formas normales computadas por narrowing para el término $x+s(s(0))$ con respecto a los programas $\mathcal{R}$ y $\mathcal{R}'$ coinciden (aunque no sucede así para todas sus instancias, sino sólo para aquéllas que están "cubiertas", de alguna forma, por el patrón de la llamada, como formalizaremos posteriormente).

La primera cuestión a resolver se plantea cuando nos proponemos extender la anterior noción de resultante al caso general de programas condicionales, con una relación de *narrowing* definida sobre objetivos ecuacionales. En primer lugar, nos restringimos a considerar sólo objetivos atómicos, por las mismas razones que en el caso de la deducción parcial de los programas lógicos. El siguiente ejemplo pone de manifiesto los problemas que deberemos afrontar al elegir esta aproximación.

Ejemplo 19 *Sea $\mathcal{R}$ el siguiente conjunto de reglas:*

$$\begin{array}{rcll} x+0 & \rightarrow & x & \\ x+s(y) & \rightarrow & s(x+y) & \\ parity(x) & \rightarrow & even & \Leftarrow y+y=x \end{array}$$

y $g \equiv (parity(0+z) = parity(z+0))$ un objetivo ecuacional atómico. Consideremos el árbol de búsqueda parcial formado por las siguientes derivaciones[1] "un paso":

$$\begin{array}{ll} \langle(\underline{parity(0+z)} = parity(z+0)), \epsilon\rangle & \leadsto \langle(y+y=0+z, even = parity(z+0)), \epsilon\rangle \\ \langle(parity(\underline{0+z}) = parity(z+0)), \epsilon\rangle & \leadsto \langle parity(0) = parity(0+0), \{z/0\}\rangle \\ \langle(parity(\underline{0+z}) = parity(z+0)), \epsilon\rangle & \leadsto \langle parity(s(0+y)) = parity(s(y)+0), \{z/s(y)\}\rangle \\ \langle(parity(0+z) = \underline{parity(z+0)}), \epsilon\rangle & \leadsto \langle(y+y=z+0, parity(0+z) = even), \epsilon\rangle \\ \langle(parity(0+z) = parity(\underline{z+0})), \epsilon\rangle & \leadsto \langle parity(0+z) = parity(z), \epsilon\rangle \end{array}$$

Los resultantes asociados, aplicando la definición de EP de la programación lógica, son:

$$\begin{array}{lcl} parity(0+z) & = & parity(z+0) \Leftarrow y+y=0+z, even = parity(z+0) \\ parity(0+0) & = & parity(0+0) \Leftarrow parity(0) = parity(0+0) \\ parity(0+s(y)) & = & parity(s(y)+0) \Leftarrow parity(s(0+y)) = parity(s(y)+0) \\ parity(0+z) & = & parity(z+0) \Leftarrow y+y=z+0, parity(0+z) = even \\ parity(0+z) & = & parity(z+0) \Leftarrow parity(0+z) = parity(z) \end{array}$$

Entonces observamos que: 1) estas reglas no constituyen una definición especializada de la función 'parity' de $\mathcal{R}$, sino que deben mas bien entenderse como alguna forma de especialización de la relación de igualdad; 2) no es inmediato establecer un criterio que permita orientar las cabezas de las cláusulas resultantes como reglas de reescritura; y 3) si se intenta ejecutar el objetivo g usando el anterior conjunto de resultantes, no se obtienen todas las soluciones que pueden obtenerse para g en el conjunto de cláusulas original (por ejemplo, ni siquiera es posible obtener la respuesta ϵ, aún si se incluyen también las reglas para la suma).

El resultado del ejemplo anterior no es muy satisfactorio, aunque sí es un resultado coherente. Si se realiza la especialización de una ecuación (cuyo operador principal

[1]Para una mayor simplicidad, las sustituciones en las derivaciones aparecen restringidas a las variables del objetivo inicial g.

es la igualdad), los resultantes obtenidos definen (una especialización de) la relación de igualdad. Nosotros estamos interesados en especializar funciones y la solución que adoptamos se puede resumir como sigue. Para especializar un programa $\mathcal{R}$ con respecto a un término s, obtenemos un árbol (finito) de *narrowing* para el objetivo "artificial" $s = y$, donde y es una variable nueva (i.e. $y \notin Var(s)$). La razón para usar este tipo de objetivos es computar los distintos términos a los que se puede reducir por *narrowing* el término s usando el programa $\mathcal{R}$. Así, consideramos derivaciones de la forma:

$$\langle s = y, \epsilon \rangle \leadsto^* \langle (e_1, \ldots, e_n, t = y), \theta \rangle$$

donde $(t = y)$ es la forma reducida (por *narrowing*) de la ecuación inicial $(s = y)$, y $(e_1, \ldots, e_n)$ son las ecuaciones introducidas por las condiciones de las reglas aplicadas durante la computación. Entonces, definimos el resultante asociado a dicha derivación mediante la regla:

$$s\theta \ \rightarrow \ t \ \Leftarrow \ e_1, \ldots, e_n.$$

Es inmediato observar que, dado el objetivo inicial $(s = y)$, el paso de *narrowing*:

$$\langle s = y, \epsilon \rangle \leadsto \langle (e_1, \ldots, e_n, t = y), \theta \rangle$$

usando el resultante $(s\theta \ \rightarrow \ t \ \Leftarrow \ e_1, \ldots, e_n)$ produce, exactamente, el mismo efecto (la misma respuesta computada y el mismo objetivo derivado) que la aplicación de las reglas usadas en la derivación que generó el resultante. Esta es, precisamente, la idea que se persigue con la noción de resultante, que formalizamos en la siguiente definición.

Definición 6.2.1 (resultante) *Sea $\mathcal{R}$ un programa y s un término. Sea $\mathcal{D} \equiv (\langle s = y, \epsilon \rangle \leadsto^* \langle (g, e), \theta \rangle)$ una derivación de narrowing condicional para el objetivo ecuacional $(s = y)$, $y \notin Var(s)$, con respecto al programa $\mathcal{R}_+$. Sea $\sigma = mgu(e)$, entonces el resultante de la derivación es: $((s \ \rightarrow \ y)\theta \ \Leftarrow \ g)\sigma$.*

La definición anterior de resultante parece algo más compleja que su contrapartida para programas lógicos. Veamos, mediante algunos ejemplos, la necesidad de computar el *mgu* σ.

Ejemplo 20 *Dada la regla $(f(x) \ \rightarrow \ x \ \Leftarrow \ a = x, b = x)$ en $\mathcal{R}$, el resultante de la derivación:*

$$\begin{array}{ll} \langle \underline{f(f(z))} = y, \epsilon \rangle & \leadsto \langle (a = f(z), b = f(z), \underline{f(z) = y}), \{x/f(z)\} \rangle \\ & \leadsto \langle (a = f(z), b = f(z), true), \{x/f(z), y/f(z)\} \rangle \end{array}$$

es la regla: $(f(f(z)) \ \rightarrow \ f(z) \ \Leftarrow \ a = f(z), b = f(z))$.

El resultante asociado a la derivación:

$$\langle \underline{f(z)} = y, \epsilon\rangle \quad \leadsto \langle (a = x, b = x, x = y), \{z/x\}\rangle$$

es la regla: $(f(y) \;\rightarrow\; y \;\Leftarrow\; a = y, b = y)$*. Nótese que, de no aplicarse el mgu* $\sigma = \{x/y\}$ $(\sigma = \{y/x\})$ *de la última ecuación* $x = y$*, habríamos obtenido la regla:* $(f(x) \;\rightarrow\; y \;\Leftarrow\; a = x, b = x, x = y)$*, que contiene una variable extra 'y' en la parte derecha de la cabeza de la regla.*
Por último, el resultante asociado a la derivación:

$$\begin{aligned}\langle \underline{f(z)} = y, \epsilon\rangle \quad &\leadsto \langle (\underline{a = x}, b = x, x = y), \{z/x\}\rangle \\ &\leadsto \langle (true, b = a, a = y), \{z/a, x/a\}\rangle\end{aligned}$$

es la regla: $(f(a) \;\rightarrow\; a \;\Leftarrow\; true, b = a)$*.*

Una vez que se dispone de una definición de resultante adecuada, la formalización del concepto de EP resulta muy simple. La EP de un término s en un programa $\mathcal{R}$ se obtiene construyendo un árbol de búsqueda (posiblemente incompleto[2]) para el objetivo $(s = y)$, y extrayendo posteriormente la definición especializada –los resultantes– asociados a las hojas del árbol. Decimos que un resultante es *trivial* si se ha obtenido aplicando la regla de unificación sintáctica directamente sobre el objetivo inicial. Nótese que siempre aparecerá un resultante trivial en el proceso de especialización ya que, dada la forma del objetivo inicial, la siguiente derivación siempre es posible: $\langle s = y, \epsilon\rangle \leadsto \langle true, \{y/s\}\rangle$, y da lugar al resultante trivial $s \;\rightarrow\; s$, que no debe ser considerado al construir la EP de un objetivo ecuacional, tal y como aparece en la siguiente definición.

Definición 6.2.2 (evaluación parcial)
Sea $\mathcal{R}$ *un programa,* s *un término e* $y \notin Var(s)$ *una variable nueva. Sea* τ *un árbol finito de narrowing (posiblemente incompleto) para el objetivo* $\langle s = y, \epsilon\rangle$ *en el programa extendido* $\mathcal{R}_+$*, conteniendo al menos un nodo distinto de la raíz. Dado el conjunto de hojas no falladas en las derivaciones de* τ*:* $\{\langle g_i, \sigma_i\rangle \mid i = 1, \ldots, k\}$*, formamos el conjunto de resultantes no triviales* $\{r_i \mid i = 1, \ldots, k-1\}$ *asociados a las derivaciones* $\{\langle s = y, \epsilon\rangle \leadsto^+ \langle g_i, \sigma_i\rangle \mid i = 1, \ldots, k\}$*. Al conjunto* $\{r_i \mid i = 1, \ldots, k-1\}$ *lo llamamos una* evaluación parcial de s en $\mathcal{R}$ (usando τ).

Esta definición se puede extender de manera inmediata a conjuntos de términos y a objetivos ecuacionales arbitrarios.

[2]De forma similar a [LS91], consideramos que las derivaciones pueden ser potencialmente incompletas. Una rama del árbol puede ser entonces de fallo, incompleta, de éxito o infinita.

Definición 6.2.3 *Sea $\mathcal{R}$ un programa y S un conjunto finito de términos (módulo variantes[3]). Una evaluación parcial de S en $\mathcal{R}$ (o EP de $\mathcal{R}$ con respecto a S) se define como la unión de las evaluaciones parciales de los elementos de S en $\mathcal{R}$.*

La evaluación parcial de un objetivo ecuacional $(s_1 = t_1, \ldots, s_n = t_n)$ en $\mathcal{R}$ consiste en la EP del conjunto de términos $\{s_1, \ldots, s_n, t_1, \ldots, t_n\}$ en $\mathcal{R}$.

Como ya hemos comentado, la restricción a objetivos de la forma $(s = y)$ se debe a que se quiere realizar la especialización del programa respecto a términos, y no respecto a ecuaciones. Puede parecer, entonces, que se está perdiendo cierto potencial para la especialización, debido a que los términos s_i y t_i de cada ecuación $s_i = t_i$ se evalúan parcialmente de forma independiente. Sin embargo, no existe tal pérdida de generalidad en restringir nuestro discurso a la EP de términos, ya que nuestra construcción también puede considerar una ecuación $s = t$ como un "término" y podemos así "forzar el método" para especializar el programa respecto al objetivo $(s = t) = y$. Más aún, si existe un símbolo de función (binario) libre $\otimes$ en la signatura, podemos realizar la especialización respecto a un conjunto de ecuaciones evaluando objetivos de la forma $(\otimes(s_1, \ldots \otimes (s_{n-1}, s_n)) = \otimes(t_1, \ldots \otimes (t_{n-1}, t_n))) = y$. Por supuesto, en estos casos, lo que nuestro método especializa es la relación de igualdad y el operador de conjunción de ecuaciones, respectivamente.

Una propiedad importante relacionada con nuestra noción de evaluación parcial es que la EP de un término en un programa noetheriano es un programa noetheriano. Para demostrar este resultado, introducimos algunos conceptos y lemas previos.

Definición 6.2.4 (reducción sin evaluación de las condiciones [BW95]) *Sean g, g' secuencias de ecuaciones y $\mathcal{R}$ un programa. La relación de* reescritura condicional sin evaluación de las condiciones *se define como: $g \twoheadrightarrow_{\mathcal{R}} g'$ si existe una ocurrencia $u \in O(g)$, una regla $(\lambda \to \rho \Leftarrow C) \ll \mathcal{R}_+$ y una sustitución σ tales que $g_{|u} = \lambda\sigma$ y $g' \equiv (C\sigma, g[\rho\sigma]_u)$.*

Dado un programa $\mathcal{R}$, denotamos por $\mathcal{R}_u$ la "parte incondicional" de $\mathcal{R}$, es decir, el conjunto de reglas obtenidas a partir de $\mathcal{R}$ por la eliminación de las condiciones (i.e. las reglas de $\mathcal{R}_u$ son las cabezas de las reglas de $\mathcal{R}$). El siguiente lema establece una correspondencia precisa entre las derivaciones de *narrowing* condicional y las de la reducción sin evaluación de las condiciones.

Lema 6.2.5 *Sean g, g' secuencias de ecuaciones y sea $\mathcal{R}$ un programa. Si $\langle g, \epsilon \rangle \leadsto^* \langle g', \theta \rangle$ en $\mathcal{R}$, entonces $g\theta \twoheadrightarrow^*_{\mathcal{R}} g'$.*

[3]En el resto del trabajo, aquellos términos que sean variantes se considerarán idénticos (y, de dichas variantes, se tomará una arbitraria). De la misma forma, la comparación entre (conjuntos de) términos se realizará módulo renombramiento de variables.

DEMOSTRACIÓN. La prueba resulta inmediata a partir de la corrección del cálculo de *narrowing* condicional y el hecho de que las reglas de los programas considerados no contienen variables extra. □

Finalmente, la siguiente proposición establece el resultado deseado.

Proposición 6.2.6 *La evaluación parcial de un término s en un programa noetheriano $\mathcal{R}$ es un programa noetheriano.*

DEMOSTRACIÓN. Las derivaciones consideradas en la Definición 6.2.1 (resultante) tienen la forma:

$$\langle s = y, \epsilon\rangle \leadsto^* \langle (g, true), \theta\rangle, \quad \text{o bien} \quad \langle s = y, \epsilon\rangle \leadsto^* \langle (g, t = y), \theta\rangle.$$

Tratamos los dos casos de manera independiente.

a) Podemos dividir la derivación $\langle s = y, \epsilon\rangle \leadsto^* \langle (g, true), \theta\rangle$ en la forma:

$$\langle s = y, \epsilon\rangle \leadsto^+ \langle (g', t = y), \gamma_1\rangle \leadsto \langle (g', true), \gamma_1\sigma\rangle \leadsto^* \langle (g, true), \gamma_1\sigma\gamma_2\rangle \equiv \langle (g, true), \theta\rangle$$

cuyo resultante asociado es: $((s \to y)\gamma_1\sigma\gamma_2 \Leftarrow g)$. Ya que existe la subderivación $(\langle s = y, \epsilon\rangle \leadsto^+ \langle (g', t = y), \gamma_1\rangle)$ entonces, por el Lema 6.2.5, existe la siguiente secuencia de reescritura condicional sin evaluación de las condiciones:

$$(s = y)\gamma_1 \twoheadrightarrow^*_{\mathcal{R}} g', t = y$$

lo que implica que existe la secuencia de reescritura no condicional:

$$s\gamma_1 \to^*_{\mathcal{R}_u} t.$$

Ya que $\mathcal{R}$ es noetheriano, existe un orden monótono bien fundado $\succ$ sobre $\tau(\Sigma \cup V)$ que es estable bajo sustitución (i.e. tal que, si $l \succ r$ entonces $l\sigma \succ r\sigma$, para toda sustitución σ de las variables de l y r) donde $\lambda \succ \rho$, para toda regla $(\lambda \to \rho \Leftarrow C) \in \mathcal{R}$ [DJ90]. Junto con la monotonicidad del orden, esta condición asegura que $t \succ s$ siempre que t se reescribe a s, para cualquier par de términos $t, s \in \tau(\Sigma \cup V)$. Por tanto, tenemos que $s\gamma_1 \succ t$. Puesto que las reglas de $\mathcal{R}$ no contienen variables extra, entonces $Var(t) \subseteq Var(s\gamma_1)$ y $Var(g') \subseteq Var(s\gamma_1)$.

Tenemos que demostrar que $s\gamma_1\sigma\gamma_2 \succ y\gamma_1\sigma\gamma_2$, $Var(y\gamma_1\sigma\gamma_2) \subseteq Var(s\gamma_1\sigma\gamma_2)$ y $Var(g) \subseteq Var(s\gamma_1\sigma\gamma_2)$. Consideramos, entonces, dos casos:

i) Si σ es de la forma $\{y/t\}$, entonces $y\gamma_1\sigma \equiv y\sigma \equiv t$ y $s\gamma_1\sigma \equiv s\gamma_1$, ya que $y \notin Var(s)$. Por la estabilidad de $\succ$, $s\gamma_1 \succ t \Rightarrow s\gamma_1\gamma_2 \succ t\gamma_2$, esto es, $s\gamma_1\sigma\gamma_2 \succ y\gamma_1\sigma\gamma_2$. Ahora, los hechos $Var(t) \subseteq Var(s\gamma_1)$ y $Var(g') \subseteq Var(s\gamma_1)$ implican que $Var(t\gamma_2) \subseteq Var(s\gamma_1\sigma\gamma_2)$ y $Var(g) \subseteq Var(s\gamma_1\sigma\gamma_2)$, respectivamente.

ii) Si $t \equiv x$, $x \in V$ y σ es de la forma $\{x/y\}$, entonces $y\gamma_1\sigma \equiv y$, ya que $y \notin Var(s)$. Por la estabilidad de $\succ$, $s\gamma_1 \succ t \Rightarrow s\gamma_1 \succ x \;\Rightarrow\; s\gamma_1\sigma \succ y \Rightarrow s\gamma_1\sigma\gamma_2 \succ y\gamma_2$, esto es, $s\gamma_1\sigma\gamma_2 \succ y\gamma_1\sigma\gamma_2$. Ahora, tenemos que $Var(t) \subseteq Var(s\gamma_1) \;\Rightarrow\; x \in Var(s\gamma_1) \Rightarrow x\sigma \equiv y \in Var(s\gamma_1\sigma) \Rightarrow Var(y\gamma_2) \subseteq Var(s\gamma_1\sigma\gamma_2)$. Análogamente, $Var(g') \subseteq Var(s\gamma_1) \;\Rightarrow\; Var(g) \subseteq Var(s\gamma_1\sigma\gamma_2)$.

b) Consideremos la siguiente derivación:

$$\langle s = y, \epsilon\rangle \leadsto^* \langle (g, t = y), \theta\rangle$$

que produce el resultante: $((s \;\rightarrow\; y)\theta \;\Leftarrow\; g)\sigma$, donde $\sigma = mgu(\{t = y\})$. La demostración es perfectamente análoga al caso a), considerando que el resultante computado podría ser producido también por una derivación de la forma:

$$\langle s = y, \epsilon\rangle \leadsto^+ \langle (g, t = y), \theta\rangle \leadsto \langle (g, true), \theta\sigma\rangle.$$

$\square$

Sin embargo, es bien conocido que la EP de programas no preserva, en general, la semántica del programa original. En el caso de la deducción parcial de programas lógicos, se introducen unas condiciones de "cierre" (*closedness*) e "independencia" [LS91] que garantizan la equivalencia semántica entre el programa original y el programa especializado. En la siguiente sección, formulamos una extensión adecuada de estas nociones que nos permite demostrar la corrección y completitud de la EP de programas lógico-funcionales, con respecto a la semántica de las respuestas normalizadas computadas por *narrowing* condicional.

6.3 Corrección y Completitud de la EP

Consideramos, en primer lugar, la completitud de la transformación. Para garantizar que todas las respuestas del programa original se pueden computar usando las reglas del programa especializado, es necesario imponer algún tipo de condición de cierre sobre el mismo. Informalmente, la condición de cierre debe garantizar que todas las llamadas a función que pueden ocurrir durante la ejecución del programa especializado, están cubiertas por alguna regla del mismo.

En [LS91], se define la condición de cierre de la siguiente forma: dado un conjunto de átomos S y un programa especializado P, se dice que P está cerrado o cubierto por S sii todos los átomos que aparecen en P son instancia de algún átomo de S. El siguiente ejemplo ilustra cómo el uso de una noción de cierre similar (basada en la relación "ser instancia de") no es suficiente para garantizar la completitud de la transformación en el caso de los programas lógico-funcionales.

Ejemplo 21 *Consideremos el siguiente programa $\mathcal{R}$:*

$$\begin{array}{rcl} h(x) & \rightarrow & x \\ f(0) & \rightarrow & 0 \\ f(c(x)) & \rightarrow & h(f(x)) \end{array}$$

Una posible EP de $\mathcal{R}$ con respecto al conjunto de términos $S = \{f(c(x)),\ h(x)\}$ es el programa especializado $\mathcal{R}'$:

$$\begin{array}{rcl} h(x) & \rightarrow & x \\ f(c(x)) & \rightarrow & h(f(x)) \end{array}$$

En este caso, pese a que todos los términos que aparecen en $\mathcal{R}'$ son instancia de alguno de los términos en S, el programa $\mathcal{R}'$ no puede ser considerado cerrado respecto a S, ya que la llamada a función $f(x)$ que aparece en el término $h(f(x))$ no está suficientemente cubierta por las reglas de $\mathcal{R}'$ –ya que no puede ser reducida al término 0, tal y como era posible haciendo uso de la segunda regla del programa original $\mathcal{R}$–. Como consecuencia de ello, el objetivo $f(c(0)) = 0$ (que está supuestamente cubierto por S), tiene éxito en $\mathcal{R}$ con respuesta computada ϵ mientras que falla en $\mathcal{R}'$.

Para extender el concepto de *cierre*, introducimos la función *terms*, que permite extraer de forma selectiva los términos a considerar de una expresión dada.

Definición 6.3.1 *Sea O una expresión consistente en una regla (o conjunto de reglas) de reescritura, o en una ecuación (o conjunto de ecuaciones). Definimos la función $terms(O)$ como sigue:*

$$terms(O) \;=\; \left\{ \begin{array}{ll} \bigcup_{i=1}^{n} terms(o_i) & si\ O \equiv \{o_1, \ldots, o_n\} \\ \{\rho\} \cup terms(\{e_1, \ldots, e_n\}) & si\ O \equiv (\lambda \rightarrow \rho \Leftarrow e_1, \ldots, e_n) \\ \{l, r\} & si\ O \equiv (l = r) \end{array} \right.$$

A continuación, definimos la condición de cierre para un conjunto de términos y la extendemos al caso de programas. En adelante, diremos que un conjunto de términos T cumple la condición de cierre respecto a un conjunto de términos S, o simplemente que T es S-cerrado, si el predicado $closed(S, T)$ es cierto. Formalmente,

Definición 6.3.2 (cierre) *Sean S y T dos conjuntos finitos de términos. Decimos que T es S-cerrado si $closed(S, T)$, donde el predicado closed se define inductivamente como sigue:*

$$closed(S, O) \;\Leftrightarrow\; \left\{ \begin{array}{ll} true & si\ O \equiv \emptyset\ \vee\ O \equiv x \in V \\ closed(S, t_1)\ \wedge\ \ldots\ \wedge\ closed(S, t_n) & si\ O \equiv \{t_1, \ldots, t_n\} \\ closed(S, \{t_1, \ldots, t_n\}) & si\ O \equiv c(t_1, \ldots, t_n),\ c \in \mathcal{C} \\ (\exists s \in S.\ s\theta = O)\ \wedge\ closed(S, terms(\widehat{\theta})) & si\ O \equiv f(t_1, \ldots, t_n),\ f \in \mathcal{F} \end{array} \right.$$

Decimos que un término t es S-cerrado si $closed(S,t)$ es cierto, y decimos que un programa $\mathcal{R}$ es S-cerrado si $closed(S, terms(\mathcal{R}))$ es cierto.

Informalmente, un término t es S-cerrado si se cumple alguna de las siguientes condiciones: 1) t es una variable, 2) t es un término constructor, ó 3) t es una instancia de un término $s \in S$, de forma que $t = s\theta$, y además se verifica que los términos de $\widehat{\theta}$ son S-cerrados.

En el resto del trabajo resultará útil disponer, no sólo de la información acerca de si un término t es S-cerrado, sino de cúal ha sido el conjunto de ocurrencias $\{u_1, \ldots, u_n\}$ que han permitido demostrar que lo es, verificando que, para todo $i \in \{1, \ldots, n\}$, $t_{|u_i}$ es instancia de algún término $s \in S$, $t_{|u_i} = s\theta$, y $terms(\widehat{\theta})$ es también S-cerrado. Esto nos lleva a introducir la siguiente noción de *conjunto de ocurrencias de cierre*.

Definición 6.3.3 (conjunto de ocurrencias de cierre) *Sea S un conjunto de términos y t un término S-cerrado. Definimos el conjunto de ocurrencias de cierre $CSet : \wp\mathcal{T} \times \mathcal{T} \rightarrow \wp\wp I\!N^*$ como sigue:*

$$CSet(S,t) = \{O \mid O \in c\_set(S,t), \quad u.0 \notin O, \quad u \in I\!N^*\}$$

donde la función auxiliar c_set se define inductivamente como sigue:

$$c\_set(S,t) \ni \begin{cases} \emptyset & si\ t \in V \quad o \quad t \equiv c \in \mathcal{C} \\ \displaystyle\bigcup_{i \in \{1,\ldots,n\}} \{i.p \mid p \in c\_set(S,t_i)\} & si\ t \equiv c(t_1, \ldots, t_n),\ c \in \mathcal{C} \\ \{\Lambda\} \cup \{u.p \mid u \in O_V(s) \wedge p \in c\_set(S, s_{|u}\theta)\} & si\ t \equiv f(t_1, \ldots, t_n),\ f \in \mathcal{F} \\ & y\ \ \exists s \in S.\ s\theta = t \\ \{0\} & en\ otro\ caso \end{cases}$$

Las ocurrencias que terminan en el marcador '0' identifican la situación en la que algún subtérmino de t no es instancia de ninguno de los términos de S. Así, los conjuntos que contienen alguna ocurrencia de la forma $u.0$ no deben ser considerados conjuntos de ocurrencias de cierre. La función $CSet$ se puede extender a ecuaciones y objetivos ecuacionales de la forma obvia.

Nótese que, dado un conjunto S, la forma en que un término t puede ser S-cerrado usando la Definición 6.3.2 no es única. Consecuentemente, la función $CSet$ es indeterminista también y pueden existir, por tanto, distintos conjuntos de ocurrencias de cierre que demuestran que un término t es S-cerrado. El siguiente ejemplo ilustra esta posibilidad.

Ejemplo 22 *Sea S el conjunto de términos $\{f(x), g(x), f(g(x))\}$ y t el término $f(g(0))$. Podemos verificar que t es S-cerrado de dos formas:*

1. *Tomando* $f(g(x)) \in S$, *tenemos que* $f(g(x))\{x/0\} = t$ *y, trivialmente, el predicado* $closed(S, terms(\{x = 0\}))$ *es cierto. El conjunto de ocurrencias de cierre asociado es, en este caso,* $\{\Lambda\}$.

2. *Tomando* $f(x) \in S$, *tenemos que* $f(x)\{x/g(0)\} = t$ *y el predicado* $closed(S, terms(\{x = g(0)\}))$ *es cierto, dado que ahora podemos tomar* $g(x) \in S$ *y tenemos que* $g(x)\{x/0\} = g(0)$, *y* $closed(S, terms(\{x = 0\}))$ *es cierto también. En este caso, el conjunto de ocurrencias de cierre asociado es* $\{\Lambda, 1\}$.

De esta forma, obtenemos: $CSet(S, t) = \{\{\Lambda\}, \{\Lambda, 1\}\}$.

Pasamos, a continuación, a establecer la corrección y completitud de nuestra propuesta para la EP de programas lógico-funcionales. Enunciamos, en primer lugar, una serie de definiciones y lemas previos. La siguiente definición introduce una noción de complejidad asociada con una derivación, similar a la que puede encontrarse en [OMI95].

Definición 6.3.4 (complejidad) *Definimos la complejidad* $|\mathcal{D}|$ *de una derivación de narrowing* $\mathcal{D} \equiv (\langle g, \sigma\rangle \leadsto^* \langle g', \sigma'\rangle)$ *como el número de pasos de narrowing en* $\mathcal{D}$ *que no usan la regla* $(x = x \ \rightarrow \ true)$.

La completitud fuerte del *narrowing* condicional se puede formular como sigue. Básicamente, nuestro enunciado establece que la completitud y la completitud fuerte coinciden, cuando nos restringimos a sustituciones normalizadas.

Lema 6.3.5 (completitud fuerte) *Sea* $\mathcal{R}$ *un programa confluente y* $\mathcal{S}$ *una función de selección. Para toda derivación de narrowing condicional de éxito* $\mathcal{D} \equiv (\langle g, \epsilon\rangle \leadsto^* \langle \top, \theta\rangle)$ *tal que* $\theta_{\lceil Var(g)}$ *está normalizada, existe una derivación* $\mathcal{D}_{\mathcal{S}} \equiv (\langle g, \epsilon\rangle \leadsto^* \langle \top, \theta\rangle)$ *que respeta* $\mathcal{S}$ *con la misma complejidad.*

DEMOSTRACIÓN. La prueba es estándar, ver e.g. [OMI95]. □

Introducimos ahora algunas transformaciones de aplanamiento que facilitan nuestras pruebas de completitud. Las demostraciones de Lloyd y Shepherdson en [LS91] no usan este tipo de transformaciones, sino que se basan en el uso de una forma general de resolución que emplea unificadores que no son los más generales. La técnica introducida en nuestras demostraciones podría también simplificar las pruebas de [LS91].

Informalmente, un aplanamiento del objetivo g se consigue reemplazando una ocurrencia de un término no variable t de g por una variable nueva x, y añadiendo la ecuación $(t = x)$ al objetivo resultante. Denotamos por $\breve{O}(g)$ el conjunto de ocurrencias no variables de g que no referencian ni ecuaciones ni la expresión g completa, i.e. $\breve{O}(s_1 = t_1, \ldots, s_n = t_n) = \bar{O}(s_1 = t_1, \ldots, s_n = t_n) - \{\Lambda, 1, \ldots, n\}$.

Definición 6.3.6 (aplanamiento simple) *Dado un objetivo no vacío g, definimos la función $flat$: $Goal \rightarrow \wp Goal$ como sigue:*

$$flat(g) = \{g' \in Goal \mid \; g' \equiv (g_{|u} = x, g[x]_u), \text{ donde } x \text{ es una variable nueva y la ocurrencia } u \text{ pertenece a } \check{O}(g)\}.$$

La transformación de aplanamiento se demuestra correcta y completa en conjunción con el procedimiento de *narrowing* básico [NRS89] bajo las condiciones de completitud para éste. En el caso de la relación de *narrowing* condicional, la transformación es trivialmente completa. Basta pensar que es suficiente "plegar" el subtérmino aplanado a su posición original (haciendo uso de la regla de unificación sintáctica) y, a continuación, imitar en su mismo orden los pasos de la derivación para el objetivo original. La transformación de aplanamiento resulta clave para nuestra prueba de corrección de la EP (y no nos interesa invertirla), como también lo es el que las derivaciones a partir del objetivo aplanado utilicen las reglas del programa en el mismo orden que las de la derivación original. Sin embargo, si excluimos las derivaciones que invierten en algún momento dicha transformación inicial, plegando de nuevo el término a su posición original (derivaciones que no son de interés), las restantes derivaciones que parten del objetivo aplanado podrían no aplicar las reglas del programa en el mismo orden que las del objetivo original, incluso aunque el programa considerado sea confluente y sin variables extra. El siguiente ejemplo ilustra esta dificultad.

Ejemplo 23 *Sea $\mathcal{R}$ el siguiente programa confluente y noetheriano:*

$$\begin{array}{rcccl} f(x) & \rightarrow & a & \Leftarrow & x = b,\; x = c \\ d & \rightarrow & b & & \\ d & \rightarrow & c & & \\ b & \rightarrow & c & \Leftarrow & f(d) = a \end{array}$$

Dado el objetivo $g \equiv (f(d) = a)$, existe la siguiente derivación de éxito:

$$\begin{array}{rcl} \langle \underline{f(d)} = a, \epsilon\rangle & \leadsto & \langle (\underline{d} = b, d = c, a = a), \{x/d\}\rangle \\ & \leadsto & \langle (b = b, \underline{d} = c, a = a), \{x/d\}\rangle \\ & \leadsto & \langle (b = b, c = c, a = a), \{x/d\}\rangle \\ & \leadsto^* & \langle \top, \{x/d\}\rangle \end{array}$$

que computa la respuesta (normalizada) ϵ. Consideramos ahora el objetivo aplanado $g' \equiv (d = x,\; f(x) = a)$. Resulta inmediato comprobar que es imposible construir una derivación de éxito para el objetivo g' en $\mathcal{R}$ que aplique las reglas en el mismo orden que las de la derivación original, sin plegar de nuevo el término extraído a la posición anidada que ocupaba dentro del objetivo original.

Intuitivamente, el problema que se presenta en el ejemplo anterior se debe a que la transformación de aplanamiento consigue un efecto similar a la técnica de compartición de variables (*sharing*) utilizada en la programación funcional, ya que el subtérmino desanidado no se "propaga" (cuando se aplican reglas del programa) a las múltiples copias de éste que puedan ser introducidas en el objetivo derivado a través de la parte derecha de la cabeza y el cuerpo de la regla empleada, sino que en su lugar se generan múltiples copias de la variable nueva (que sustituye al término aplanado). Cuando se trabaja con una implementación basada en *sharing*, la estrategia de *narrowing* condicional no es completa bajo las condiciones habituales de confluencia (y noetherianidad) del programa. En el ejemplo anterior, el término al que se instancia la variable x de la primera regla debería ser reducido a dos términos distintos, lo que es imposible si se mantiene el término desanidado (como lo sería también, de alguna forma, en una implementación basada en *sharing*). Resulta interesante observar la estrecha relación existente entre la relación de *narrowing* básico y la técnica de compartición de variables por *sharing*. Concretamente, la estrategia básica no permite la reducción de los términos introducidos por instanciación con lo que, trivialmente, una implementación basada en esta forma de *sharing* no afecta a la completitud del cálculo. El programa del Ejemplo 23 se usa en [MH94] para ilustrar la incompletitud del *narrowing* condicional básico bajo el requerimiento estándar de programas confluentes y noetherianos, y cómo se recupera la completitud al exigir adicionalmente que los programas sean decrecientes (o confluentes por niveles, alternativamente) cuando no hay variables extra [MH94]. Resulta inmediato comprobar que esta misma condición asegura que no se pierde completitud como consecuencia del aplanamiento en las derivaciones de *narrowing* condicional. En el resto de esta sección, excepto que se impongan explícitamente otras condiciones, consideramos que los programas con los que se trabaja son confluentes y también decrecientes.

Los siguientes lemas establecen una forma de equivalencia, respecto a las sustituciones de respuesta computada, entre las derivaciones para un objetivo y para cualquiera de sus formas aplanadas. El primer resultado muestra que, dado un objetivo g y uno de sus aplanamientos g', toda respuesta computada para g es una respuesta computada también para g'. Por abuso, usamos la terminología "derivación de éxito" para denotar derivaciones que parten de un estado $\langle g, \sigma\rangle$ y alcanzan un estado $\langle \top, \sigma\theta\rangle$. Considerar este tipo de estados iniciales resulta conveniente cuando se plantea aplicar este lema a la prueba por inducción de resultados posteriores.

Lema 6.3.7 *Sea g un objetivo ecuacional. Si existe una derivación de narrowing condicional de éxito $\mathcal{D} \equiv (\langle g, \sigma\rangle \leadsto^* \langle \top, \sigma\theta\rangle)$ entonces, para todo objetivo $g' \in flat(g)$, existe una derivación $\mathcal{D}' \equiv (\langle g', \sigma\rangle \leadsto^* \langle \top, \sigma\theta'\rangle)$ tal que $\theta = \theta'[Var(g)]$ y $\mathcal{D}'$ emplea las mismas reglas y en el mismo orden que $\mathcal{D}$ (sobre las posiciones correspondientes), más una aplicación extra de la regla $(x = x \ \rightarrow \ true)$.*

DEMOSTRACIÓN. La prueba es inmediata. □

Para aplicar la propiedad de completitud fuerte a las derivaciones que parten de un objetivo $g' \equiv (g_{|u} = x, g[x]_p) \in flat(g)$ y computan respuestas θ' que, restringidas a las variables de g, están normalizadas, es necesario exigir que los términos introducidos por la instanciación de la variable nueva x no hayan sido reducidos por *narrowing* en la correspondiente derivación para el objetivo g'. Como consecuencia de ello, el resultado contrario del Lema 6.3.7 no es cierto en general, tal y como ilustra el siguiente ejemplo.

Ejemplo 24 *Sea $\mathcal{R}$ el siguiente programa confluente (y decreciente):*

$$\begin{array}{rcl} f(c(z)) & \rightarrow & z \\ f(h(z)) & \rightarrow & z \\ h(z) & \rightarrow & c(z) \end{array}$$

y $g \equiv (f(c(y)) = 0)$ un objetivo ecuacional. Dado el aplanamiento simple $g' \equiv (c(y) = x, f(x) = 0) \in flat(g)$, existe la siguiente derivación de éxito para g' en $\mathcal{R}$:

$$\begin{array}{rcl} \langle (c(y) = x, \underline{f(x)} = 0), \epsilon \rangle & \leadsto & \langle (c(y) = h(z), \underline{z = 0}), \{x/h(z)\} \rangle \\ & \leadsto & \langle (c(y) = \underline{h(0)}, true), \{x/h(0), z/0\} \rangle \\ & \leadsto & \langle (\underline{c(y) = c(0)}, true), \{x/h(0), z/0\} \rangle \\ & \leadsto & \langle \top, \{x/h(0), z/0, y/0\} \rangle. \end{array}$$

Resulta inmediato comprobar que no es posible construir una refutación para el objetivo original g en $\mathcal{R}$, haciendo uso de las mismas reglas que en la derivación de éxito anterior (sobre las posiciones correspondientes), ya que no es posible aplicar la segunda regla del programa $\mathcal{R}$ en ninguna derivación que parte de g.

El siguiente lema establece la requerida propiedad de completitud fuerte para los objetivos aplanados.

Lema 6.3.8 *Sea $\mathcal{R}$ un programa y $\mathcal{S}$ una función de selección. Sea g un objetivo y $g' \equiv (g_{|u} = x, g[x]_u) \in flat(g)$. Entonces, para toda derivacion de narrowing condicional de éxito $\mathcal{D}' \equiv (\langle g', \epsilon \rangle \leadsto^* \langle \top, \theta' \rangle)$ en la que los términos introducidos por la instanciación de la variable x no han sido reducidos por narrowing con posterioridad a la instanciación y $\theta'_{\lceil Var(g)}$ está normalizada, existe una derivación $\mathcal{D}'_{\mathcal{S}} \equiv (\langle g', \epsilon \rangle \leadsto^* \langle \top, \theta' \rangle)$ que respeta $\mathcal{S}$.*

Demostración. El resultado se prueba, de forma inmediata, siguiendo las líneas de la demostración de la completitud fuerte del *narrowing* ordinario respecto a respuestas normalizadas [OMI95], explotando el hecho de que el término asociado a la variable nueva x en θ' no se reduce por *narrowing* en $\mathcal{D}'$. □

El siguiente lema establece la dirección contraria del Lema 6.3.7, es decir, dada una derivación de éxito para el objetivo aplanado $g' \equiv (g_{|u} = x, g[x]_u) \in flat(g)$ en la que los términos introducidos por la instanciación de la variable x no han sido reducidos por *narrowing* con posterioridad a la instanciación y $\theta'_{\lceil Var(g)}$ está normalizada, entonces dicha respuesta se computa también para el objetivo original g.

Lema 6.3.9 *Sea $\mathcal{R}$ un programa, g un objetivo y $g' \in flat(g)$. Entonces, para toda derivacion de narrowing condicional de éxito $\mathcal{D}' \equiv (\langle g', \sigma\rangle \leadsto^* \langle \top, \sigma\theta'\rangle)$ en la que los términos introducidos por la instanciación de la variable de $Var(g') - Var(g)$ no han sido reducidos por narrowing con posterioridad a la instanciación y $\theta'_{\lceil Var(g)}$ está normalizada, existe una derivación $\mathcal{D} \equiv (\langle g, \sigma\rangle \leadsto^* \langle \top, \sigma\theta\rangle)$ tal que $\theta = \theta'[Var(g)]$ y $\mathcal{D}$ emplea las mismas reglas que $\mathcal{D}'$ (sobre las posiciones correspondientes).*

Demostración. El resultado se sigue directamente del Lema 6.3.8, que establece la completitud fuerte del cálculo de *narrowing* condicional para objetivos aplanados bajo las condiciones impuestas por el lema.

Sea $g' \equiv (g_{|u} = x, g[x]_u)$, con $u \in \check{O}(g)$ y x una variable nueva. Por el Lema 6.3.8, para cualquier función de selección $\mathcal{S}$ existe una derivación de *narrowing* condicional de éxito $(\langle g', \sigma\rangle \leadsto^* \langle \top, \sigma\theta'\rangle)$ que respeta $\mathcal{S}$. Sin pérdida de generalidad, consideramos la siguiente derivación para $\langle g', \sigma\rangle$, que computa la sustitución θ' de acuerdo a una regla que selecciona las ecuaciones de izquierda a derecha:

$$\langle g', \sigma\rangle \equiv \langle g'_0, \sigma\rangle \leadsto_{[w_1, r_1]} \langle g'_1, \sigma\sigma_1\rangle \leadsto_{[w_2, r_2]} \ldots \leadsto_{[w_{n-1}, r_{n-1}]} \langle g'_n, \sigma\sigma_1 \ldots \sigma_n\rangle \equiv \langle \top, \sigma\theta'\rangle$$

con $\theta' = \sigma_1 \ldots \sigma_n$, donde $r_i \equiv (\lambda_i \to \rho_i \Leftarrow C_i) \ll \mathcal{R}_+$, $w_i \in \bar{O}(g_{i-1})$, y $\sigma_i = mgu(\{g'_{i-1|w_i} = \lambda_i\})$, $i = 1, \ldots, n$, $n \geq 2$. Nótese que n es mayor o igual que 2, ya que el número de ecuaciones en el objetivo aplanado g' es mayor o igual que 2. La prueba se realiza por inducción sobre el número n de pasos de la derivación.

Sea $n = 2$. En este caso, $w_1 \equiv 1$, $w_2 \equiv 2$, y la regla de unificación sintáctica $(x = x \to true)$ ha sido aplicada dos veces. Entonces, $\langle g'_0, \sigma\rangle \equiv \langle (g_{|u} = x, g[x]_u), \sigma\rangle \leadsto \langle g, \sigma\sigma_1\rangle \leadsto \langle \top, \sigma\sigma_1\sigma_2\rangle$, donde $\sigma_1 = \{x/(g_{|u})\}$ y $\sigma_2 = mgu(g) = \theta'_{\lceil Var(g)}$ y, por tanto, se cumple el resultado enunciado.

Consideremos ahora el caso inductivo $n > 2$. Tenemos dos posibilidades:

- Sea $w_1 \equiv 1$. Entonces $\langle g'_0, \sigma\rangle \equiv \langle g_{|u} = x, g[x]_u, \sigma\rangle \leadsto_{[1, x=x \to true]} \langle g, \sigma\{x/g_{|u}\}\rangle$, y el resultado se cumple ya que $\{x/g_{|u}\}_{\lceil Var(g)} = \epsilon$.

- Sea $w_1 \equiv 1.1.w'$, con $w' \in \bar{O}(g_{|u})$. Entonces $g'_1 \equiv (C_1, g_{|u}[\rho_1]_{w'} = x, g[x]_u)\sigma_1$. Ya que $w' \in \bar{O}(g_{|u})$, entonces $\langle g, \sigma\rangle \leadsto_{[u.w', r_1]} \langle (C_1, g[\rho_1]_{u.w'})\sigma_1, \sigma\sigma_1\rangle \equiv \langle g_1, \sigma\sigma_1\rangle$, y $g'_1 \in flat(g_1)$. Sea $\mathcal{D}'' \equiv (\langle g'_1, \sigma\sigma_1\rangle \leadsto^* \langle \top, \sigma\sigma_1 \ldots \sigma_n\rangle)$, con $(\sigma_2 \ldots \sigma_n)_{\lceil Var(g'_1)}$ $\equiv ((\sigma_2 \ldots \sigma_n)_{\lceil Var(g_1)} \cup \{x/t\})$. Entonces, $(\sigma_2 \ldots \sigma_n)_{\lceil Var(g_1)}$ está normalizada y el término t no se ha reducido por *narrowing* en $\mathcal{D}''$ con posterioridad a la aplicación del enlace $\{x/t\}$. Por la hipótesis de inducción, se cumple que $\langle g_1, \sigma\sigma_1\rangle \leadsto^*$ $\langle \top, \sigma\sigma_1\vartheta\rangle$, con $\vartheta = (\sigma_2 \ldots \sigma_n)[Var(g_1)]$ y, por tanto, $\langle g, \sigma\rangle \leadsto^* \langle \top, \sigma\theta\rangle$, con $\theta = (\sigma_1 \ldots \sigma_n)[Var(g)]$.

□

Dado un objetivo ecuacional g y un término s más general que algún subtérmino $s\gamma$ de g, definimos un s-aplanamiento de g como un nuevo objetivo que contiene la ecuación $(s = x)$ y puede construirse a partir de g aplicando una secuencia (finita) de aplanamientos simples.

Definición 6.3.10 (s-aplanamiento) *Sea g un objetivo ecuacional y s un término. Definimos la función $flat : Goal \times \mathcal{T} \rightarrow \wp Goal$ como sigue:*

$$flat(g, s) = \{g' \mid g' \equiv (s = x, \widehat{\gamma}, g[x]_u),\ u \in \breve{O}(g),\ g_{|u} = s\gamma\}.$$

El siguiente lema establece la equivalencia entre un objetivo g y cualquiera de sus s-aplanamientos respecto a los conjuntos de respuestas normalizadas computadas.

Lema 6.3.11 *Sea g un objetivo ecuacional, s un término y $g' \in flat(g, s)$. Entonces, $\mathcal{D} \equiv (\langle g, \sigma\rangle \leadsto^* \langle \top, \sigma\theta\rangle)$ sii existe una derivación de éxito $\mathcal{D}' \equiv (\langle g', \sigma\rangle \leadsto^* \langle \top, \sigma\theta'\rangle)$ en la que los términos introducidos por la instanciación de las variables de $Var(g') - Var(g)$ no han sido reducidos por narrowing en $\mathcal{D}'$ con posterioridad a la instanciación, $\theta'_{\lceil Var(g)}$ está normalizada, $\theta = \theta'[Var(g)]$ y $\mathcal{D}'$ emplea las mismas reglas que $\mathcal{D}$ (sobre las posiciones correspondientes), más un número finito de aplicaciones extra de la regla $(x = x \rightarrow true)$.*

Demostración. Inmediata a partir de los Lemas 6.3.7 y 6.3.9, ya que g' puede obtenerse a partir de g aplicando una secuencia finita de aplanamientos simples. □

Intuitivamente, el interés de construir objetivos s-aplanados es facilitar la tarea de reordenar las derivaciones que parten de un objetivo dado, para imitar la secuencia de pasos que se aplican para construir los resultantes para los términos parcialmente evaluados. De esta forma, se pueden reemplazar algunos de dichos pasos en la derivación original por un único paso, obtenido por la aplicación del resultante correspondiente. El siguiente resultado es auxiliar.

Lema 6.3.12 *Sean $\mathcal{R}$ un programa, g un objetivo ecuacional y $g' \equiv (s = x, \widehat{\gamma}, g[x]_p) \in flat(g, s)$ tales que existe una derivación de la forma:*

$$\mathcal{D}' \equiv \langle g', \sigma\rangle \leadsto^n \langle (C, t = x, \widehat{\gamma}, g[x]_p)\vartheta', \sigma\vartheta'\rangle \leadsto^* \langle \top, \sigma\vartheta'\delta'\rangle \equiv \langle \top, \sigma\theta'\rangle$$

en la que los términos introducidos por la instanciación de las variables de $Var(g') - Var(g)$ no han sido reducidos por narrowing con posterioridad a la instanciación, $\theta'_{\lceil Var(g)}$ está normalizada, y las ecuaciones de $\widehat{\gamma}$ se reducen en $\mathcal{D}'$ haciendo uso únicamente de la regla $(x = x \rightarrow true)$. Entonces, existe una derivación para g en $\mathcal{R}$ de la forma:

$$\langle g, \sigma\rangle \leadsto^n \langle h, \sigma\vartheta\rangle \leadsto^* \langle \top, \sigma\vartheta\delta\rangle \equiv \langle \top, \sigma\theta\rangle$$

tal que $\theta = \theta'[Var(g)]$, $(C, t = x, g[x]_p)\vartheta' mgu(\widehat{\gamma}\vartheta') \in flat(h)$ *y la subderivación* $\langle g, \sigma\rangle \leadsto^n \langle h, \sigma\vartheta\rangle$ *emplea las mismas reglas y en el mismo orden que los primeros* n *pasos de la derivación* $\mathcal{D}'$ *(en las posiciones correspondientes).*

DEMOSTRACIÓN. Por el Lema 6.3.8, podemos considerar (sin pérdida de generalidad) que la derivación $\mathcal{D}'$ tiene la forma:

$$\begin{array}{rl} \langle g', \sigma\rangle \equiv \langle (s = x, \widehat{\gamma}, g[x]_p), \sigma\rangle & \leadsto \langle (C_1, s_1 = x, \widehat{\gamma}, g[x]_p)\vartheta'_1, \sigma\vartheta'_1\rangle \\ & \leadsto \ldots \\ & \leadsto \langle (C_n, s_n = x, \widehat{\gamma}, g[x]_p)\vartheta'_n, \sigma\vartheta'_n\rangle \\ & \leadsto^* \langle (C_n, s_n = x, g[x]_p)\vartheta'_n mgu(\widehat{\gamma}\vartheta'_n), \sigma\vartheta'_n mgu(\widehat{\gamma}\vartheta'_n)\rangle \\ & \leadsto^* \langle \top, \sigma\theta'\rangle,\ n \geq 1 \end{array}$$

donde los pasos de *narrowing*:

$$\langle (C_n, s_n = x, \widehat{\gamma}, g[x]_p)\vartheta'_n, \sigma\vartheta'_n\rangle \leadsto^* \langle (C_n, s_n = x, g[x]_p)\vartheta'_n mgu(\widehat{\gamma}\vartheta'_n), \sigma\vartheta'_n mgu(\widehat{\gamma}\vartheta'_n)\rangle$$

sólo emplean la regla de unificación sintáctica.

En estas condiciones, para probar el lema es suficiente con verificar que, para toda derivación de la forma:

$$\begin{array}{rl} \langle g', \sigma\rangle \equiv \langle (s = x, \widehat{\gamma}, g[x]_p), \sigma\rangle & \leadsto \langle (C_1, s_1 = x, \widehat{\gamma}, g[x]_p)\vartheta'_1, \sigma\vartheta'_1\rangle \\ & \leadsto \ldots \\ & \leadsto \langle (C_n, s_n = x, \widehat{\gamma}, g[x]_p)\vartheta'_n, \sigma\vartheta'_n\rangle \end{array}$$

existe una derivación:

$$\langle g, \sigma\rangle \leadsto \langle g_1, \sigma\vartheta_1\rangle \leadsto \ldots \leadsto \langle g_n, \sigma\vartheta_n\rangle$$

que emplea las mismas reglas y en el mismo orden que en la derivación anterior (sobre las posiciones correspondientes) y tal que $\vartheta'_n mgu(\widehat{\gamma}\vartheta'_n) = \gamma\vartheta_n$ (con lo que $\vartheta'_n mgu(\widehat{\gamma}\vartheta'_n) = \vartheta_n[Var(g)]$, ya que $Dom(\gamma) \cap Var(g) = \emptyset$), y

$$(C_n, s_n = x, g[x]_p)\vartheta'_n mgu(\widehat{\gamma}\vartheta'_n) \in flat(g_n)$$

haciendo uso en g_n de la ocurrencia p de g para realizar el aplanamento simple. Entonces, ya que $(C_n, s_n = x, g[x]_p)\vartheta'_n mgu(\widehat{\gamma}\vartheta'_n) \in flat(g_n)$ y

$$\langle (C_n, s_n = x, g[x]_p)\vartheta'_n mgu(\widehat{\gamma}\vartheta'_n), \sigma\vartheta'_n mgu(\widehat{\gamma}\vartheta'_n)\rangle \leadsto^* \langle \top, \sigma\theta'\rangle$$

se cumple, por el Lema 6.3.9, que $\langle g_n, \sigma\vartheta_n\rangle \leadsto^* \langle \top, \theta\rangle$ con $\theta = \theta'[Var(g)]$.

Realizamos la demostración del anterior argumento por inducción sobre la longitud n de las derivaciones.

Sea $n = 1$. Dada la derivación:

$$\langle g', \sigma\rangle \equiv \langle (s = x, \widehat{\gamma}, g[x]_p), \sigma\rangle \leadsto \langle (C, s[\rho]_u = x, \widehat{\gamma}, g[x]_p)\vartheta', \sigma\vartheta'\rangle$$

donde $g_{|p} = s\gamma$, existen $(\lambda \to \rho \Leftarrow C) \ll \mathcal{R}$, $u \in \bar{O}(s)$ y $\vartheta' = mgu(\{s_{|u} = \lambda\}) \not\equiv fail$. Puesto que $g' \in flat(g, s)$, entonces $g \equiv g[s\gamma]_p$ y se cumplen las siguientes equivalencias:

$$\begin{array}{rcll}
\vartheta' mgu(\widehat{\gamma}\vartheta') & = & \vartheta' \Uparrow \gamma & \text{(por el Lema 3.2.1)} \\
& = & \gamma mgu(\widehat{\vartheta'}\gamma) & \text{(por el Lema 3.2.1)} \\
& = & \gamma mgu(\widehat{mgu}(\{s_{|u} = \lambda\})\gamma) & \\
& = & \gamma mgu(\{s_{|u} = \lambda\}\gamma) & \\
& = & \gamma mgu(\{s\gamma_{|u} = \lambda\}) & \text{(ya que } Var(\lambda) \cap Dom(\gamma) = \emptyset) \\
& = & \gamma\vartheta &
\end{array}$$

donde $\vartheta = mgu(\{s\gamma_{|u} = \lambda\})$. Ahora, puesto que $\mathcal{D}'$ es una derivación de éxito y las ecuaciones de $\widehat{\gamma}$ se reducen con la regla $(x = x \to true)$, entonces $\vartheta' mgu(\widehat{\gamma}\vartheta') \not\equiv fail$ y, por tanto, $\vartheta \not\equiv fail$. Así, existe la derivación:

$$\langle g, \sigma\rangle \leadsto \langle (C, g[s\gamma[\rho]_u]_p)\vartheta, \sigma\vartheta\rangle$$

que emplea la misma regla que la transición para g' (sobre la posición correspondiente $p.u$). Ahora, dado que $(C, s[\rho]_u = x, g[x]_p)\vartheta' mgu(\widehat{\gamma}\vartheta') = (C, s[\rho]_u = x, g[x]_p)\gamma\vartheta = (C, s\gamma[\rho]_u = x, g[x]_p)\vartheta$, resulta inmediato verificar que $(C, s\gamma[\rho]_u = x, g[x]_p)\vartheta \in flat((C, g[s\gamma[\rho]_u]_p)\vartheta)$, haciendo uso del subtérmino a la ocurrencia p para realizar el aplanamiento simple.

Consideremos el caso inductivo $n > 1$. Por la hipótesis de inducción, dada la derivación:

$$\langle g', \sigma\rangle \equiv \langle (s = x, \widehat{\gamma}, g[x]_p), \sigma\rangle \leadsto \ldots \leadsto \langle (C_{n-1}, s_{n-1} = x, \widehat{\gamma}, g[x]_p)\vartheta'_{n-1}, \sigma\vartheta'_{n-1}\rangle$$

existe la secuencia de pasos de *narrowing*:

$$\langle g, \sigma\rangle \leadsto \ldots \leadsto \langle g_{n-1}, \sigma\vartheta_{n-1}\rangle$$

en la que se han empleado las mismas reglas y en el mismo orden que en la derivación anterior (sobre las posiciones correspondientes) y tal que $\vartheta'_{n-1} mgu(\widehat{\gamma}\vartheta'_{n-1}) = \gamma\vartheta_{n-1}$ y $(C_{n-1}, s_{n-1} = x, g[x]_p)\vartheta'_{n-1} mgu(\widehat{\gamma}\vartheta'_{n-1}) \in flat(g_{n-1})$, haciendo uso en g_n de la ocurrencia p de g para realizar el aplanamiento simple. Ahora, consideremos la siguiente transición[4]:

$$\langle (C_{n-1}, s_{n-1} = x, \widehat{\gamma}, g[x]_p)\vartheta'_{n-1}, \sigma\vartheta'_{n-1}\rangle \leadsto \langle (C_n, s_n = x, \widehat{\gamma}, g[x]_p)\vartheta'_n, \sigma\vartheta'_n\rangle,$$

[4]El caso en el que la ocurrencia reducida pertenece a C_{n-1} es inmediato.

con $(\lambda \to \rho \Leftarrow C) \ll \mathcal{R}$, $u \in \bar{O}(s_{n-1})$ y $\vartheta' = mgu(\{(s_{n-1}\vartheta'_{n-1})_{|u} = \lambda\}) \not\equiv fail$, y donde $C_n = (C, C_{n-1})$, $s_n = s_{n-1}[\rho]_u$ y $\vartheta'_n = \vartheta'_{n-1}\vartheta'$. De forma similar al caso base, es posible demostrar las siguientes equivalencias:

$$
\begin{array}{lll}
\vartheta'_n mgu(\widehat{\gamma}\vartheta'_n) & = & \\
\gamma \Uparrow \vartheta'_n & = & \text{(por el Lema 3.2.1)} \\
\gamma \Uparrow (\vartheta'_{n-1}\vartheta') & = & \\
\gamma \Uparrow (\vartheta'_{n-1} mgu(\{(s_{n-1}\vartheta'_{n-1})_{|u} = \lambda\})) & = & \\
\gamma \Uparrow (\vartheta'_{n-1} mgu(\widehat{mgu}(\{(s_{n-1}\vartheta'_{n-1})_{|u} = \lambda\}))) & = & \\
\gamma \Uparrow (\vartheta'_{n-1} mgu(\widehat{mgu}(\{s_{n-1|u} = \lambda\})\vartheta'_{n-1})) & = & \text{(ya que } Var(\lambda)\cap \\
 & & \quad Dom(\vartheta'_{n-1}) = \emptyset) \\
\gamma \Uparrow (\vartheta'_{n-1} \Uparrow mgu(\{s_{n-1|u} = \lambda\})) & = & \text{(por el Lema 3.2.1)} \\
\vartheta'_{n-1} \Uparrow \gamma \Uparrow mgu(\{s_{n-1|u} = \lambda\}) & = & \text{(por la conmut. y asoc. de } \Uparrow) \\
(\vartheta'_{n-1} mgu(\widehat{\gamma}\vartheta'_{n-1})) \Uparrow mgu(\{s_{n-1|u} = \lambda\}) & = & \text{(por el Lema 3.2.1)} \\
(\gamma\vartheta_{n-1}) \Uparrow mgu(\{s_{n-1|u} = \lambda\}) & = & \text{(por la hipótesis de inducción)} \\
\gamma\vartheta_{n-1} mgu(\widehat{mgu}(\{s_{n-1|u} = \lambda\})\gamma\vartheta_{n-1}) & = & \text{(por el Lema 3.2.1)} \\
\gamma\vartheta_{n-1} mgu(\{s_{n-1|u} = \lambda\}\gamma\vartheta_{n-1}) & = & \\
\gamma\vartheta_{n-1} mgu(\{(s_{n-1}\gamma\vartheta_{n-1})_{|u} = \lambda\}) & = & \text{(ya que } Var(\lambda)\cap \\
 & & \quad Dom(\gamma\vartheta_{n-1}) = \emptyset) \\
\gamma\vartheta_{n-1}\vartheta & = & \\
\gamma\vartheta_n & &
\end{array}
$$

donde $\vartheta = mgu(\{(s_{n-1}\gamma\vartheta_{n-1})_{|u} = \lambda\})$. Por la hipótesis de inducción, $(C_{n-1}, s_{n-1} = x, g[x]_p)\vartheta'_{n-1} mgu(\widehat{\gamma}\vartheta'_{n-1}) \in flat(g_{n-1})$, haciendo uso en g_{n-1} de la ocurrencia p de g para realizar el aplanamiento simple. Por tanto, $g_{n-1} = (C_{n-1}, g[s_{n-1}]_p)\vartheta'_{n-1}$ $mgu(\widehat{\gamma}\vartheta'_{n-1}) = (C_{n-1}, g_{n-1}[s_{n-1}]_p)\gamma\vartheta_{n-1} = (C_{n-1}, g_{n-1}[s_{n-1}\gamma]_p)\vartheta_{n-1}$, y podemos probar la transición:

$$
\begin{array}{ll}
\langle g_{n-1}, \sigma\vartheta_{n-1}\rangle & \equiv \langle (C_{n-1}, g_{n-1}[s_{n-1}\gamma]_p)\vartheta_{n-1}, \sigma\vartheta_{n-1}\rangle \\
 & \leadsto \langle (C_n, g_{n-1}[s_n\gamma]_p)\vartheta_n, \sigma\vartheta_n\rangle \equiv \langle g_n, \sigma\vartheta_n\rangle
\end{array}
$$

donde $C_n = (C, C_{n-1})$, $s_n = s_{n-1}[\rho]_u$, y $\vartheta_{n-1}\vartheta = \vartheta_n$. Puesto que $\vartheta'_n mgu(\widehat{\gamma}\vartheta'_n) = \gamma\vartheta_n$, se cumple, de forma análoga al caso base, $(C_n, s_n = x, g[x]_p)\vartheta'_n mgu(\widehat{\gamma}\vartheta'_n) = (C_n, s_n = x, g[x]_p)\gamma\vartheta_n = (C_n, s_n\gamma = x, g[x]_p)\vartheta_n \in flat(g_n)$, lo que completa la demostración. □

El siguiente lema es crucial para la demostración de la completitud de nuestra transformación y establece, esencialmente, que toda derivación de éxito $\mathcal{D}$ en el programa original (para un objetivo g que es S-cerrado), se puede reordenar de forma que se apliquen los pasos de *narrowing* en el mismo orden que en las derivaciones que han generado algunos de los resultantes del programa transformado. La forma de realizar esta reordenación es fundamental ya que, en general, no es posible aplicar los pasos de *narrowing* empleados en la construcción de un resultante para el término s sobre un subtérmino $s\gamma$ de g. El motivo es que, si los subtérminos que aparecen en la sustitución γ deben reducirse en la derivación para el objetivo g, entonces es posible que sin la reducción previa de dichos términos no se pueda imitar la secuencia de pasos considerada. Intuitivamente, podemos garantizar que los pasos realizados para construir un resultante para el término s se pueden aplicar sobre el subtérmino $g_{|p}$ de g cuando se cumplen las siguientes condiciones: 1) $g_{|p}$ es una instancia de algún término de S, i.e. $g_{|p} = s\gamma$, $s \in S$; 2) alguno de los términos de $g_{|p}$ se reduce por *narrowing* en $\mathcal{D}$ usando, al menos, una regla distinta de $(x = x \rightarrow true)$; y 3) los términos de $\widehat{\gamma}$ no se reducen por *narrowing* en la derivación $\mathcal{D}$. De alguna forma, las anteriores condiciones determinan un subtérmino de g que se puede considerar, respecto a la derivación $\mathcal{D}$, un término "innermost". Esto no significa que los términos de $\widehat{\gamma}$ deban ser necesariamente constructores. Simplemente no deben haber sido reducidos por *narrowing* en la derivación $\mathcal{D}$. De este modo, las tres condiciones impuestas sobre el subtérmino $g_{|p}$ de g, garantizan que efectivamente se pueden reordenar los pasos de la derivación $\mathcal{D}$, dando lugar a una nueva derivación $\mathcal{D}'$, en la que los pasos de *narrowing* que se realizan en $\mathcal{D}$ sobre el subtérmino $s\gamma$ se pueden realizar, de forma consecutiva, al inicio de la derivación $\mathcal{D}'$. El objetivo final de esta reordenación es, lógicamente, obtener una derivación para el objetivo g que siga la misma secuencia de pasos de *narrowing* que la derivación que ha generado alguno de los resultantes del programa especializado. Como resultado preparatorio, el siguiente lema establece la equivalencia entre una derivación a partir del objetivo g y cualquier reordenación de la misma en la que se reduzcan los subtérminos de g (que permiten demostrar que el objetivo es S-cerrado) de forma "innermost".

Lema 6.3.13 *Sea $\mathcal{R}$ un programa, g un objetivo ecuacional y S un conjunto de términos. Sea $\mathcal{R}'$ una EP de S en $\mathcal{R}$ tal que $\mathcal{R}' \cup \{g\}$ es S-cerrado. Sea $\mathcal{D} \equiv (\langle g, \sigma\rangle \leadsto^* \langle \top, \sigma\theta\rangle)$ una derivación de éxito para g en $\mathcal{R}$ tal que $\theta_{\lceil Var(g)}$ está normalizada. Sea $W = \{w_1, \ldots, w_k\} \in CSet(S, g)$ un conjunto de ocurrencias de cierre para g con respecto a S y $O = \{u_1, \ldots, u_n\} \subseteq W$ el subconjunto de las posiciones de W que señalan subtérminos de g que se reducen en $\mathcal{D}$. Entonces, existe una derivación $\mathcal{D}' \equiv (\langle g, \sigma\rangle \leadsto^* \langle \top, \sigma\theta'\rangle)$ tal que $\theta = \theta'[Var(g)]$ y los subtérminos apuntados por las ocurrencias de O se explotan dando prioridad a los más internos.*

Demostración. Consideramos, sin pérdida de generalidad, que las ocurrencias del conjunto O verifican que para todo $i, j \in \{1, \ldots, n\}$. $(i < j \Rightarrow u_i \not\leq u_j)$. Realizamos la prueba por inducción sobre el número n de elementos de O.

El caso base $n = 0$ es trivial, ya que entonces sólo se ha aplicado la regla $(x = x \rightarrow true)$ en la derivación $\mathcal{D}$ y, por tanto, basta con considerar $\mathcal{D} \equiv \mathcal{D}'$.

Sea $n > 0$. Consideramos la ocurrencia $u \equiv u_1$, donde $g_{|u} = s\gamma$, $s \in S$. Por el orden impuesto sobre el conjunto O, se cumple de manera inmediata que los términos de $\widehat{\gamma}$ no se reducen en $\mathcal{D}$.
Sea $g' \equiv (s = x, \widehat{\gamma}, g[x]_u) \in flat(g, s)$ un s-aplanamiento del objetivo g. Por el Lema 6.3.11, existe una derivación de éxito para g' en $\mathcal{R}$ que computa una solución θ'', tal que $\theta''_{\lceil Var(g)}$ está normalizada, los términos introducidos por la instanciación de las variables de $Var(g') - Var(g)$ no han sido reducidos por *narrowing* con posterioridad a la instanciación y $\theta'' = \theta[Var(g)]$. Por el Lema 6.3.8, podemos restringirnos, sin pérdida de generalidad, a derivaciones de la forma:

$$\begin{array}{rl} \mathcal{D}'' \equiv \langle g', \sigma \rangle & \leadsto^k \overbrace{\langle (\top, t = x, \widehat{\gamma}, g[x]_u)\vartheta', \sigma\vartheta' \rangle \leadsto^* \langle (\top, t = x, \top, g[x]_u)\vartheta'\delta', \sigma\vartheta'\delta' \rangle}^{\mathcal{D}''_1} \\ & \leadsto \underbrace{\langle (\top, g[t]_u)\vartheta'\delta'\{x/t\}, \sigma\vartheta'\delta'\{x/t\} \rangle \leadsto^* \langle \top, \sigma\theta'' \rangle}_{\mathcal{D}''_2} \end{array}$$

donde en $\mathcal{D}''_1$ sólo se emplea la regla de unificación sintáctica (es decir, $\delta' = mgu(\widehat{\gamma}\vartheta')$) y en $\mathcal{D}''_2$ no se reduce el término t.
Ahora, aplicando el Lema 6.3.12, existe la siguiente derivación para g:

$$\mathcal{D}' \equiv \langle g, \sigma \rangle \leadsto^k \langle (g[t]_u)\vartheta, \sigma\vartheta \rangle \leadsto^* \langle \top, \sigma\theta' \rangle$$

tal que $\theta' = \theta''[Var(g)]$ y emplea en los primeros k pasos exactamente las mismas reglas y en el mismo orden que los primeros k pasos de la derivación $\mathcal{D}''$.
Finalmente, para aplicar la hipótesis de inducción sobre la derivación $\langle (g[t]_u)\vartheta, \sigma\vartheta \rangle \leadsto^* \langle \top, \sigma\theta' \rangle$, debemos verificar que el número de posiciones reducibles del objetivo ha disminuido tras los primeros k pasos de la derivación $\mathcal{D}'$. En primer lugar, $g[x]_u$ es S-cerrado haciendo uso del conjunto de ocurrencias $O' = O - \{u_1\}$. Por construcción de la derivación $\mathcal{D}''$, el término t no se reduce en $\mathcal{D}'$ y, dado que $\theta'_{\lceil Var(g)}$ está normalizada, los términos introducidos por la aplicación de ϑ en $g[t]_u$ tampoco deben reducirse en $\mathcal{D}'$. En estas condiciones, las posiciones que deben reducirse en $\mathcal{D}'$ del objetivo $(g[t]_u)\vartheta$ son los $n - 1$ subtérminos apuntados por los elementos de O'. El resultado se sigue, por tanto, aplicando la inducción sobre la subderivación $\langle (g[t]_u)\vartheta, \sigma\vartheta \rangle \leadsto^* \langle \top, \sigma\theta' \rangle$. □

El siguiente lema establece una propiedad básica de los resultantes. Concretamente, establece la equivalencia entre una derivación que emplea una secuencia de pasos de *narrowing*, y otra en la que se utiliza el resultante asociado a dicha derivación.

Lema 6.3.14 (resultante) *Sea $\mathcal{R}$ un programa y $r \equiv (s\theta \to t \Leftarrow C)$ un resultante para s en $\mathcal{R}$, originado por la siguiente derivación:*

$$\mathcal{D}_s \equiv \langle s = y, \epsilon \rangle \leadsto^+ \langle (C, t = y), \theta \rangle.$$

Sea g un objetivo ecuacional, tal que $g_{|u} = s\gamma$, para el que existe la derivación:

$$\langle g, \sigma\rangle \leadsto^{+} \langle g', \sigma'\rangle$$

que utiliza las mismas reglas que la derivación $\mathcal{D}_s$, en el mismo orden y sobre las posiciones correspondientes. Entonces, se cumple que $mgu(\{s\gamma = s\theta\}) \not\equiv fail$ y es posible probar el paso de narrowing:

$$\langle g, \sigma\rangle \leadsto \langle g', \sigma''\rangle$$

empleando el resultante r, de forma que $\sigma' = \sigma''[Var(g)]$.

Demostración. La prueba es estándar. Ver, por ejemplo, la técnica empleada en el Lema 4.12 de [LS91] que demuestra un resultado análogo para el caso de los programas lógicos. □

Finalmente, podemos enunciar y demostrar uno de los resultados principales de este capítulo: la corrección y completitud de la EP de los programas lógico-funcionales usando *narrowing* condicional. En esta sección, consideramos que la semántica operacional $\mathcal{O}_{\mathcal{R}}$ sólo observa las sustituciones de respuesta computada normalizadas, es decir, dado un programa $\mathcal{R}$ y un objetivo ecuacional g, la semántica operacional de g en $\mathcal{R}$ se define como:

$$\mathcal{O}_{\mathcal{R}}(g) = \{\theta_{\lceil Var(g)} \mid \langle g, \epsilon\rangle \leadsto^{*} \langle \top, \theta\rangle,\ \theta_{\lceil Var(g)} \text{ está normalizada }\}.$$

Teorema 6.3.15 (corrección y completitud de la EP)
Sea $\mathcal{R}$ un programa, g un objetivo, S un conjunto finito de términos y $\mathcal{R}'$ una EP de $\mathcal{R}$ con respecto a S. Entonces, se cumple:

1. *(Corrección)* $\theta \in \mathcal{O}_{\mathcal{R}'}(g) \Rightarrow \exists\gamma \in \mathcal{O}_{\mathcal{R}}(g)$ *tal que* $\gamma \leq \theta[Var(g)]$.
2. *(Completitud fuerte)* $\mathcal{O}_{\mathcal{R}'}(g) \supseteq \mathcal{O}_{\mathcal{R}}(g)$, *si* $\mathcal{R}' \cup \{g\}$ *es* S*-cerrado.*

Demostración.
(Corrección). Denotamos por $\mathcal{E}_{\mathcal{R}}$ la teoría ecuacional presentada por $\mathcal{R}$. La corrección del cálculo de *narrowing* condicional implica que, para todo resultante $r \in \mathcal{R}'$, se cumple $\mathcal{E}_{\mathcal{R}} \models \mathcal{E}_{\{r\}}$. Asímismo, si θ es una sustitución de respuesta computada normalizada para g en $\mathcal{R}'$, por la corrección del cálculo de *narrowing* condicional tenemos que $\mathcal{E}_{\mathcal{R}'} \models g\theta$ y, por tanto, $\mathcal{E}_{\mathcal{R}} \models \mathcal{E}_{\mathcal{R}'} \models g\theta$, lo que significa que θ es un $\mathcal{E}$-unificador de g en $\mathcal{E}_{\mathcal{R}}$. Ahora, ya que *narrowing* condicional es un algoritmo completo de $\mathcal{E}$-unificación para programas confluentes (respecto a sustituciones normalizadas), existe una derivación de *narrowing* condicional ($\langle g, \epsilon\rangle \leadsto^{*} \langle \top, \gamma\rangle$) en $\mathcal{R}$ tal que $\gamma \leq \theta[Var(g)]$ y, por tanto, $\gamma \in \mathcal{O}_{\mathcal{R}}(g)$.

(Completitud fuerte). Vamos a probar que, para toda derivación de éxito de *narrowing* condicional $(\langle g_0, \sigma\rangle \rightsquigarrow^* \langle\top, \sigma\theta\rangle)$ en $\mathcal{R}$, tal que $\theta_{\lceil Var(g_0)}$ está normalizada, existe una derivación de éxito correspondiente $(\langle g_0, \sigma\rangle \rightsquigarrow^* \langle\top, \sigma\theta''\rangle)$ para g_0 en $\mathcal{R}'$ tal que $\theta = \theta''[Var(g_0)]$.
Consideremos la derivación:

$$\mathcal{D} \equiv (\langle g_0, \sigma\rangle \rightsquigarrow \langle g_1, \sigma\theta_1\rangle \rightsquigarrow \ldots \rightsquigarrow \langle g_m, \sigma\theta_1 \ldots \theta_m\rangle \equiv \langle\top, \sigma\theta\rangle)$$

con $\theta = \theta_1 \ldots \theta_m$, donde $(\lambda_i \rightarrow \rho_i \Leftarrow C_i) \ll \mathcal{R}_+$, $\theta_i = mgu(\{g_{i-1|u_i} = \lambda_i\})$, $g_{i+1} \equiv (C_i, g_i[\rho_i]_{u_i})\theta_i$, $i \geq 0$, y el objetivo inicial g_0 es S-cerrado.
En primer lugar, por el Lema 6.3.13, sabemos que es posible construir una derivación de éxito para g_0 que computa la misma respuesta que la derivación original y reduce uno de los conjuntos de subtérminos que permiten demostrar que g_0 es S-cerrado, siguiendo un orden "innermost". Este hecho garantiza que es posible aplicar las mismas secuencias de pasos que se han empleado en la construcción de algunos de los resultantes de $\mathcal{R}'$, sin que se produzcan conflictos en el cómputo de los unificadores más generales. Formalmente, ya que g_0 es S-cerrado, existe al menos un conjunto de ocurrencias de cierre $O \in CSet(S, g_0)$ tal que $\forall u \in O.\ \exists s \in S.\ s \leq g_{0|u}$. Por el Lema 6.3.13, existe la siguiente derivación de éxito para g_0 en $\mathcal{R}$:

$$\mathcal{D}' \equiv \langle g_0, \sigma\rangle \rightsquigarrow^* \langle\top, \sigma\theta'\rangle$$

en la que los subtérminos apuntados por O se reducen dando prioridad a los más internos. Realizamos la prueba por inducción sobre la complejidad n de la derivación $\mathcal{D}'$.

Si $n = 0$, entonces sólo se ha empleado en $\mathcal{D}'$ la regla $(x = x \rightarrow true)$ y, por tanto, la misma derivación puede construirse también en $\mathcal{R}'_+$.

Consideremos el caso inductivo $n > 0$. Sea u la ocurrencia del conjunto O tal que $g_{0|u}$ es el primer subtérmino de g_0 que es reducido en $\mathcal{D}'$. Entonces $g_{0|u} = s\gamma$, donde el término s se reduce en $\mathcal{D}'$ empleando al menos una regla distinta de $(x = x \rightarrow true)$ y los subtérminos que aparecen en $\widehat{\gamma}$ no se reducen en la refutación. Consideremos que $\mathcal{D}'$ tiene la forma:

$$\mathcal{D}' \equiv (\langle g_0, \sigma\rangle \equiv \langle g_0[s\gamma]_u, \sigma\rangle \rightsquigarrow_{[u_1, r_1]} \ldots \rightsquigarrow_{[u_k, r_k]} \langle h, \sigma\delta'\rangle \rightsquigarrow^* \langle\top, \sigma\theta'\rangle)$$

Dado que $\mathcal{D}'$ es una derivación de éxito, podemos asegurar que existirá un resultante:

$$r' \equiv (s\vartheta \rightarrow t \Leftarrow C) \in \mathcal{R}'$$

construido a partir de la derivación:

$$\langle s = y, \epsilon\rangle \rightsquigarrow_{[w_1, r_1]} \ldots \rightsquigarrow_{[w_k, r_k]} \langle (C, t = y), \vartheta\rangle, \quad k > 0$$

que aplica exactamente las mismas reglas y en el mismo orden que los primeros k pasos de la derivación $\mathcal{D}'$. Ahora, por el Lema 6.3.14, podemos sustituir los k pasos realizados con las reglas de $\mathcal{R}$ por la aplicación del resultante $r' \equiv (s\vartheta \to t \Leftarrow C) \in \mathcal{R}'$, obteniendo:

$$\langle g_0, \sigma\rangle \equiv \langle g_0[s\gamma]_u, \sigma\rangle \leadsto_{[u,r']} \langle (C, g_0[t]_u)\delta'', \sigma\delta''\rangle \leadsto^* \langle \top, \sigma\theta''\rangle$$

donde $\delta'' = mgu(\{s\gamma = s\vartheta\}) \not\equiv fail$, $\delta' = \delta''[Var(g_0)]$, $h = (C, g_0[t]_u)\delta''$ y $\theta' = \theta''[Var(g_0)]$. Ahora, puesto que los términos de $terms(r')$ son S-cerrados, entonces t y C son S-cerrados. Además, si $g_0[s\gamma]_u$ es S-cerrado con el conjunto de ocurrencias de cierre O, entonces $g_0[x]_u$ también se puede demostrar S-cerrado usando las correspondientes ocurrencias del conjunto O que corresponden a términos de $g_0[x]_u$. Por último, la sustitución δ'' sólo puede introducir términos que aparecen en γ, que por construcción no deben reducirse en la derivación $\mathcal{D}'$, o bien términos de ϑ, que tampoco pueden ser reducidos en $\mathcal{D}'$, ya que en ese caso la respuesta θ'' no sería normalizada. Por tanto, todos los términos que deben ser reducidos por las reglas de $\mathcal{R}$ en el objetivo $(C, g_0[t]_u)\delta''$ son S-cerrados. Finalmente, aplicando la hipótesis de inducción, se completa la demostración. $\square$

El resultado anterior no permite establecer aún la equivalencia fuerte entre las semánticas de respuestas computadas del programa original y el programa parcialmente evaluado. Esta es una situación similar a la que ocurre en el caso de la programación lógica, donde el programa especializado puede generar más soluciones que el programa original. Veamos un ejemplo que ilustra esta situación en el caso de los programas lógico-funcionales.

Ejemplo 25 *Dado el siguiente programa* $\mathcal{R}$*:*

$$f(x) \quad \to \quad x$$

y el conjunto $S = \{f(0), f(x)\}$*, una EP de* $\mathcal{R}$ *con respecto a* S *es el programa* $\mathcal{R}'$*:*

$$\begin{array}{lcl} f(0) & \to & 0 \\ f(x) & \to & x \end{array}$$

Es inmediato verificar que $\mathcal{R}' \cup \{f(z) = z\}$ *es* S*-cerrado y tiene una refutación con respuesta computada* $\{z/0\}$*, mientras que* $\mathcal{R} \cup \{f(z) = z\}$ *sólo es capaz de computar una solución más general.*

En el marco para la EP de programas lógicos presentado en [LS91], se introduce una condición de independencia sobre el conjunto de átomos evaluados parcialmente que evita la generación de respuestas adicionales en el programa especializado. Informalmente, la condición de independencia se formula exigiendo que no exista un par de átomos unificables en el conjunto de átomos evaluados parcialmente. El siguiente ejemplo ilustra la necesidad de una noción de independencia más fuerte en el caso de los programas lógico-funcionales.

Ejemplo 26 *Sea S el conjunto de términos $\{f(h(x)), f(x), h(0)\}$. Una EP del programa $\mathcal{R}$:*

$$\begin{array}{lcl} f(x) & \rightarrow & 0 \\ h(x) & \rightarrow & x \end{array}$$

respecto a S, es el programa $\mathcal{R}'$:

$$\begin{array}{rcl} f(h(x)) & \rightarrow & f(x) \\ f(h(x)) & \rightarrow & 0 \\ f(x) & \rightarrow & 0 \\ h(0) & \rightarrow & 0 \end{array}$$

Sea $g \equiv (f(h(z)) = z)$ un objetivo que sólo puede demostrarse S-cerrado usando el término $f(h(x)) \in S$ (es decir, $CSet(S, g) = \{\{1.1\}\}$). Por tanto, g sólo debería computar la solución $\{z/x\}$ en $\mathcal{R}'$. Sin embargo, usando los anteriores resultantes, el objetivo g es capaz de computar también la solución $\{z/0\}$ que, además, está normalizada. El problema se debe, intuitivamente, a que el solapamiento entre los términos de S (concretamente, entre el término $h(0)$ y el subtérmino $h(x)$ de $f(h(x))$) puede producir interferencias entre las reglas, de forma que el objetivo puede reducirse con algún resultante que no se originó a partir de los términos de S que permiten demostrar que g es S-cerrado, lo cual produce respuestas indeseadas.

Con el fin de formular una noción de independencia adecuada recordamos, en primer lugar, el concepto estándar de *solapamiento* entre términos.

Definición 6.3.16 (solapamiento [Klo92]**)** *Dos términos s y t se superponen (solapan) si existe un subtérmino no variable $s_{|u}$ de s tal que $s_{|u}$ y t unifican. Si $s \equiv t$, entonces exigimos que $s_{|u}$ sea un subtérmino propio de s (i.e. $u \not\equiv \Lambda$).*

Así pues, nuestra condición de independencia sustituye la condición de "no unificabilidad" entre los elementos del conjunto, por la condición (más fuerte) de "no solapamiento". La siguiente definición formaliza el concepto de independencia.

Definición 6.3.17 (independencia) *Un conjunto de términos S se dice* independiente *sii no contiene dos términos $s, t \in S$ (no necesariamente distintos) tales que s y t se superponen.*

Los siguientes resultados formalizan algunas propiedades de los conjuntos de términos independientes.

Lema 6.3.18 *Si S es un conjunto de términos independiente, entonces la forma en que un término t puede ser S-cerrado es única, es decir, el conjunto $CSet(S,t)$ es unario.*

DEMOSTRACIÓN. Demostramos el resultado por reducción al absurdo.

Supongamos que existe un término t que puede ser S-cerrado de dos formas distintas. Por la Definición 6.3.2, esto sólo es posible si existe un subtérmino $t_{|u}$ de t que es instancia de dos términos (distintos) $s, s' \in S$. Por tanto, existen dos sustituciones θ, θ' tales que $t_{|u} = s\theta$ y $t_{|u} = s'\theta'$. Ahora, dado que las variables de los términos de S se consideran renombradas aparte, podemos formar la sustitución $\vartheta = \theta \cup \theta'$ tal que $s\vartheta = s'\vartheta$, lo que contradice la hipótesis sobre la independencia de S. □

El siguiente lema formaliza una propiedad fundamental de los conjuntos de términos independientes que permitirá probar que el programa especializado no genera soluciones adicionales, lo cual es esencial para la demostración del Teorema 6.3.20 que establece la corrección y completitud fuertes de la EP de programas lógico-funcionales.

Lema 6.3.19 *Sea S un conjunto de términos independiente, t un término S-cerrado y $t_{|u}$ un subtérmino de t, donde $u \in \bar{O}(t)$. Si $t_{|u}$ unifica con algún término $s \in S$, entonces $s \leq t_{|u}$.*

DEMOSTRACIÓN. Demostramos el resultado por reducción al absurdo.

Supongamos que existe un subtérmino $t_{|u}$ de t, $u \in \bar{O}(t)$, tal que $t_{|u}$ unifica con $s \in S$ y $s \not\leq t_{|u}$.

Sea $\{u_1, \ldots, u_n\} \in CSet(S,t)$. Entonces $\forall i \in \{1, \ldots, n\}$. $\exists s_i \in S$. $s_i \leq t_{|u_i}$, $n \geq 0$. Por el Lema 6.3.18, dicho conjunto de ocurrencias es único. Ahora, podemos considerar dos casos:

- $u \in \{u_1, \ldots, u_n\}$. Entonces, existe un k, $1 \leq k \leq n$, tal que $u \equiv u_k$ y $s_k \leq t_{|u_k}$. Ahora, puesto que existe un término $s \in S$ tal que $t_{|u_k}$ unifica con s pero $s \not\leq t_{|u_k}$, entonces $s_k \not\equiv s$. Por último, dado que $s_k \leq t_{|u_k}$ y $t_{|u_k}$ unifica con s, entonces s_k y s unifican, lo que contradice la hipótesis acerca de la independencia de S.

- $u \notin \{u_1, \ldots, u_n\}$. En este caso, por la Definición 6.3.2 (cierre) y el hecho de que el conjunto de ocurrencias de cierre $\{u_1, \ldots, u_n\}$ es único, existe un i, $1 \leq i \leq n$, y un w tales que $u = u_i.w$, $s_i \leq t_{|u_i}$ y $s_{i|w} \notin V$. Ahora, puesto que $s_i \leq t_{|u_i}$ y $u_i.w = u$, entonces $s_{i|w} \leq t_{|u}$. Finalmente, podemos distinguir dos posibilidades:

  a) $s \not\equiv s_i$. Dado que $s_{i|w} \leq t_{|u}$ y s unifica con $t_{|u}$, entonces $s_{i|w}$ unifica con s, $s_{i|w} \notin V$ y, por tanto, existe solapamiento entre dos términos de S, lo que contradice la hipótesis de independencia.

b) $s \equiv s_i$. Entonces $s_{|w} \leq t_{|u}$ y, dado que $t_{|u}$ unifica con s, existe un subtérmino propio $s_{|w}$ de s que unifica con s, contradiciendo de nuevo la independencia de S.

$\square$

Finalmente, el siguiente resultado muestra que, cuando se exige la independencia del conjunto de términos respecto al cual se realiza la EP, la semántica de respuestas computadas del programa original y el programa especializado coinciden.

Teorema 6.3.20 (corrección y completitud fuertes de la EP)
Sea $\mathcal{R}$ un programa, g un objetivo, S un conjunto finito de términos y $\mathcal{R}'$ una EP de $\mathcal{R}$ con respecto a S tal que $\mathcal{R}' \cup \{g\}$ es S-cerrado. Entonces, se cumple:

1. *(Corrección fuerte)* $\mathcal{O}_{\mathcal{R}'}(g) \subseteq \mathcal{O}_{\mathcal{R}}(g)$, *si S es independiente.*
2. *(Completitud fuerte)* $\mathcal{O}_{\mathcal{R}'}(g) \supseteq \mathcal{O}_{\mathcal{R}}(g)$.

Demostración. La completitud fuerte ha sido ya probada en el Teorema 6.3.15. Respecto a la corrección fuerte, ésta se sigue de forma inmediata a partir del siguiente argumento. Dado un término t cerrado con respecto a S, tal que existe el paso de *narrowing*:

$$\langle t = z, \sigma\rangle \leadsto_{[u,r']} \langle (C, t[\rho]_u = z)\theta, \sigma\theta\rangle$$

en el que se ha empleado la regla r' del programa especializado $\mathcal{R}'$, entonces existe la siguiente derivación de *narrowing*:

$$\langle t = z, \sigma\rangle \leadsto^n \langle (C, t[\rho]_u = z)\theta', \sigma\theta'\rangle$$

usando las reglas del programa original $\mathcal{R}$, de forma que $\theta = \theta'[Var(t)]$.

En primer lugar, dado que $\langle t = z, \sigma\rangle \leadsto_{[u,r']} \langle (C, t[\rho]_u = z)\theta, \sigma\theta\rangle$, existen $u \in \bar{O}(t)$, $r' \equiv (s\vartheta \to \rho \Leftarrow C) \ll \mathcal{R}'$, y $\theta = mgu(\{t_{|u} = s\vartheta\}) \not\equiv fail$. Consideramos que el resultante r' ha sido generado por la siguiente derivación para $s = y$ en $\mathcal{R}$:

$$\langle s = y, \epsilon\rangle \leadsto^n \langle (C, \rho = y), \vartheta\rangle.$$

Puesto que $t_{|u}$ unifica con $s\vartheta$, $s \in S$, entonces $t_{|u}$ también unifica con s. Por el Lema 6.3.19, se cumple que $s \leq t_{|u}$ y, por tanto, existe una sustitución γ tal que $s\gamma = t_{|u}$. Esto nos permite asegurar que no se van a producir respuestas adicionales, ya que no es posible emplear resultantes formados a partir de términos más instanciados que el subtérmino $t_{|u}$ seleccionado.

Ahora, por un argumento similar al empleado en la demostración del Lema 6.3.12, resulta inmediato que, dado que existe la derivación:

$$\langle s = y, \epsilon\rangle \leadsto^n \langle (C, \rho = y), \vartheta\rangle$$

en $\mathcal{R}$, y se cumple que $\sigma = mgu(\{t_{|u} = s\vartheta\}) = mgu(\{s\gamma = s\vartheta\}) \not\equiv fail$, entonces es posible también construir la derivación:

$$\langle t = z, \sigma \rangle \leadsto^n \langle (C, t[\rho]_u = z)\theta', \sigma\theta' \rangle$$

usando las mismas reglas y en el mismo orden que en la derivación para $(s = y)$ (sobre las posiciones correspondientes), de forma que $\theta = \theta'[Var(t)]$. $\square$

El Teorema 6.3.15 y el Teorema 6.3.20 nos han permitido establecer, de forma precisa, la relación existente entre un programa y una EP del mismo. Sin embargo, dichos teoremas no proporcionan ninguna información acerca de cómo debe realizarse en la práctica la EP de un programa, ni cómo pueden garantizarse, a través de un proceso automático, las condiciones de cierre e independencia requeridas. En el siguiente capítulo, presentamos un algoritmo genérico para la especialización de programas lógico-funcionales que da respuesta a estas cuestiones.

Capítulo 7

Un Marco General para la EP de Programas

Este último capítulo está dedicado a la formalización de un método genérico y automático para la EP de programas lógico-funcionales. El esquema que presentamos sigue las líneas generales del marco para asegurar la terminación global de la deducción parcial, formulado por Martens y Gallagher en [MG95]. Nuestro algoritmo es paramétrico respecto a la relación de *narrowing* que construye los árboles de búsqueda, una regla de desplegado, que determina cómo y cuándo detener la construcción de los árboles, y un operador de abstracción, que asegura que el conjunto de términos obtenido durante la EP es finito (lo cual garantiza la terminación del proceso) mientras aún se consigue un nivel adecuado de especialización.

Para el algoritmo genérico, y para todas sus instancias, demostramos la propiedad de cierre del programa parcialmente evaluado. En cambio, la terminación del proceso depende de la elección realizada para la regla de desplegado y para el operador de abstracción. Presentamos una instancia del método que se basa en el uso de una extensión del orden de subsumción estándar (*homeomorphic embedding ordering* [DJ90]) para controlar la expansión de los árboles, junto con el uso del operador de "generalización más especifica" [LMM88] para realizar la operación de abstracción. De esta forma, disponemos de una instancia del algoritmo que, mediante una estrategia simple pero efectiva, garantiza la terminación del proceso y es independiente de la estrategia de *narrowing* empleada para construir los árboles de búsqueda.

La organización del capítulo es la siguiente. En la Sección 7.1 introducimos la idea intuitiva del método, que se presenta formalmente en la Sección 7.2. En primer lugar, formulamos una relación de transición entre configuraciones que representa el núcleo del algoritmo genérico de EP. A continuación demostramos que, si el operador de abstracción propaga correctamente la información de cierre, es posible establecer, de manera genérica, el cierre del programa resultante respecto al conjunto de términos evaluados parcialmente. En la Sección 7.3, presentamos una instancia particular del método que garantiza la terminación de la EP, siendo aún independiente de la estrategia de *narrowing* empleada. Posteriormente, en la Sección 7.4, se trata el problema de garantizar (de forma automática) la independencia del conjunto de términos parcialmente evaluado, y se esboza un procedimiento de renombramiento para el programa especializado. Ilustramos nuestro método con varios ejemplos representativos que consiguen: especialización (en particular, la generación de un programa para el emparejamiento de patrones eficiente) y eliminación de estructuras de datos intermedias. Por último, en la Sección 7.5, mostramos que nuestro método pasa el test KMP, es decir, el programa que resulta de especializar un programa ineficiente de emparejamiento de patrones con respecto a un patrón fijo, posee la eficiencia del algoritmo de Knuth, Morris y Pratt [KMP77] que construye un autómata determinista.

7.1 Introducción

Presentamos en primer lugar una descripción informal, en el estilo de [Gal93], del procedimiento de EP que formalizaremos en la Sección 7.2, con la idea de introducir los principales problemas que se plantean al intentar la automatización. El procedimiento a seguir puede formularse informalmente como sigue. Dado un programa $\mathcal{R}$ y un objetivo ecuacional g, una EP de $\mathcal{R}$ con respecto a g se computa siguiendo los siguientes pasos:

1. Inicializar el conjunto S' con los términos que aparecen en el objetivo g.

2. Actualizar el conjunto S con los términos de S'.

3. Computar una EP $\mathcal{R}'$ de S en $\mathcal{R}$.

4. Formar un nuevo conjunto S' con los términos de S y los términos de $\mathcal{R}'$ que no son S-cerrados.

5. Repetir los pasos $2-4$ hasta que se cumpla $S = S'$.

Asumiendo que el procedimiento anterior termina, éste computa un conjunto de términos S y un conjunto de reglas $\mathcal{R}'$ (la EP de $\mathcal{R}$ con respecto a S) tales que se verifica la condición de cierre para $\mathcal{R}' \cup \{g\}$ con respecto a S. Nótese que, siguiendo el marco de [MG95], no consideramos en el algoritmo la cuestion de la independencia del conjunto S. Esta se puede conseguir mediante algún tipo de técnica de renombramiento posterior (ver e.g. [BL90, BH93] para el caso de programas lógicos) que depende de la estrategia de *narrowing* utilizada, tal y como comentamos en la Sección 7.4. El siguiente ejemplo ilustra el método.

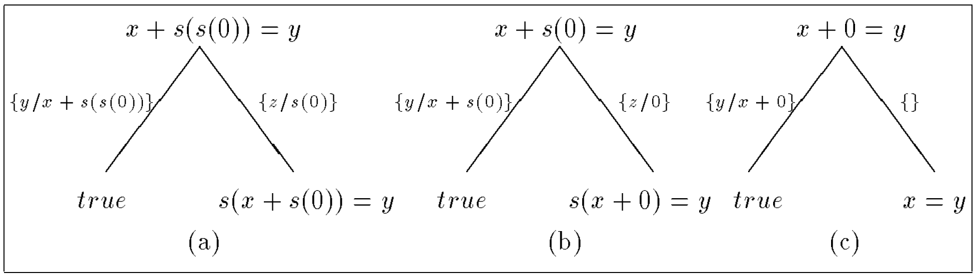

Figura 7.1: Arboles parciales para $(x + s(s(0)) = y)$, $(x + s(0) = y)$, y $(x + 0 = y)$.

Ejemplo 27 *Consideremos de nuevo el programa $\mathcal{R}$ del Ejemplo 18:*

$$\begin{array}{rcl} x + 0 & \rightarrow & x \\ x + s(z) & \rightarrow & s(x + z) \end{array}$$

y el objetivo inicial $(x + s(s(0)) = y)$. Consideramos árboles de búsqueda (parciales) muy simples, construidos por derivaciones "un paso" a partir del objetivo inicial. Partiendo del conjunto $S_0 = \{x + s(s(0))\}$, construimos el árbol de la Figura 7.1 (a) y extraemos los resultantes asociados, derivando el programa:

$$\mathcal{R}_1 = \{x + s(s(0)) \rightarrow s(x + s(0))\}.$$

Ahora, dado que el término $x + s(0)$ no es S_0-cerrado, formamos el nuevo conjunto $S_1 = \{x + s(s(0)),\ x + s(0)\}$. El árbol correspondiente al objetivo $(x + s(0) = y)$ se puede ver en la Figura 7.1 (b). Entonces, la EP de $\mathcal{R}$ con respecto a S_1 es el programa

$$\mathcal{R}_2 = \{x + s(s(0)) \rightarrow s(x + s(0)),\ x + s(0) \rightarrow s(x + 0)\}.$$

Por último, dado que el término $x + 0$ no es S_1-cerrado, obtenemos el conjunto de términos $S = \{x + s(s(0)),\ x + s(0),\ x + 0\}$, de forma que la EP de S en $\mathcal{R}$ es el siguiente programa especializado $\mathcal{R}'$:

$$\begin{array}{rcl} x + s(s(0)) & \rightarrow & s(x + s(0)) \\ x + s(0) & \rightarrow & s(x + 0) \\ x + 0 & \rightarrow & x \end{array}$$

Dado que todos los términos en $\mathcal{R}'$ son S-cerrados, el algoritmo concluye.

Es interesante destacar que, de la misma forma que el hecho de que t sea una instancia de algún término en S no garantiza que t sea S-cerrado (ver Ejemplo 21), la completitud tampoco se asegura para instancias cualesquiera del objetivo inicial, contrariamente a lo que ocurre en programación lógica. El siguiente ejemplo ilustra este punto.

Ejemplo 28 *Consideremos que $\mathcal{R}$, g, S y $\mathcal{R}'$ son los definidos en el Ejemplo 27. El objetivo $g' \equiv (0 + w) + s(s(0)) = y$, que es instancia de g pero no es S-cerrado, tiene una refutación en $\mathcal{R}$ con respuesta computada $\theta = \{w/s(z), y/s(s(s(0 + z)))\}$, mientras que esta respuesta no puede obtenerse usando las reglas de $\mathcal{R}'$.*

Hay dos cuestiones involucradas en la corrección de un procedimiento de EP: terminación –dado un objetivo de entrada, la ejecución debe alcanzar un estado en el que no hay posibilidad de continuar– y corrección parcial –si la ejecución termina, entonces la semántica operacional con respecto al programa especializado y con respecto al programa original debe coincidir–. Disponer de un mecanismo de control automático para realizar el proceso de EP es un punto crucial en el desarrollo de un sistema real para la especialización de programas.

Un evaluador parcial puede entrar en ciclo de dos formas: por entrar en un proceso infinito de desplegado de una llamada, o bien por crear infinitas definiciones especializadas. Siguiendo la aproximación de [MG95], diferenciamos dos niveles en el control de la terminación del proceso:

- El primer nivel, se corresponde con la construcción de los árboles de *narrowing* que generan las evaluaciones parciales para los términos individuales que forman el conjunto S. Habitualmente, nos referiremos a las cuestiones de control involucradas en este proceso como el “control local” de la EP.

- El segundo nivel, el así llamado “control global” de la EP, hace referencia a las decisiones acerca de qué términos deben ser considerados para ser parcialmente evaluados. En este nivel, se debe garantizar que se alcanza la condición de cierre, pero manteniendo un grado de especialización apropiado. Como hemos visto, para alcanzar la condición de cierre suele ser necesario aumentar el conjunto de términos S que aparecen en el objetivo inicial. De esta forma, el control de la terminación global se puede ver como el proceso de garantizar que dicho conjunto S se mantiene finito. Para ello, generalmente se necesita hacer uso de un operador de abstracción que permita “simplificar” algunos de los términos que se introducen.

En la práctica, los dos niveles de terminación suelen presentarse combinados, en el sentido de que las decisiones tomadas en un nivel influyen en el otro. Sin embargo, conceptualmente (y, a menudo, también en la práctica), resulta interesante tratar ambos niveles de manera independiente. Tanto a nivel local como a nivel global, debemos guiar el proceso de EP de forma que el proceso completo obtenga la máxima ganancia en eficiencia (i.e. la mayor especialización posible). Pero, a la vez, resulta fundamental garantizar la terminación del proceso a ambos niveles, de manera que la EP siempre termine computando un programa especializado correcto.

La distinción entre el control local y global de nuestra aproximación contrasta con los métodos de transformación definidos en [GS94, SG95, Tur86] para programas funcionales, donde estas dos cuestiones no se distinguen explícitamente. En dichos trabajos, sólo se tiene en cuenta el control global, construyendo grandes estructuras de evaluación (grafos) que equivalen de alguna manera a nuestros árboles de *narrowing* locales, junto con las configuraciones del marco definido por Martens y Gallagher [MG95] para asegurar la terminación global de la deducción parcial. Cada llamada t en el grafo produce una función residual, cuyo cuerpo se deriva de los descendientes de t en el grafo. En nuestro caso, al igual que en la deducción parcial, el cuerpo de cada regla residual se construye usando la raíz y el nodo final de una derivación, lo cual resulta en un número menor de reglas.

Respecto a la terminación local, algunas de las principales herramientas usadas para controlar la terminación del desplegado en la construcción de los árboles de búsqueda son: límites de profundidad en la expansión del árbol, grafos de detección de bucles, y el uso de órdenes bien fundados [BdSM92]. Respecto a la terminación global, una primera posibilidad sería imponer simplemente un límite al número de términos que puede contener el conjunto S. Sin embargo, ésta no parece la solución más adecuada (además, sería muy complejo asegurar la condición de cierre). En la Sección 7.3.2, definiremos un operador de abstracción basado en la noción de *generalización más específica* (*msg*, "most specific generalization" [LMM88]), que nos permitirá disponer de un método más racional para asegurar la terminación del proceso.

En la siguiente sección, formalizamos nuestro esquema genérico para la EP de programas lógico-funcionales y establecemos las propiedades del mismo. Asumimos que existe una estrategia de desplegado apropiada que decide qué términos desplegar y cómo detener el desplegado, y relegamos así todas las cuestiones relacionadas con la terminación del proceso a la Sección 7.3.

7.2 Un Esquema Genérico para la EP

En esta sección, formalizamos un algoritmo general para la EP de programas lógico-funcionales basado en *narrowing*, que siempre termina (para instancias apropiadas) y que es capaz de soportar un nivel adecuado de *polivarianza* –es decir, la posibilidad de producir varias especializaciones independientes para una llamada a función dada, usando distintos datos de entrada–, pese al empleo del operador de generalización *msg* (cuyo uso suele venir acompañado de una cierta pérdida de precisión). Nuestro algoritmo es genérico con respecto a los siguientes parámetros:

1. La *relación de narrowing*, con la que se construyen los árboles de búsqueda parciales.

2. La *regla de desplegado*, que determina cuándo y cómo terminar la construcción de los árboles.

3. El dominio de *configuraciones*, que determina la cantidad de información a considerar (acerca de los términos parcialmente evaluados) en el nivel global del procedimiento.

4. El *operador de abstracción*, usado para garantizar que el conjunto de términos obtenidos durante el proceso de EP se mantiene finito.

Nuestro procedimiento genérico está inspirado en el marco para asegurar la terminación global de la deducción parcial de programas lógicos presentado en [MG95], pero la extensión a un contexto lógico–funcional no es en absoluto trivial. Introducimos, en primer lugar, el concepto genérico de *regla de desplegado*. Informalmente, una regla de desplegado $U_{\leadsto_\varphi}$ (que denotaremos simplemente por U_φ, cuando no haya confusión posible), construye árboles de búsqueda finitos (posiblemente incompletos) usando la relación de *narrowing* $\leadsto_\varphi$ y extrae, de éstos, los resultantes asociados a las derivaciones del árbol[1].

Definición 7.2.1 (regla de desplegado U_φ) *Una regla de desplegado U_φ es una función que, dado un programa $\mathcal{R}$, un término s y una relación de narrowing $\leadsto_\varphi$, computa un conjunto finito de resultantes $U_\varphi(s, \mathcal{R})$ que constituyen una EP de s en $\mathcal{R}$ usando $\leadsto_\varphi$.*

Si S es un conjunto finito de términos y $\mathcal{R}$ es un programa, entonces el conjunto de resultantes obtenidos aplicando U_φ a cada término $s \in S$, se denomina EP de S en $\mathcal{R}$ usando U_φ (en símbolos $U_\varphi(S, \mathcal{R})$).

Formulamos nuestro método para computar una EP de un programa $\mathcal{R}$ con respecto a un conjunto de términos S usando $\leadsto_\varphi$, mediante un sistema de transición $(Conf, \longmapsto_\mathcal{P})$ cuya relación de transición $\longmapsto_\mathcal{P} \subseteq Conf \times Conf$ formaliza los pasos de computación. El conjunto $Conf$ de configuraciones es un parámetro de la definición, de modo que la noción de configuración es un parámetro a ser precisado en el proceso de instanciación. Denotamos por $c[S] \in Conf$ una configuración genérica, cuya estructura se deja sin especificar ya que depende del algoritmo de EP específico, pero que debe contener al menos el conjunto de términos S. Cuando S esté claro por el contexto, denotaremos $c[S]$ simplemente por c.

[1]Consideramos, en esta sección, que los conceptos asociados a la EP introducidos en el capítulo anterior se extienden, de manera natural, al caso de una relación de *narrowing* genérico con estrategia $\leadsto_\varphi$.

Definición 7.2.2 (relación de transición $\longmapsto_{\mathcal{P}}$) *Definimos la relación de transición $\longmapsto_{\mathcal{P}}$ como la menor relación que satisface:*

$$\frac{\mathcal{R}' = U_\varphi(S, \mathcal{R})}{c[S] \longmapsto_{\mathcal{P}} abstract(c[S], terms(\mathcal{R}'))}$$

donde la función $abstract(c, T)$ extiende (de manera apropiada) la configuración actual c con el conjunto de términos T, dando lugar a una nueva configuración.

Informalmente, en cada paso de computación, el conjunto de términos S (almacenados en $c[S]$) es parcialmente evaluado (usando U_φ). Entonces, los términos que aparecen en el programa residual $\mathcal{R}'$ que no sean S-cerrados, se introducen (apropiadamente) en la configuración c, para ser evaluados en la proxima iteración del algoritmo. Para asegurar la terminación del proceso, la actualización de las configuraciones se realiza usando un operador de abstracción que debe garantizar que el conjunto de términos se mantiene finito. De forma similar a [MG95], la aplicación del operador *abstract* en cada iteración nos permite ajustar el nivel de polivarianza deseado, así como controlar el progreso hacia la terminación de la EP del programa.

La siguiente definición formaliza el núcleo de nuestro método de EP.

Definición 7.2.3 (comportamiento del cálculo $\longmapsto_{\mathcal{P}}$) *Sea c_0 la configuración "vacía" del dominio Conf. El comportamiento de la relación de transición $\longmapsto_{\mathcal{P}}$ viene determinado por la siguiente función:*

$$\mathcal{P}(\mathcal{R}, g) = S \quad si \quad abstract(c_0, terms(g)) \longmapsto^*_{\mathcal{P}} c[S] \quad y \quad c[S] \longmapsto_{\mathcal{P}} c[S].$$

Nótese que el procedimiento de la Definición 7.2.3 no construye un programa especializado, sino que computa un conjunto de términos parcialmente evaluados S. Sin embargo, el conjunto de términos S determina de forma unívoca la EP $\mathcal{R}'$ de S en $\mathcal{R}$ (usando U_φ).

El siguiente lema es necesario para demostrar la corrección del método, y muestra que la noción de cierre introducida posee una cierta forma de propiedad de transitividad.

Lema 7.2.4 *Si t es S_1-cerrado y S_1 es S_2-cerrado, entonces t es S_2-cerrado.*

Demostración. Realizamos la prueba por inducción estructural sobre la profundidad del término t.
Si $depth(t) = 1$, el resultado es trivial.
Si $depth(t) = k + 1$, $k > 0$, y la propiedad se cumple para todo término t' tal que $depth(t') \leq k$ entonces, por la Definición 6.3.2, podemos considerar dos casos:

1. $t = c(t_1, \ldots, t_n)$, $c \in \mathcal{C}$, y $closed(S_1, \{t_1, \ldots, t_n\})$;

2. $t = f(t_1, \ldots, t_n)$, $f \in \mathcal{F}$, existe un término $s_1 \in S_1$. $s_1\theta_1 = t$ y se verifica $closed(S_1, terms(\widehat{\theta}_1)$.

Consideremos el primer caso. Por la hipótesis de inducción se cumple que, para todo $i = 1, \ldots, n$, el término t_i es S_2-cerrado. Por tanto, por la Definición 6.3.2, t es S_2-cerrado, ya que $c \in \mathcal{C}$.
Consideremos el segundo caso. Ya que $s_1 \in S_1$ entonces, por las hipótesis del lema, s_1 es S_2-cerrado y, por tanto, existe un término $s_2 \in S_2$. $s_2\theta_2 = s_1$ y el predicado $closed(S_2, terms(\widehat{\theta}_2))$ es cierto. Entonces, $s_2\theta_2\theta_1 = s_1\theta_1 = t$. Finalmente, debemos demostrar que $closed(S_2, terms(\widehat{\theta_2\theta_1}))$ se evalúa a cierto. Puesto que $closed(S_1, terms(\widehat{\theta}_1))$ es cierto entonces, por la hipótesis de inducción, $closed(S_2, terms(\widehat{\theta}_1))$ también lo es. Ahora, ya que $terms(\widehat{\theta_2\theta_1}) = terms(\widehat{\theta}_2\theta_1 \cup \widehat{\theta}_1\lceil_{(V-Dom(\theta_2))}) = terms(\widehat{\theta}_2)\theta_1 \cup terms(\widehat{\theta}_1\lceil_{(V-Dom(\theta_2))})$, se cumple el resultado enunciado. □

El siguiente teorema establece que el programa transformado es cerrado respecto al conjunto de términos parcialmente evaluado, siempre que el operador de abstracción considerado propague correctamente la información de cierre.

Teorema 7.2.5 *Sea $\mathcal{R}$ un programa y g un objetivo ecuacional. Sea 'abstract' un operador de abstracción particular que satisface la siguiente condición:*

$$abstract(c_1[S_1], S') = c_2[S_2] \implies (S_1 \cup S') \text{ es } S_2\text{-cerrado.}$$

Entonces, si la función $\mathcal{P}(\mathcal{R}, g)$ termina computando un conjunto de términos S, se cumple que $\mathcal{R}' \cup \{g\}$ es S-cerrado, donde $\mathcal{R}' = U_\varphi(S, \mathcal{R})$.

Demostración. Consideremos que $\mathcal{P}(\mathcal{R}, g)$ termina computando el conjunto de términos S. Por la Definición 7.2.3, se cumple:

$$abstract(c_0, terms(g)) \longmapsto^*_{\mathcal{P}} c[S] \;\text{ y }\; c[S] \longmapsto_{\mathcal{P}} c[S].$$

Por la Definición 7.2.2, $\mathcal{R}' = U_\varphi(S, \mathcal{R})$ y $c[S] = abstract(c[S], terms(\mathcal{R}'))$. Entonces, por la condición sobre el operador *abstract* impuesta en el teorema, todo término $s \in terms(\mathcal{R}')$ es S'-cerrado y, por tanto, $\mathcal{R}'$ es S'-cerrado también (por la Definición 6.3.2).
Mostramos ahora que, para todo $i = 0, \ldots, n$, si $c_1[S_1] \longmapsto^*_{\mathcal{P}} c_{i+1}[S_{i+1}]$, entonces el objetivo g es S_{i+1}-cerrado. Realizamos la prueba por inducción sobre la longitud i de la computación. El caso base $i = 0$ es trivial, así que consideramos el caso inductivo $i > 0$. Por la hipótesis de inducción, g es S_i-cerrado. Por la condición impuesta sobre el operador *abstract*, para todo $j = 1, \ldots, i$, se cumple que S_j es S_{j+1}-cerrado. Por tanto, el resultado se sigue directamente aplicando el Lema 7.2.4. □

El teorema anterior garantiza, en el caso de que la especialización de programas esté basada en *narrowing* condicional, la corrección parcial del algoritmo (ya que se satisfacen las condiciones del Teorema 6.3.15). En el caso de otras estrategias de *narrowing*, nuestras investigaciones han constatado que la condición de cierre es necesaria, pero no suficiente, para garantizar la completitud de la transformación [AFJV96a, AFJV96b].

La Definición 7.2.3 representa sólo el esquema de un método completo de EP para lenguajes lógico-funcionales. Las evaluaciones parciales resultantes pueden ser aún optimizadas eliminando símbolos de función redundantes y repeticiones de variables innecesarias, adaptando, por ejemplo, las técnicas estándar presentadas en [BH93, BL90, Gal93, GB90]. En la siguiente sección abordamos el problema de garantizar la terminación (local y global) del algoritmo de EP.

7.3 Terminación

En esta sección, instanciamos algunos de los parámetros de nuestro método de EP de programas lógico-funcionales, introducido en la Definición 7.2.3, para garantizar la terminación del proceso. Para ello, introducimos una regla de desplegado, un dominio de configuraciones y un operador de abstracción particulares, con los que demostramos la terminación del algoritmo presentado en la Definición 7.2.3. Por supuesto, otras elecciones serían también posibles, como veremos en algunos ejemplos posteriores. La estrategia de *narrowing* empleada se mantiene, en cualquier caso, como parámetro de la construcción.

Las herramientas empleadas habitualmente para demostrar propiedades de terminación, suelen estar basadas en el uso de algun tipo de orden. Intuitivamente, utilizamos un cuasi-orden bien fundado según el cual un término que es "sintácticamente más simple" que otro, se considera más pequeño. La siguiente definición extiende el orden de subsumción estándar (*homeomorphic embedding ordering* [DJ90]) a términos que (posiblemente) contienen variables.

Definición 7.3.1 (relación de subsumción) [SG95] *La relación de subsumción* $\trianglelefteq$ *sobre términos de* $\tau(\Sigma \cup V)$ *se define como la menor relación que satisface:* $x \trianglelefteq y$ *para todo* $x, y \in V$, *y* $s \equiv f(s_1, \ldots, s_m) \trianglelefteq g(t_1, \ldots, t_n) \equiv t$, *si y sólo si:*

1. $f \equiv g$ *(con* $m \equiv n$*) y* $s_i \trianglelefteq t_i$ *para todo* $i = 1, \ldots, n$, *o bien*
2. $s \trianglelefteq t_j$, *para algún* j, $1 \leq j \leq n$.

Intuitivamente, s está subsumido por t, i.e. $s \trianglelefteq t$, si s se puede obtener a partir de t por eliminación de operadores. Por ejemplo,

$$\surd\surd(u \times (u + v)) \trianglelefteq (w \times \surd\underline{\surd\surd}((\surd\underline{u} + \surd u)\underline{\times}(\surd\underline{u+}\surd\underline{v}))).$$

El siguiente teorema, una variante del Teorema de Kruskal [Kru60], es la base para utilizar dicho orden en el estudio de las propiedades de terminación.

Teorema 7.3.2 *Para toda secuencia infinita de términos $t_1, t_2, \ldots$ (construída a partir de un número finito de operadores), existen un par de naturales i, j tal que $i < j$ y $t_i \trianglelefteq t_j$.*

Demostración. Sea σ_ϕ la sustitución que asigna a todas las variables de V un símbolo de constante libre ϕ. Es inmediato comprobar que, para todo par de términos $s, t \in \tau(\Sigma \cup V)$, $s \trianglelefteq t$ sii $s\sigma_\phi \preceq t\sigma_\phi$, donde $\preceq$ es el orden de subsumción estándar sobre el conjunto de términos básicos $\tau(\Sigma \cup \{\phi\})$. Entonces, ya que $\preceq$ es un cuasi-orden, también lo es $\trianglelefteq$, y el resultado se sigue a partir del Teorema de Kruskal. □

En los dos siguientes apartados estudiamos, de manera independiente, los problemas de terminación presentes en el método de EP. La relación de subsumción será usada, en la Sección 7.3.1, para asegurar la terminación de los árboles de *narrowing* y, en la Sección 7.3.2, para asegurar que el conjunto de términos parcialmente evaluados permanece finito.

7.3.1 Terminación Local

En esta sección, presentamos una estrategia de desplegado que intenta maximizar la expansión del árbol de *narrowing*, pero manteniendo siempre la terminación (local) del proceso. Nuestra estrategia es simple, pero menos drástica que imponer un límite *ad-hoc* a la profundidad del árbol. El método se inspira en el trabajo de Sørensen y Glück [SG95], en el que se utiliza un orden de subsumción para garantizar la terminación de la supercompilación positiva.

El siguiente criterio hace uso de la relación de subsumción de una forma constructiva para producir árboles de *narrowing* finitos. Informalmente, para evitar secuencias infinitas de llamadas divergentes, comparamos el *redex* seleccionado por la relación de *narrowing* con los *redexes* seleccionados en los objetivos anteriores de la misma derivación. Cuando el último *redex* de la derivación subsume a un *redex* anterior de la misma derivación, detenemos la expasión de dicha rama del árbol de búsqueda.

En principio, la comparación citada anteriormente puede ser innecesariamente fuerte. Partiendo del hecho de que existe un número finito de operadores en Σ, podemos limitar la comparación sólo a ciertos términos, tal y como se formaliza en la siguiente definición.

Definición 7.3.3 (términos comparables) *Dados dos términos $s, t \in \tau(\Sigma \cup V)$, decimos que s y t son comparables, y lo denotamos por comparable(s, t), si los símbolos de función más externos de s y t coinciden.*

A continuación introducimos la noción auxiliar de *derivación admisible*.

Definición 7.3.4 (derivación admisible) *Sea $\mathcal{D}$ una derivación de narrowing para g_0 en $\mathcal{R}$. Decimos que $\mathcal{D}$ es admisible sii no contiene un par de redexes comparables incluidos en la relación $\trianglelefteq$. Formalmente,*

$$\begin{aligned}
&admissible(\langle g_0, \theta_0\rangle \leadsto_{\varphi[u_0,r_0]} \ldots \leadsto_{\varphi[u_{n-1},r_{n-1}]} \langle g_n, \theta_n\rangle)\\
&\qquad \Leftrightarrow \ \forall i = 1,\ldots,n,\ \forall u \in \varphi(g_i),\ \forall j = 0,\ldots,i-1.\\
&\qquad\qquad (comparable(g_{j|u_j}, g_{i|u}) \ \Rightarrow\ g_{j|u_j} \not\trianglelefteq g_{i|u}).
\end{aligned}$$

Ahora, haciendo uso de la noción de derivación admisible, introducimos un tipo particular de árboles de *narrowing* (posiblemente) incompletos, que garantizan la terminación de todas las derivaciones.

Definición 7.3.5 (árbol de narrowing sin subsumciones $\tau_\varphi^{\trianglelefteq}$) *Sea $\mathcal{R}$ un programa y g_0 un objetivo ecuacional. Definimos el árbol de narrowing sin subsumciones $\tau_\varphi^{\trianglelefteq}(g_0,\mathcal{R})$, asociado a g_0 y $\mathcal{R}$, como sigue:*

$$\begin{aligned}
\tau_\varphi^{\trianglelefteq}(g_0,\mathcal{R}) = \{\ &\langle g_0,\theta_0\rangle \leadsto_{\varphi[u_0,r_0]} \ldots \leadsto_{\varphi[u_{n-2},r_{n-2}]} \langle g_{n-1},\theta_{n-1}\rangle \leadsto_{\varphi[u_{n-1},r_{n-1}]} \langle g_n,\theta_n\rangle \mid\\
&admissible(\langle g_0,\theta_0\rangle \leadsto_{\varphi[u_0,r_0]} \ldots \leadsto_{\varphi[u_{n-2},r_{n-2}]} \langle g_{n-1},\theta_{n-1}\rangle) \ \wedge\\
&(g_n \equiv \top \ \ \vee\ \ g_n \text{ es una hoja de fallo } \vee\\
&(\exists u \in \varphi(g_n),\ \exists i < n.\ comparable(g_{i|u_i}, g_{n|u}) \ \wedge\ g_{i|u_i} \trianglelefteq g_{n|u}))\}.
\end{aligned}$$

De acuerdo a la definición anterior, las derivaciones se detienen cuando éstas tienen éxito, fallan, o algún *redex* del último objetivo desplegado subsume alguno de los *redexes* anteriores de la misma derivación. Intuitivamente, si se explota un *redex* "mayor" (con respecto al orden $\trianglelefteq$) que un *redex* seleccionado previamente (de la misma derivación), no podemos garantizar que el riesgo de generar una derivación infinita sea nulo y, por tanto, detenemos la derivación evitando así el peligro. Este es un criterio "seguro", tal y como se establece en el siguiente teorema, y aún proporciona especializaciones significativas. Nótese que nuestro método para construir árboles finitos es independiente de la estrategia de *narrowing* empleada. Obviamente, para estrategias refinadas, este criterio se podría combinar con otros que exploten las características particulares de la estrategia de *narrowing* aplicada.

Teorema 7.3.6 (terminación local) *Dado un programa $\mathcal{R}$ y un objetivo ecuacional g, $\tau_\varphi^{\trianglelefteq}(g,\mathcal{R})$ es un árbol de narrowing finito (posiblemente incompleto) para $\mathcal{R}\cup\{g\}$ usando $\leadsto_\varphi$.*

Demostración. En primer lugar, podemos afirmar que $\tau_\varphi^{\trianglelefteq}(g_0,\mathcal{R})$ es un árbol de *narrowing* (posiblemente incompleto) para $\mathcal{R} \cup \{g_0\}$ usando $\leadsto_\varphi$, puesto que todas y cada una de las derivaciones (posiblemente incompletas) para g_0 en $\mathcal{R}$ usando $\leadsto_\varphi$ son consideradas.

Ahora, debemos demostrar que toda derivación $\mathcal{D} \in \tau_\varphi^{\trianglelefteq}(g_0,\mathcal{R})$ es finita. Demostramos el resultado por reducción al absurdo. Sea $\mathcal{D} \in \tau_\varphi^{\trianglelefteq}(g_0,\mathcal{R})$ una derivación infinita.

Esto nos permite construir una secuencia infinita $\mathcal{S}$ de términos $g_{0|u_0}, g_{1|u_1}, \ldots$ formada por los *redexes* seleccionados en cada paso de la derivación $\mathcal{D}$. Consideremos ahora que el conjunto de símbolos de función definidos es $\mathcal{F} = \{f_i\}_{i=1}^n$. Entonces, podemos dividir la secuencia $\mathcal{S}$ como máximo en n subsecuencias (posiblemente infinitas) $\mathcal{S}_i$ de términos, cada una de las cuales está formada por los términos de $\mathcal{S}$ cuyo símbolo de función más externo es f_i, $i = 1, \ldots, n$. Así, todos los términos en cada subsecuencia $\mathcal{S}_i$ son comparables. Puesto que la secuencia $\mathcal{S}$ es infinita y el número de símbolos de función definidos es finito, entonces al menos una de las subsecuencias es también infinita. Sin pérdida de generalidad, consideramos que $\mathcal{S}_m$, $1 \leq m \leq n$ es una subsecuencia infinita. Por el Teorema 7.3.2, $\mathcal{S}_m$ contiene dos términos en la relación $\trianglelefteq$, lo que contradice la hipótesis de partida. $\square$

Veamos ahora un ejemplo que ilustra la Definición 7.3.5, mostrando de qué forma se previenen derivaciones de *narrowing* infinitas.

Ejemplo 29 *Consideremos el clásico programa append/2 para la concatenación de dos listas*[2]*:*

$$\begin{array}{rcl} append(nil, y_s) & \rightarrow & y_s \\ append(x : x_s, y_s) & \rightarrow & x : append(x_s, y_s) \end{array}$$

y el objetivo inicial $append(1 : 2 : x_s, y_s) = y$. *Entonces, existe la siguiente rama infinita en el árbol de narrowing condicional para el objetivo:*

$$\begin{array}{rl} \langle \underline{append(1 : 2 : x_s, y_s)} = y, \epsilon\rangle & \leadsto \langle 1 : \underline{append(2 : x_s, y_s)} = y, \epsilon\rangle \\ & \leadsto \langle 1 : 2 : \underline{append(x_s, y_s)} = y, \epsilon\rangle \\ & \leadsto \langle 1 : 2 : x' : \underline{append(x'_s, y_s)} = y, \{x_s/x' : x'_s\}\rangle \\ & \leadsto \ldots \end{array}$$

Sin embargo, en el árbol de narrowing sin subsumciones (y de acuerdo con la Definición 7.3.5), el desplegado de esta rama se detiene en el cuarto objetivo, ya que la siguiente subderivación:

$$\begin{array}{rl} \langle append(1 : 2 : x_s, y_s) = y, \epsilon\rangle & \leadsto \langle 1 : append(2 : x_s, y_s) = y, \epsilon\rangle \\ & \leadsto \langle 1 : 2 : append(x_s, y_s) = y, \epsilon\rangle \end{array}$$

es admisible, y el objetivo $\langle 1 : 2 : x' : append(x'_s, y_s) = y, \{x_s/x' : x'_s\}\rangle$ *contiene un redex que subsume al redex seleccionado en el paso anterior:*

$$append(x_s, y_s) \trianglelefteq append(x'_s, y_s).$$

Una vez que disponemos de un método automático para generar árboles de búsqueda finitos, la regla de desplegado correspondiente se puede definir como sigue.

[2] Siguiendo la notación habitual en los programas funcionales, usamos 'nil' y ':' como constructores de listas.

Definición 7.3.7 (regla de desplegado $U_{\varphi}^{\trianglelefteq}$)
Definimos $U_{\varphi}^{\trianglelefteq}(s,\mathcal{R})$ como la EP de s en $\mathcal{R}$ usando $\tau_{\varphi}^{\trianglelefteq}(s=y,\mathcal{R})$, $y \notin Var(s)$.

Como ya comentamos, la no terminación del procedimiento de EP puede estar provocada, no sólo por la construcción de un árbol de *narrowing* infinito, sino también por no alcanzar nunca la condición de cierre (y, por tanto, generar un conjunto infinito de términos parcialmente evaluados). En la siguiente sección abordamos dicho problema.

7.3.2 Terminación Global

La terminación del algoritmo presentado en la Definición 7.2.3 puede garantizarse definiendo un dominio de configuraciones y un operador de abstracción apropiados. En esta sección, introducimos un dominio de configuraciones simple, compuesto por secuencias de términos, y un operador de abstracción que manipula dichas secuencias de forma que la terminación del proceso está garantizada. Posteriormente, mostraremos que esta instancia parcial del algoritmo genérico asegura, de una forma sencilla, la terminación global sin incurrir en una pérdida importante de especialización. En [MG95] se puede encontrar un dominio de configuraciones más sofisticado, basado en el uso árboles cuyos nodos son los átomos parcialmente evaluados, con el que se puede conseguir, en algunos casos, mejorar el nivel de especialización del programa residual.

Introducimos ahora el dominio de configuraciones $Conf^*$, formado por secuencias (ordenadas) de términos pertenecientes a $\tau(\Sigma \cup V)$.

Definición 7.3.8 (configuraciones PE^*) *Sea $Conf^* = \tau(\Sigma \cup V)^*$ el monoide libre estándar sobre el conjunto de términos, donde la secuencia vacía de términos se denota por "nil" y la operación de concatenación se denota por ",". Definimos una configuración PE^* como una secuencia de términos $(t_1,\ldots,t_n) \in Conf^*$. La configuración PE^* vacía se denota por nil.*

Tras cada iteración del algoritmo, la configuración actual $q \equiv (t_1,\ldots,t_n)$ debe ser transformada para "cubrir" los nuevos términos que resultan de la EP de q en $\mathcal{R}$, es decir, $terms(U_{\varphi}(\{t_1,\ldots,t_n\},\mathcal{R}))$. Esta transformación se realiza usando el siguiente operador de abstracción $abstract^*(q,T)$.

Definición 7.3.9 (operador $abstract^*$) *Sea q una configuración PE^* y T una expresión (un término o un conjunto de términos). Definimos inductivamente la función $abstract^*$ como sigue:*

$$abstract^*(q,T) = \begin{cases} q & si\ T \equiv \emptyset \ \acute{o}\ T \equiv x \in V \\ abstract^*(\ldots abstract^*(q,t_1),\ldots,t_m) & si\ T \equiv \{t_1,\ldots,t_m\} \\ abstract^*(q,\{t_1,\ldots,t_n\}) & si\ T \equiv c(t_1,\ldots,t_n),\ c \in \mathcal{C} \\ abs\_call(q,T) & si\ T \equiv f(t_1,\ldots,t_n),\ f \in \mathcal{F} \end{cases}$$

con $m \geq 1$ y $n \geq 0$, donde la función abs_call(q, T) (para un término T) se define de la forma:

$$
\begin{array}{l}
abs\_call(nil, T) = T \\
abs\_call(q, T) = \left\{
\begin{array}{ll}
(q_1, \ldots, q_n, T) & si\ \not\exists i \in \{1, \ldots, n\}.\ (comparable(q_i, T) \\
 & \quad y\ q_i \trianglelefteq T) \\
abstract^*((q_1, \ldots, q_n), T') & si\ i = \max\limits_{j=1,\ldots,n}(comparable(q_j, T)), \\
 & \quad q_i \trianglelefteq T,\ \exists\theta.\ q_i\theta = T,\ \ y\ T' = terms(\widehat{\theta}) \\
abstract^*(q', T') & si\ i = \max\limits_{j=1,\ldots,n}(comparable(q_j, T)), \\
 & \quad q_i \trianglelefteq T,\ T\ no\ es\ una\ instancia\ de\ q_i, \\
 & \quad msg(\{q_i, T\}) = \langle w, \{\theta_1, \theta_2\}\rangle, \\
 & \quad q' \equiv (q_1, \ldots, q_{i-1}, q_{i+1}, \ldots, q_n)),\ \ y \\
 & \quad T' = \{w\} \cup terms(\widehat{\theta}_1 \cup \widehat{\theta}_2)
\end{array}
\right.
\end{array}
$$

donde $q \equiv (q_1, \ldots, q_n)$.

Informalmente, la función $abstract^*(q, t)$ extiende la configuración actual q con un nuevo término t de la siguiente forma:

- Si t es una variable o una constante, simplemente se ignora.
- Si t es un término encabezado por un símbolo constructor, se realiza una llamada recursiva para incorporar sus argumentos.
- Si t es un término encabezado por un símbolo de función definido, consideramos los siguientes casos:
  1. t no subsume a ningún término comparable de q. En este caso, t se añade directamente a q, ya que incrementará (posiblemente) la especialización del programa residual, sin riesgo para la terminación.
  2. t sí subsume a algún término comparable de q. En este caso, si t está (parcialmente) cerrado por los términos de q, se realiza una llamada recursiva para añadir sólo aquellos subtérminos que no estén cerrados.
  3. Por último, si t subsume a un cierto término t' de la configuración q y no es instancia del mismo, entonces computamos una generalización de ambos términos $msg(\{t, t'\})$, y realizamos una llamada recursiva para añadir los términos de la generalización.

Queremos hacer notar que la pérdida de precisión causada por el uso del operador de generalización msg es bastante razonable y se compensa con la simplicidad del método resultante.

Para garantizar que el algoritmo de la Definición 7.2.3 genera realmente un programa cerrado, es suficiente comprobar que el operador $abstract^*$ satisface la premisa del Teorema 7.2.5, i.e. propaga correctamente la información sobre el cierre de los términos.

Lema 7.3.10 *Sean q, q' configuraciones PE^* y S un conjunto de términos. Si $abstract^*(q, S) = q'$, entonces los términos de $terms(q) \cup S$ están cerrados con respecto a $terms(q')$.*

DEMOSTRACIÓN. Probamos el lema por inducción bien fundada sobre el conjunto S. Definimos la complejidad $\mathcal{M}_S$ de S como el multiconjunto finito de números naturales correspondiente a la profundidad de los elementos de S. Formalmente, $\mathcal{M}_S = \{depth(s) \mid s \in S\}$. Consideramos el orden total bien fundado $<_{mul}$ sobre complejidades extendiendo el orden bien fundado $<$ sobre $I\!N$ al conjunto $M(I\!N)$ de multiconjuntos finitos sobre $I\!N$. El conjunto $M(I\!N)$ está bien fundado bajo el orden $<_{mul}$ [DJ90]. Sean $\mathcal{M}, \mathcal{M}'$ complejidades, definimos el orden $<_{mul}$ como sigue: $\mathcal{M} <_{mul} \mathcal{M}' \quad \Leftrightarrow \quad \exists X \subseteq \mathcal{M}, \exists X' \subseteq \mathcal{M}'$ tales que $\mathcal{M} = (\mathcal{M}' - X') \cup X$ y $\forall n \in X.\ \exists n' \in X'.\ n < n'$.

Si $S = \emptyset$, entonces $q' = abstract^*(q, S) = abstract^*(q, \emptyset) = q$ y la prueba está terminada.
Si $S = S_0 \cup \{s\}$, $S_0 = \{s_1, \ldots, s_{n-1}\}$, y la propiedad se cumple para todo S' tal que $\mathcal{M}_{S'} <_{mul} \mathcal{M}_S$ entonces, por la Definición 7.3.9 y el Lema 7.2.4, los términos de $terms(q) \cup S_0$ están cerrados con respecto a $terms(abstract^*(q, S_0))$. Consideramos a continuación tres casos.

Si s es una variable o una constante $c \in \mathcal{C}$ entonces, por la Definición 7.3.9, $q' = abstract^*(q, S) = abstract^*(q, S_0 \cup \{s\}) = abstract^*(abstract^*(q, S), s) = abstract^*(q, S_0)$. Por la hipótesis de inducción, los términos de $terms(q) \cup \{S_0\}$ están cerrados con respecto a $terms(q')$ y, por la Definición 6.3.2, también lo están los términos de $terms(q) \cup S_0 \cup \{s\}$.

Si $s = c(t_1, \ldots, t_m)$, $c \in \mathcal{C}$, $m > 0$, entonces $q' = abstract^*(q, S) = abstract^*(q, S_0 \cup \{s\}) = abstract^*(abstract^*(q, S_0), c(t_1, \ldots, t_m)) = abstract^*(abstract^*(q, S_0), \{t_1, \ldots, t_m\}) = abstract^*(q, S_0 \cup \{t_1, \ldots, t_m\})$. Puesto que $\mathcal{M}_{S_0 \cup \{t_1, \ldots, t_m\}} <_{mul} \mathcal{M}_{S_0 \cup \{c(t_1, \ldots, t_m)\}} = \mathcal{M}_S$ entonces, por la hipótesis de inducción, los términos de $terms(q) \cup S_0 \cup \{t_1, \ldots, t_m\}$ están cerrados con respecto a $terms(q')$ y, por la Definición 6.3.2, también lo están los términos de $terms(q) \cup S_0 \cup \{c(t_1, \ldots, t_m)\} = terms(q) \cup S$.

Si $s = f(t_1, \ldots, t_m)$, $f \in \mathcal{F}$, $m \geq 0$, entonces $q' = abstract^*(q, S) = abstract^*(q, S_0 \cup \{s\}) = abstract^*(abstract^*(q, S_0), s) = abs\_call(abstract^*(q, S_0), s)$. Asumimos que $abstract^*(q, S_0)$ es $(q_1, \ldots, q_p)$, $p \geq 1$, ya que el caso en el que $abstract^*(q, S_0) = nil$ es inmediato. Entonces, tenemos que demostrar que los términos de $abs\_call((q_1, \ldots, q_p),$ $s)$ están cerrados con respecto a q'. Ahora podemos distinguir nuevamente tres casos, que se corresponden con los tres casos de la función *abs_call* en la Definición 7.3.9, concretamente: $abs\_call((q_1, \ldots, q_p), s) =$

$$\begin{cases} 1)\ (q_1, \ldots, q_p, s) & \\ & \\ 2)\ abstract^*((q_1, \ldots, q_p), terms(\widehat{\theta})) & \text{donde } \exists\ i \in \{1, \ldots, p\}.\ q_i\theta = s. \\ 3)\ abstract^*(q', s') & \text{donde } \exists\ i \in \{1, \ldots, p\}.\ msg(\{q_i, s\}) = \langle w, \{\theta_1, \theta_2\}\rangle \\ & \quad s \text{ no es una instancia de } q_i, \\ & \quad q' \equiv (q_1, \ldots, q_{i-1}, q_{i+1}, \ldots, q_p),\ \text{y} \\ & \quad s' = \{w\} \cup terms(\widehat{\theta}_1 \cup \widehat{\theta}_2). \end{cases}$$

El caso 1) es inmediato, ya que $q' = abs\_call(abstract^*(q, S_0), s) = abs\_call(abstract^*((q_1, \ldots, q_p), s) = (q_1, \ldots, q_p, s)$ y, por tanto, $terms(q') = \{q_1, \ldots q_p\}$ $\cup\{s\}$. Ahora, por la hipótesis de inducción, los términos de $terms(q) \cup S_0$ están cerrados con respecto a $\{q_1, \ldots, q_p\}$. Por tanto, por el Lema 7.2.4, los términos de $terms(q) \cup S_0 \cup \{s\}$ están cerrados con respecto a $terms(abstract^*(q, S_0)) \cup \{s\} = \{q_1, \ldots, q_p\} \cup \{s\} = terms(q')$.
Consideramos ahora el caso 2). Entonces, $q' = abs\_call(abstract^*(q, S_0), s) = abs\_call((q_1, \ldots, q_p), s) = abstract^*((q_1, \ldots, q_p), terms(\widehat{\theta}))$, donde $q_i\theta = s$. Puesto que $\mathcal{M}_{terms(\widehat{\theta})} <_{mul} \mathcal{M}_{\{s\}}$ entonces, por la hipótesis de inducción, los términos de $\{q_1, \ldots, q_p\} \cup terms(\widehat{\theta})$ están cerrados con respecto a $terms(q')$. Ahora, ya que $\{s\}$ está cerrado con respecto a $\{q_i\} \cup terms(\widehat{\theta})$, por la Definición 6.3.2 y el Lema 7.3.9, los términos de $\{q_1, \ldots, q_p\} \cup \{s\}$ están cerrados con respecto a $terms(q')$, lo que completa la prueba.
Por último, el caso 3) se demuestra con un argumento similar al caso anterior, haciendo uso del hecho de que, por la Definición 6.3.2, los términos de $\{q_i, s\}$ están cerrados con respecto a $\{w\} \cup terms(\widehat{\theta}_1 \cup \widehat{\theta}_2)$. □

El siguiente teorema establece la terminación global del proceso de EP, usando el operador $abstract^*$ sobre las configuraciones del dominio $Conf^*$.

Teorema 7.3.11 (terminación global)
El algoritmo de la Definición 7.2.3 termina cuando se usa el operador $abstract^*$ *sobre el dominio* $Conf^*$ *de configuraciones* PE^*.

DEMOSTRACIÓN. Consideremos la siguiente computación: $q_0 \longmapsto_{\mathcal{P}} \ldots \longmapsto_{\mathcal{P}} q_{n-1} \longmapsto_{\mathcal{P}} q_n$ originada por la llamada $\mathcal{P}(\mathcal{R}, g)$, donde $q_0 = abstract^*(nil, terms(g))$ y $q_i = abstract^*(q_{i-1}, terms(U_{\varphi}^{\trianglelefteq}(terms(q_{i-1}), \mathcal{R})))$, $i \geq 1$.

En primer lugar, demostramos que todas las subsecuencias de términos comparables en una configuración PE^* $q_i \equiv (q_{i1}, \ldots, q_{im_i})$ satisfacen la siguiente propiedad (en adelante, propiedad de no subsumción):

$$\forall j, k \in \{1, \ldots, m_i\}.\ (j < k \ \wedge\ comparable(q_{ij}, q_{ik})) \ \Rightarrow\ q_{ij} \not\trianglelefteq q_{ik}).$$

Esto es suficiente para probar el teorema, ya que entonces el resultado se sigue de los siguientes hechos:

- El número de subsecuencias de términos incomparables que se pueden formar usando la signatura del programa es finito.
- Por el Teorema 7.3.2 y la propiedad de no subsumción enunciada antes, las subsecuencias de términos comparables en cualquier configuración PE^* son finitas.
- Por el Teorema 7.3.6, los conjuntos de términos $terms(U_\varphi^{\trianglelefteq}(terms(q_i), \mathcal{R}))$, $i \geq 1$, son todos finitos y, por tanto, la computación de cada configuración PE^* es también finita.
- Por la Definición 7.3.9, las longitudes de las secuencias q_i nunca disminuyen a medida que i crece, es decir, para $i > 0$, $|q_{i-1}| \leq |q_i|$.

Ahora, demostramos el resultado por inducción sobre la longitud n de la computación.

Sea $n = 0$ y $q_0 = abstract^*(nil, terms(g))$. Realizamos la prueba por inducción bien fundada sobre $S = terms(g)$. Usamos el orden bien fundado total $<_{mul}$ introducido en la demostración del Lema 7.3.10.

Si $S = \emptyset$, entonces $q_0 = abstract^*(nil, S) = abstract^*(nil, \emptyset) = nil$ y el resultado es cierto.

Si $S = S_0 \cup \{s\}$, donde $S_0 = \{s_1, \ldots, s_{k-1}\}$, y la propiedad se cumple para todo S' tal que $\mathcal{M}_{S'} <_{mul} \mathcal{M}_S$ entonces, por la Definición 7.3.9, $q_0 = abstract^*(nil, S) = abstract^*(nil, S_0 \cup \{s\}) = abstract^*(abstract^*(nil, S_0), s)$ y, por la hipótesis de inducción, la secuencia de términos $q' = abstract^*(nil, S_0) = (q'_1, \ldots, q'_p)$, $p \geq 0$, cumple la propiedad de no subsumción. Ahora, necesitamos considerar tres casos.

Si s es una variable o un símbolo de constante $c \in \mathcal{C}$ entonces, por la Definición 7.3.9, se cumple $q_0 = abstract^*(abstract^*(nil, S_0), s) = abstract^*(nil, S_0)$ y el resultado se sigue directamente por la hipótesis de inducción.

Si s es de la forma $c(t_1, \ldots, t_n)$, $c \in \mathcal{C}$, $n > 0$, entonces se cumple $q_0 = abstract^*(abstract^*(nil, S_0), c(t_1, \ldots, t_n)) = abstract^*(abstract^*(nil, S_0), \{t_1, \ldots, t_n\})$

$= abstract^*(nil, S_0 \cup \{t_1, \ldots, t_n\})$. Puesto que $\mathcal{M}_{S_0 \cup \{t_1,\ldots,t_n\}} <_{mul} \mathcal{M}_{S_0 \cup \{c(t_1,\ldots,t_n)\}} = \mathcal{M}_S$, el resultado es inmediato por la hipótesis de inducción.

Si $s = f(t_1, \ldots, t_n), f \in \mathcal{F}, n \geq 0$, entonces se cumple $q_0 = abstract^*(abstract^*(nil, S_0), s) = abs\_call(abstract^*(nil, S_0), s)$. Asumamos que $abstract^*(nil, S_0)$ es igual a $(q'_1, \ldots, q'_p)$, $p > 0$, ya que el caso en el que $abstract^*(nil, S_0)$ es igual a nil es trivial. Distinguimos ahora tres casos, que se corresponden con los tres casos de la función abs_call de la Definición 7.3.9, concretamente: $abs\_call((q'_1, \ldots, q'_p), s) =$

$$\begin{cases} 1)\ (q'_1, \ldots, q'_p, s) & \text{si } \nexists i \in \{1, \ldots, p\}.\ (comparable(q'_i, s) \text{ and } q'_i \trianglelefteq s). \\ 2)\ abstract^*((q'_1, \ldots, q'_p), terms(\widehat{\theta})) & \text{donde } \exists\ i \in \{1, \ldots, p\}.\ q'_i\theta = s. \\ 3)\ abstract^*(q'', s') & \text{donde } \exists\ i \in \{1, \ldots, p\}.\ msg(\{q'_i, s\}) = \langle w, \{\theta_1, \theta_2\}\rangle \\ & \quad s \text{ no es una instancia de } q'_i, \\ & \quad q'' \equiv (q'_1, \ldots, q'_{i-1}, q'_{i+1}, \ldots, q'_p), \text{ y} \\ & \quad s' = \{w\} \cup terms(\widehat{\theta}_1 \cup \widehat{\theta}_2). \end{cases}$$

Consideremos el caso 1). Entonces, $q_0 = abs\_call(abstract^*(nil, S_0), s) = abs\_call((q'_1, \ldots, q'_p), s) = (q'_1, \ldots, q'_p, s)$, y se cumplen los siguientes hechos: i) por la hipótesis de inducción, los términos comparables de $(q'_1, \ldots, q'_p)$ satisfacen la propiedad de no subsumción, y ii) por la premisa del caso 1), el término s considerado no infringe la propiedad de no subsumción con respecto a ninguno de los términos comparables de $(q'_1, \ldots, q'_p)$.

Consideremos ahora el caso 2). Entonces, $q_0 = abs\_call(abstract^*(nil, S_0), s) = abs\_call((q'_1, \ldots, q'_p), s) = abstract^*((q'_1, \ldots, q'_p), terms(\widehat{\theta})) = abstract^*(abstract^*(nil, S_0), terms(\widehat{\theta})) = abstract^*(nil, S_0 \cup terms(\widehat{\theta}))$, con $q'_i\theta = s$. Puesto que $\mathcal{M}_{S_0 \cup terms(\widehat{\theta})} <_{mul} \mathcal{M}_S$ entonces, por la hipótesis de inducción, los términos comparables de q_0 satisfacen la propiedad de no subsumción.

El caso 3) se demuestra siguiendo un argumento similar al del caso 2), ya que $\mathcal{M}_{(S_0 - \{q'_i\}) \cup \{w\} \cup terms(\widehat{\theta}_1 \cup \widehat{\theta}_2)} <_{mul} \mathcal{M}_S$.

Consideremos, finalmente, el caso inductivo $n > 0$. Asumamos $q_n = abstract^*(q_{n-1}, terms(U^{\trianglelefteq}_{\varphi}(terms(q_{n-1}), \mathcal{R})))$. Por la hipótesis de inducción, cada subsecuencia de términos comparables de q_{n-1} satisface la propiedad de no subsumción, con lo que la demostración es perfectamente análoga a la del caso base. $\square$

En este punto, disponemos de una instancia del algoritmo genérico de EP que garantiza, de forma automática, la condición de cierre para el programa transformado, así como la terminación del proceso de especialización. A continuación, mostramos como nuestro método es capaz de asegurar la terminación, consiguiendo aún obtener un buen nivel de especialización en el programa transformado.

7.3.3 Especialización *vs* Terminación

Una cuestión esencial a resolver en cualquier método de EP es el problema de la terminación. Existen gran variedad de métodos de EP que consiguen, en general, un nivel de especialización elevado, pero que no aseguran la terminación del método en todos los casos (ver e.g. [BL89, BL90, GS94, SGJ94]). Por el contrario, imponer restricciones demasiado fuertes (para garantizar la terminación del proceso) puede resultar a veces en una pérdida total de especialización.

Siendo nuestro método paramétrico respecto a la relación de *narrowing* que construye los árboles de búsqueda, para potenciar los efectos de la especialización vamos a considerar, en este apartado, la relación de *narrowing* condicional con normalización. Este es un refinamiento completo de la relación de *narrowing* para programas confluentes y decrecientes [Hus85]. La idea consiste en normalizar el objetivo ecuacional antes de aplicar cada paso de *narrowing*. Los resultados de corrección y completitud para la EP de programas, establecidos en el capítulo anterior, se pueden extender usando técnicas estándar para considerar la estrategia de *narrowing* condicional con normalización. El siguiente ejemplo muestra cómo la fase de normalización es capaz de incrementar notablemente la especialización del programa residual sin riesgo para la terminación. De esta forma, el ejemplo pretende evidenciar cómo nuestro esquema permite evaluaciones capaces de conseguir eficazmente ambos objetivos: terminación y especialización.

Ejemplo 30 *El siguiente programa $\mathcal{R}$ comprueba si una secuencia de caracteres es un palíndromo, haciendo uso de una función de inversión con parámetro acumulador auxiliar:*

$$\begin{array}{rcl}
palindrome(x) & \rightarrow & true \ \Leftarrow \ reverse(x) = x \\
reverse(x) & \rightarrow & rev(x, nil) \\
rev(nil, y_s) & \rightarrow & y_s \\
rev(x : x_s, y_s) & \rightarrow & rev(x_s, x : y_s)
\end{array}$$

Consideremos el término inicial $palindrome(1 : 2 : x)$. Construimos el programa especializado mediante el algoritmo de la Definición 7.2.3, haciendo uso de la regla de desplegado $U_\varphi^{\trianglelefteq}$ de la Definición 7.3.7, el operador de abstracción abstract* de la Definición 7.3.9 y la relación de narrowing condicional con normalización.
La configuración inicial es:

$$q_1 = abstract^*(nil, palindrome(1 : 2 : x_s)) = palindrome(1 : 2 : x_s).$$

La evaluación parcial de $palindrome(1 : 2 : x_s)$ en $\mathcal{R}$ es el siguiente programa $\mathcal{R}_1$:

$$palindrome(1 : 2 : x : x_s) \ \rightarrow \ true \ \Leftarrow \ rev(x_s, x : 2 : 1 : nil) = 1 : 2 : x : x_s$$

donde $terms(\mathcal{R}_1) = \{true, rev(x_s, x:2:1:nil), 1:2:x:x_s\}$. *Entonces, aplicando el operador de abstracción, la nueva configuración es:*

$$\begin{array}{rcl} q_2 & = & abstract^*(palindrome(1:2:x_s), \{true, rev(x_s, x:2:1:nil), 1:2:x:x_s\}) \\ & = & (palindrome(1:2:x_s), rev(x_s, x:2:1:nil)). \end{array}$$

Computamos ahora la EP de $rev(x_s, x:2:1:nil)$ *en* $\mathcal{R}$, *generando el programa* $\mathcal{R}_2$*:*

$$\begin{array}{rcl} rev(nil, x:2:1:nil) & \rightarrow & x:2:1:nil \\ rev(x':x'_s, x:2:1:nil) & \rightarrow & rev(x'_s, x':x:2:1:nil) \end{array}$$

donde $terms(\mathcal{R}_2) = \{x:2:1:nil, rev(x'_s, x':x:2:1:nil)\}$. *La nueva configuración es, por tanto:*

$$\begin{array}{rcl} q_3 & = & abstract^*((palindrome(1:2:x_s), rev(x_s, x:2:1:nil)), \\ & & \qquad\qquad \{x:2:1:nil, rev(x'_s, x':x:2:1:nil)\}) \\ & = & (palindrome(1:2:x_s), rev(x_s, x_1:x_2:x_3:y_s)). \end{array}$$

Ahora, la EP del término $rev(x_s, x_1:x_2:x_3:y_s)$ *en* $\mathcal{R}$ *es el siguiente programa* $\mathcal{R}_3$*:*

$$\begin{array}{rcl} rev(nil, x_1:x_2:x_3:y_s) & \rightarrow & x_1:x_2:x_3:y_s \\ rev(x:x_s, x_1:x_2:x_3:y_s) & \rightarrow & rev(x_s, x:x_1:x_2:x_3:y_s) \end{array}$$

donde $terms(\mathcal{R}_3) = \{x_1:x_2:x_3:y_s, rev(x_s, x:x_1:x_2:x_3:y_s)\}$. *Por último, obtenemos la siguiente configuración:*

$$\begin{array}{rcl} q_4 & = & abstract^*((palindrome(1:2:x_s), rev(x_s, x_1:x_2:x_3:y_s)), \\ & & \qquad\qquad \{x_1:x_2:x_3:y_s, rev(x_s, x:x_1:x_2:x_3:y_s)\}) \\ & = & (palindrome(1:2:x_s), rev(x_s, x_1:x_2:x_3:y_s)) \;\equiv\; q_3. \end{array}$$

Por tanto, el programa especializado $\mathcal{R}'$ *resultante de la EP de* $\mathcal{R}$ *con respecto al conjunto de términos* $S' = \{palindrome(1:2:x_s), rev(x_s, y_s)\}$ *es:*

$$\begin{array}{rcl} palindrome(1:2:x:x_s) & \rightarrow & true \;\Leftarrow\; rev(x_s, x:2:1:nil) = 1:2:x:x_s \\ rev(nil, x_1:x_2:x_3:y_s) & \rightarrow & x_1:x_2:x_3:y_s \\ rev(x:x_s, x_1:x_2:x_3:y_s) & \rightarrow & rev(x_s, x:x_1:x_2:x_3:y_s) \end{array}$$

en el que algunas derivaciones innecesarias han sido eliminadas en tiempo de especialización. Se debe notar que todas las computaciones sobre la parte estática de la estructura de entrada han sido ya realizadas: en el programa especializado $\mathcal{R}'$, *los elementos conocidos de la lista que se pasa como argumento de la función palindrome han sido transferidos a la lista del segundo argumento de la función rev. La serie de inferencias lógicas necesarias para realizar esta tarea en tiempo de ejecución, cuyo número es lineal con respecto al número de elementos conocidos de la lista, han sido ya realizadas en tiempo de especialización.*

El procedimiento de supercompilación positiva de [GS94, Sør94, SGJ94] no termina sobre este ejemplo, debido a la generación infinita de nuevas llamadas a función, las cuales, debido al crecimiento particular del parámetro acumulador, nunca son una instancia de alguna llamada anterior. Los procedimientos para la deducción parcial de [BH93, BL90] tienen el mismo problema en la correspondiente versión lógica del programa. Los métodos de [MG95, SG95], sin embargo, aseguran la terminación para este ejemplo, al igual que nuestro método.

En la siguiente sección, abordamos el problema de cómo garantizar la condición de independencia de forma automática.

7.4 Renombramiento e Independencia

Haciendo uso de las instancias de la regla de desplegado y del operador de abstracción introducidas en la sección anterior, el algoritmo formalizado por la Definición 7.2.3 permite generar, de forma automática y en tiempo finito, un programa especializado que satisface la condición de cierre. Dicho algoritmo puede verse como el núcleo de un evaluador parcial que, teniendo en cuenta la estrategia de *narrowing* empleada, podría completarse de forma adecuada. Un aspecto a tener en cuenta es si resulta posible o no garantizar que el conjunto de términos producido por el procedimiento de EP es independiente. En el campo de la deducción parcial de programas lógicos, se han desarrollado diferentes técnicas de renombramiento que consiguen la independencia de forma automática [BH93, BL90, Gal93, GB90]. En algunos casos, la técnica de renombramiento se aplica de forma dinámica durante el proceso de especialización ([BH93], Sección 3), mientras que habitualmente se considera un post-proceso que debe emplearse sobre el programa resultante de la especialización (ver, e.g. [Gal93, GB90] y la Sección 4 de [BH93]). La técnica de supercompilación positiva obtiene un efecto similar al de [Gal93], gracias a la forma dinámica en que se extraen las definiciones para las funciones especializadas a partir del grafo construido por el mecanismo de *driving* [GK93, Tur88]. Presentamos, a continuación, un ejemplo de especialización que ilustra las dificultades que se presentan en el proceso de renombramiento cuando se considera la EP de programas lógico–funcionales basada en (alguno de los refinamientos de) la relación de *narrowing*. El problema del renombramiento resulta especialmente interesante en nuestro contexto, ya que las funciones que aparecen como argumento de las llamadas en modo alguno son estructuras "muertas" sino que pueden generar nuevas llamadas a función.

El siguiente problema se usa frecuentemente en la literatura relacionada con la EP de programas, como test para analizar si un método es capaz de eliminar estructuras de datos intermedias, así como de evitar el recorrido múltiple de una misma estructura en tiempo de ejecución. Este mismo efecto se consigue, por ejemplo, con los procedimientos de *deforestación* [Wad88], *tupling* [Chi93] y la supercompilación positiva de [Sør94].

Ejemplo 31 *Consideremos de nuevo el programa append/2 del Ejemplo 29:*

$$\begin{array}{rcl} append(nil, y_s) & \rightarrow & y_s \\ append(x : x_s, y_s) & \rightarrow & x : append(x_s, y_s) \end{array}$$

y la llamada $append(append(x_s, y_s), z_s)$ que permite computar la concatenación de las tres listas x_s, y_s y z_s. Si evaluamos dicha expresión respecto al programa append/2, la computación procede concatenando las dos primeras listas y, posteriormente, concatenando a este resultado la tercera lista. Evidentemente, esta forma de computar es ineficiente en cuanto a la complejidad espacial (se crea una estructura de datos intermedia que no forma parte del resultado final), y también en cuanto a la complejidad temporal (la primera lista se recorre dos veces).

Vamos a especializar el programa append/2 mediante el algoritmo que se presenta en la Sección 7.3, usando una relación de narrowing condicional con normalización para construir los árboles de búsqueda (parciales). Comenzando con la configuración inicial $q = append(append(x_s, y_s), z_s)$, obtenemos los árboles de la Figura 7.2. Nótese que append ha sido abreviado a "a" en la figura. Además, los pasos de narrowing se han representado con líneas, mientras que el (único) paso de normalización se ha representado con una flecha. El programa especializado $\mathcal{R}'$ es el siguiente:

$$\begin{array}{rcl} append(append(nil, y_s), z_s) & \rightarrow & append(y_s, z_s) \\ append(append(x : x_s, y_s), z_s) & \rightarrow & x : append(append(x_s, y_s), z_s) \\ append(nil, z_s) & \rightarrow & z_s \\ append(y : y_s, z_s) & \rightarrow & y : append(y_s, z_s) \end{array}$$

El nuevo programa $\mathcal{R}'$ es capaz de concatenar las tres listas recorriendo sus argumentos una sola vez, y sin construir ningún tipo de estructura intermedia.

En este ejemplo particular, el efecto deseado se ha conseguido, en parte, gracias a la normalización. Sin normalizar los nodos del árbol, el orden de subsumción se habría satisfecho demasiado pronto (en la rama de la derecha del árbol superior de la Figura 7.2). En este caso, el algoritmo habría calculado el msg del término de partida y el término en este nodo, resultando el término $a(x, y)$, lo cual implica que se habría reproducido el programa original y toda la especialización se habría perdido.

En [SGJ94, Sør94] se conjetura que el problema de pérdida de especialización, ilustrado por el ejemplo anterior, sólo puede evitarse gracias al uso de una estrategia perezosa en la construcción de los árboles de búsqueda. Pensamos que dicha conjetura no es cierta, ya que nuestro método consigue el efecto deseado gracias a la normalización, independientemente de la estrategia empleada por *narrowing* para realizar las computaciones.

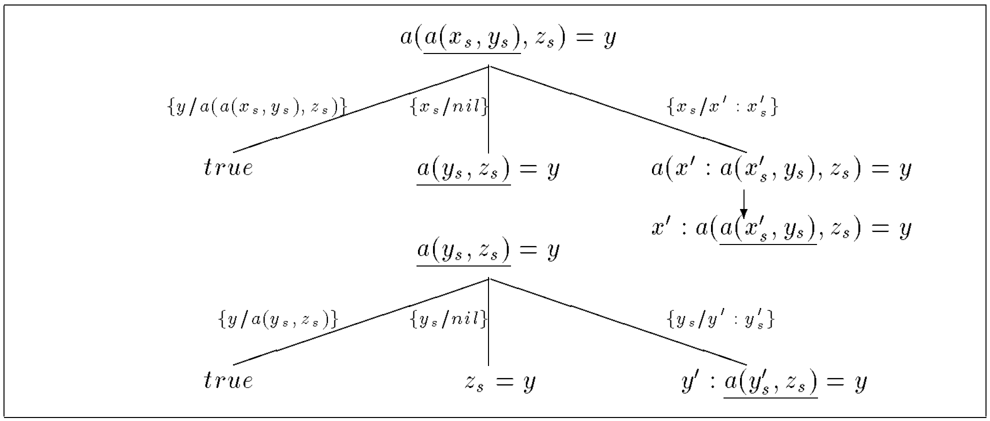

Figura 7.2: Arboles parciales para $a(a(x_s, y_s), z_s) = y$, $a(x_s, y_s) = y$.

Aquí podemos observar que el conjunto de términos parcialmente evaluados:

$$\{append(append(x_s, y_s), z_s), append(x_s, y_s)\}$$

no es independiente, lo cual hace posible que se produzcan algunas derivaciones innecesarias (que computan respuestas que no se pueden obtener en el programa original). Para restablecer la independencia del conjunto de términos parcialmente evaluados y conseguir así un programa residual que no produzca respuestas adicionales, sería posible desarrollar un post-proceso de renombramiento en el estilo del definido por [BH93]. Lo que sigue es una extensión informal del método de [BH93] para el caso de los programas lógico–funcionales.

En primer lugar, dado un programa $\mathcal{R}$ y un objetivo g, haciendo uso del algoritmo de la Definición 7.2.3 se genera el conjunto de términos S, cuya EP da lugar al programa especializado $\mathcal{R}'$. Ahora, en el caso de que el conjunto S no sea independiente, asociamos a cada término $s \in S$ una función $f^s(x_1, \ldots, x_s)$, donde f^s es un símbolo de función nuevo (i.e. que no aparece en la signatura del programa ni ha sido empleado anteriormente) y $\{x_1, \ldots, x_s\}$ son las variables distintas del término s en el mismo orden de su primera aparición. Entonces, podemos renombrar cada resultante[3] $(s\theta \rightarrow \rho \Leftarrow C)$ del programa $\mathcal{R}'$ de la forma:

$$f^s(x_1, \ldots, x_s)\theta \rightarrow \rho' \Leftarrow C'$$

donde ρ' se construye como sigue (el proceso para obtener C' es totalmente análogo). Sea $\{u_1, \ldots, u_n\}$ un conjunto de ocurrencias de cierre para ρ en S, i.e. para todo

[3]Para simplificar la notación, asumimos que cada regla lleva "anotada" la información de cuál es el término de cuya evaluación parcial ha resultado. De esta forma, expresamos los resultantes en la forma: $(s\theta \rightarrow \rho \Leftarrow C)$, donde s es el término que aparece en el objetivo inicial $(s = y)$ de la derivación que originó la regla.

subtérmino $\rho_{|u_i}$ existe un término $s_i \in S$ tal que $\rho_{|u_i} = s_i\theta_i$, $i = 1, \ldots, n$. Entonces el término renombrado ρ' se obtiene sustituyendo recursivamente los subtérminos $\rho_{|u_i}$ por la correspondiente función $f^{s_i}(x_1, \ldots, x_{s_i})$, aplicando θ_i al término resultante de cada sustitución.

Si aplicamos dicho procedimiento al programa $\mathcal{R}'$ del Ejemplo 31, considerando el siguiente renombramiento para los términos del conjunto (no independiente) S:

$$\begin{array}{lcl} append(append(x_s, y_s), z_s) & \Longrightarrow & append\_3(x_s, y_s, z_s) \\ append(x_s, y_s) & \Longrightarrow & append\_2(x_s, y_s) \end{array}$$

se obtiene el programa renombrado $\mathcal{R}''$:

$$\begin{array}{rcl} append\_3(nil, y_s, z_s) & \rightarrow & append\_2(y_s, z_s) \\ append\_3(x : x_s, y_s, z_s) & \rightarrow & x : append\_3(x_s, y_s, z_s) \\ append\_2(nil, z_s) & \rightarrow & z_s \\ append\_2(y : y_s, z_s) & \rightarrow & y : append\_2(y_s, z_s) \end{array}$$

que produce el mismo conjunto de respuestas computadas que el programa original $append/2$ para cualquier objetivo que esté cerrado (módulo renombramiento) por el conjunto de términos $\{append(append(x_s, y_s), z_s), append(x_s, y_s)\}$.

Es importante observar que la técnica de renombramiento, tal como se ha expuesto, no es en general un proceso determinista. La causa del indeterminismo es, esencialmente, el hecho de que un término S-cerrado puede tener asociados distintos conjuntos de cierre con respecto a S (excepto si S es independiente, en cuyo caso la transformación de renombramiento no sería necesaria).

La corrección de la técnica de renombramiento esbozada debe demostrarse para cada estrategia de *narrowing* particular. En [AFJV96a, AFJV96b] puede encontrarse la definición formal de una técnica de renombramiento que sigue las líneas del método aquí expuesto, y cuya corrección se demuestra bajo las condiciones de completitud de la relación de *narrowing* perezoso.

Una manera habitual de comprobar la potencia de un evaluador parcial, consiste en realizar el así llamado "test KMP". Terminamos este capítulo con una última seccion, en la que mostramos que nuestro método es capaz de superar satisfactoriamente dicho test.

7.5 Test KMP

Un ejemplo estándar en la literatura sobre evaluación parcial, es la generación de un programa eficiente para el emparejamiento de patrones a partir de la EP de una versión simple (pero ineficiente) del programa, especializado para un patrón fijo dado [Jon94, SGJ94]. El siguiente programa $\mathcal{R}$, comprueba si una cadena de caracteres p aparece dentro de otra cadena s comparando iterativamente p con un prefijo de s. En caso de que no emparejen, se elimina el primer elemento de s y el proceso se reinicia con el resto de elementos de s. Por simplificar, consideramos que el emparejamiento se define únicamente sobre cadenas de bits, i.e. cadenas que sólo están compuestas de ceros y unos.

$$match(p, s) \rightarrow loop(p, s, p, s)$$

$$\begin{array}{rcll}
loop(nil, ss, op, os) & \rightarrow & true & \\
loop(p : pp, nil, op, os) & \rightarrow & false & \\
loop(p : pp, p : ss, op, os) & \rightarrow & loop(pp, ss, op, os) & \%\ \texttt{continue} \\
loop(p : pp, s : ss, op, os) & \rightarrow & next(op, os) \Leftarrow (p = s) = false & \%\ \texttt{shift string}
\end{array}$$

$$\begin{array}{rcll}
next(op, nil) & \rightarrow & false & \\
next(op, s : ss) & \rightarrow & loop(op, ss, op, ss) & \%\ \texttt{restart loop} \\
(0 = 1) & \rightarrow & false & \\
(1 = 0) & \rightarrow & false &
\end{array}$$

El código de este programa es muy ineficiente, ya que varios elementos de la cadena pueden ser explorados varias veces. Concretamente, dado un patrón fijo como el 001, si tras comprobar la aparición de la secuencia 00 en la cadena objetivo aparece nuevamente un cero, el programa reinicia la búsqueda del patrón completo 001 desde la segunda posición de la cadena, mientras que sería suficiente con buscar un 1 en la cuarta posición. La capacidad de especialización de un método de transformación se puede verificar comprobando si es capaz de obtener automáticamente un algoritmo similar al desarrollado por Knuth, Morris y Pratt [KMP77], cuando se especializa el anterior programa $\mathcal{R}$ respecto a un patrón dado. A menudo, se hace referencia a esta prueba como el "test KMP". El ejemplo es particularmente interesante, porque es un tipo de transformación que no se puede obtener automáticamente usando la técnica de deforestación, ni por las técnicas de EP tradicionales (para programas funcionales) [SGJ94]. La deducción parcial de programas lógicos, así como la supercompilación positiva, sí son capaces de superar el test (ver [Sør94] para referencias). Mostramos, a continuación, cómo nuestro método es también capaz de resolver satisfactoriamente el problema.

Suponemos un patrón 001 de entrada fijado, con respecto al cual realizamos la especialización. En el proceso de EP empleamos el algoritmo de la Definición 7.2.3, instanciado con los siguientes parámetros:

- la relación de *narrowing* condicional con normalización, para construir los árboles de búsqueda;

- las configuraciones PE^* (Definición 7.3.8), junto con el operador de abstracción $abstract^*$ (Definición 7.3.9), para garantizar la terminación global; y

- una regla de desplegado consistente en expandir un sólo nivel del árbol de búsqueda (al igual que en el Ejemplo 27), para garantizar la terminación local.

Entonces, la EP de $\mathcal{R}$ con respecto al término $match(001, s)$ produce el siguiente programa especializado $\mathcal{R}'$[4]:

$$
\begin{array}{rcl}
match(001, s) & \rightarrow & loop(001, s, 001, s) \\
\\
loop(001, 0 : ss, 001, 0 : ss) & \rightarrow & loop(01, ss, 001, 0 : ss) \\
loop(001, s : ss, 001, s : ss) & \rightarrow & loop(001, ss, 001, ss) \Leftarrow (0 = s) = false \\
\\
loop(01, 0 : ss', 001, 00 : ss') & \rightarrow & loop(1, ss', 001, 00 : ss') \\
loop(01, s' : ss', 001, 0 : s' : ss') & \rightarrow & loop(001, s' : ss', 001, s' : ss') \Leftarrow (0 = s') = false \\
\\
loop(1, 1 : ss'', 001, 001 : ss'') & \rightarrow & true \\
loop(1, s'' : ss'', 001, 00 : s'' : ss'') & \rightarrow & loop(01, s'' : ss'', 001, 0 : s'' : ss'') \Leftarrow (1 = s'') = false
\end{array}
$$

Es posible eliminar algunas redundancias del código generado aplicando el postproceso de renombramiento esbozado en la sección anterior, dando lugar al siguiente programa residual $\mathcal{R}''$:

$$
\begin{array}{rcl}
match'(s) & \rightarrow & loop\_001(s) \\
\\
loop\_001(0 : ss) & \rightarrow & loop\_01(ss) \\
loop\_001(s : ss) & \rightarrow & loop\_001(ss) \Leftarrow (0 = s) = false \\
\\
loop\_01(0 : ss) & \rightarrow & loop\_1(ss) \\
loop\_01(s : ss) & \rightarrow & loop\_001(s : ss) \Leftarrow (0 = s) = false \\
\\
loop\_1(1 : ss) & \rightarrow & true \\
loop\_1(s : ss) & \rightarrow & loop\_01(s : ss) \Leftarrow (1 = s) = false
\end{array}
$$

La cantidad de especialización conseguida por $\mathcal{R}''$ es esencialmente análoga a la obtenida por el algoritmo de [SG95]. La complejidad del algoritmo es $O(n)$, donde n es la longitud de la cadena s. La complejidad del programa original $\mathcal{R}$ es $O(m \times n)$, donde m es la longitud del patrón p. En esencia, éste es un programa para el emparejamiento de patrones en el estilo KMP.

[4]No consideramos, por simplicidad, las reglas que definen reducciones a $false$.

Conclusiones y Trabajo Futuro

En esta tesis formalizamos las bases semánticas para algunas optimizaciones importantes de los lenguajes lógico–funcionales. Las principales aportaciones se pueden resumir tal y como sigue.

En primer lugar, se ha formalizado un esquema para el análisis estático de programas lógico-funcionales, que permite obtener una aproximación finita del comportamiento operacional del programa, entendido en términos de un *observable* basado en el conjunto de las respuestas computadas. El análisis se define de manera paramétrica respecto a la estrategia de *narrowing* y una versión "abstracta" del programa. Se ha demostrado la corrección y terminación del análisis resultante para aquellas estrategias que sean independientes del entorno. El análisis es de propósito bastante general y permite distintas aplicaciones, algunas de las cuales han sido desarrolladas y su eficacia probada a través de una implementación preliminar del sistema.

En segundo lugar, hemos introducido una técnica de evaluación parcial de programas lógico–funcionales y hemos demostrado la corrección y completitud de la transformación. Ésta se basa directamente en el uso del propio mecanismo operacional de *narrowing* durante el proceso de especialización, y constituye la primera aproximación totalmente automática, correcta y finita para la evaluación parcial de programas lógico–funcionales.

Son varias las líneas de trabajo abiertas por esta tesis. El autor se declara especialmente interesado en aquéllas que están más relacionadas con la segunda parte del trabajo. La formalización de técnicas basadas en la semántica para la especialización de programas lógico–funcionales es un campo de investigación incipiente y atractivo. Tenemos previsto extender el trabajo desarrollado teniendo en cuenta las siguientes ideas, que son argumento de varias tesis doctorales estrechamente relacionadas con la nuestra y actualmente en desarrollo en nuestro grupo:

- El desarrollo de un álgebra de programas lógico–funcionales, que englobe la composicionalidad AND y OR en un marco único, y la formulación de una semántica para los programas combinados con los operadores del álgebra. Así, la técnica de análisis se puede incorporar en un esquema más general que permita disfrutar de los beneficios aportados por ambos tipos de composicionalidad.

- La técnica de supercompilación de Turchin no sólo propaga la información positiva, sino que también es capaz de propagar información negativa a través del entorno. Esta información negativa restringe los valores que pueden tomar las variables durante la EP, permitiendo así obtener una especialización más precisa [SGJ94, Tur86]. Pensamos que en nuestro caso se podría conseguir un efecto similar, usando algún procedimiento de *narrowing* con (restricciones de) desigualdad, como los definidos en [AGL94, Fer92, RF93].

- Otra línea de trabajo interesante, consiste en definir instancias del método genérico de EP para estrategias de *narrowing* eficientes, y demostrar la corrección y completitud de la transformación para dichas estrategias. El comportamiento del método es independiente del carácter impaciente o perezoso del procedimiento de *narrowing* utilizado. Podemos, así, definir un evaluador parcial *call-by-name* (basado en un cálculo de *narrowing* condicional perezoso), cuya eficiencia ha de ser indudablemente superior a la del evaluador parcial presentado aquí. Esta instancia podría emplearse, en la práctica, para la optimización del proceso de compilación del lenguaje lógico-funcional Babel [MR92], cuya semántica operacional está basada en una forma de *narrowing* perezoso. Hemos comenzado ya algunas investigaciones en esta línea, cuyos primeros resultados se recogen en [AFJV96a, AFJV96b].

- Nuestros resultados también son relevantes para la definición de una semántica composicional para programas lógico–funcionales, que sea de utilidad para el análisis y transformación de programas modulares. Para tal fin, nos proponemos estudiar la definición de una semántica por desplegado, haciendo uso de evaluación parcial con una estrategia de desplegado que detiene las derivaciones que alcanzan símbolos de función *abiertos* (i.e. símbolos definidos en otro módulo del programa).

Bibliografía

[AEH94] S. Antoy, R. Echahed, and M. Hanus. A Needed Narrowing Strategy. In *Proc. 21st ACM Symp. on Principles of Programming Languages, Portland*, pages 268–279, 1994.

[AFJV96a] M. Alpuente, M. Falaschi, P. Julián, and G. Vidal. Call-by-Name Partial Evaluation of Functional–Logic Programs. In P. Lucio, M. Martelli, and M. Navarro, editors, *Proc. of the 1996 Joint Conf. on Declarative Programming, APPIA-GULP-PRODE'96*, pages 17–28. U. del País Vasco, 1996.

[AFJV96b] M. Alpuente, M. Falaschi, P. Julián, and G. Vidal. Call-by-Name Specialization of Functional–Logic Programs. In Tetsuo Ida and Yike Guo, editors, *Proc. of JICSLP'96 Post Conference Workshop on Multi-Paradigm Logic Programming, MPLP'96.* TU Berlin, 1996. To appear.

[AFL95] M. Alpuente, M. Falaschi, and G. Levi. Incremental Constraint Satisfaction for Equational Logic Programming. *Theoretical Computer Science*, 142(1):27–57, 1995.

[AFM95] M. Alpuente, M. Falaschi, and F. Manzo. Analyses of Unsatisfiability for Equational Logic Programming. *Journal of Logic Programming*, 22(3):221–252, 1995.

[AFRV93a] M. Alpuente, M. Falaschi, M.J. Ramis, and G. Vidal. Narrowing Approximations as an Optimization for Equational Logic Programs. In J. Penjam and M. Bruynooghe, editors, *Proc. of PLILP'93, Tallinn (Estonia)*, pages 391–409. Springer LNCS 714, 1993.

[AFRV93b] M. Alpuente, M. Falaschi, M.J. Ramis, and G. Vidal. Using Narrowing Approximations to Optimize Equational Logic Programs. In D. Saccà, editor, *Proc. of the 8th Italian Conf. on Logic Programming, GULP'93, Gizzeria Lido (Italy)*, pages 127–142. Università di Calabria, 1993.

[AFRV94] M. Alpuente, M. Falaschi, M.J. Ramis, and G. Vidal. A Compositional Semantics for Conditional Term Rewriting Systems. In H.E. Bal, editor, *Proc. of 6th IEEE Int'l Conf. on Computer Languages, ICCL'94*, pages 171–182. IEEE, New York, 1994.

[AFV93] M. Alpuente, M. Falaschi, and G. Vidal. Incremental Equational Constraint Analyses. In V. Dahl and D. Miller, editors, *Proc. of the ILPS'93, Vancouver, Canada.* The MIT Press, Cambridge, MA, 1993. (Poster).

[AFV94] M. Alpuente, M. Falaschi, and G. Vidal. Compositional Analysis for Equational Horn Programs. In G. Levi and M. Rodríguez-Artalejo, editors, *Proc. of Fifth Int'l Conf. on Algebraic and Logic Programming, ALP'94*, pages 77–94. Springer LNCS 850, 1994.

[AFV95] M. Alpuente, M. Falaschi, and G. Vidal. Narrowing-driven Specialization of Functional Logic Programs. Technical Report DSIC-II/5/95, UPV, 1995. Submitted for publication.

[AFV96a] M. Alpuente, M. Falaschi, and G. Vidal. A Compositional Semantic Basis for the Analysis of Equational Horn Programs. *Theoretical Computer Science*, 165(1):97–131, Sept. 1996.

[AFV96b] M. Alpuente, M. Falaschi, and G. Vidal. Narrowing-driven Partial Evaluation of Functional Logic Programs. In H. Riis Nielson, editor, *Proc. of the 6th European Symp. on Programming, ESOP'96*, pages 45–61. Springer LNCS 1058, 1996.

[AGL94] P. Arenas, A. Gil, and F. López. Combining Lazy Narrowing with Disequality Constraints. In *Proc. of PLILP'94*, pages 385–399. Springer LNCS 844, 1994.

[ALN87] H. Aït-Kaci, P. Lincoln, and R. Nasr. Le Fun: Logic, equations, and Functions. In *Proc. of Fourth IEEE Int'l Symp. on Logic Programming*, pages 17–23. IEEE, New York, 1987.

[Alp91] M. Alpuente. *El Lenguaje CLP($\mathcal{H}/\mathcal{E}$): una Aproximación basada en Restricciones a la Integración de la Programación Lógica y Funcional.* PhD thesis, Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, 1991.

[AN86] H. Aït-Kaci and R. Nasr. LOGIN: a logic programming language with built-in inheritance. *Journal of Logic Programming*, 3:185–215, 1986.

[AN89] H. Aït-Kaci and R. Nasr. Integrating Logic and Functional Programming. *Lisp and Symbolic Computation*, 2(1):51–89, 1989.

[BC93] A. Bossi and N. Cocco. Basic Transformation Operations which preserve Computed Answer Substitutions of Logic Programs. *Journal of Logic Programming*, 16:47–87, 1993.

[BD77] R.M. Burstall and J. Darlington. A Transformation System for Developing Recursive Programs. *Journal of the ACM*, 24(1):44–67, 1977.

[BdSM92] M. Bruynooghe, D. de Schreye, and B. Martens. A General Criterion for Avoiding Infinite Unfolding. *New Generation Computing*, 11(1):47–79, 1992.

[BE86] D. Bert and R. Echahed. Design and implementation of a generic, logic and functional programming language. In *Proc. of First European Symp. on Programming, ESOP'86*, pages 119–132. Springer LNCS 213, 1986.

[BE94] D. Bert and R. Echahed. Integrating Disequations in the Algebraic and Logic Programming Language LPG. In *Proc. of the ICLP'94 Post-Conference Workshop on Integration of Declarative Paradigms.* MPI-I-94-224, Saarbrucken, 1994.

[BE95] D. Bert and R. Echahed. Abstraction of Conditional Term Rewriting Systems. In *Proc. of the Int'l Logic Programming Symposium, ILPS'95*, pages 162–176. The MIT Press, Cambridge, MA, 1995.

[Bec76] L. Beckman *et al.* A Partial Evaluator, and Its Use as a Programming Tool. *Artificial Intelligence*, 7(4):319–357, 1976.

[BEØ93] D. Bert, R. Echahed, and B.M. Østvold. Abstract Rewriting. In *Proc. of Third Int'l Workshop on Static Analysis, WSA'93*, pages 178–192. Springer LNCS 724, 1993.

[BGM88] P. Bosco, E. Giovannetti, and C. Moiso. Narrowing vs. SLD-resolution. *Theoretical Computer Science*, 59:3–23, 1988.

[BH93] K. Benkerimi and P.M. Hill. Supporting Transformations for the Partial Evaluation of Logic Programs. *Journal of Logic and Computation*, 3(5):469–486, 1993.

[BHA86] G. Burn, C. Hankin, and S. Abramsky. Strictness Analysis for Higher Order Functions. *Science of Computer Programming*, 7:249–278, 1986.

[BL86] M. Bellia and G. Levi. The relation between logic and functional languages. *Journal of Logic Programming*, 3:217–236, 1986.

[BL89] K. Benkerimi and J.W. Lloyd. A Procedure for the Partial Evaluation of Logic Programs. Technical Report TR-89-04, Department of Computer Science, University of Bristol, Bristol, England, May 1989.

[BL90] K. Benkerimi and J.W. Lloyd. A Partial Evaluation Procedure for Logic Programs. In S. Debray and M. Hermenegildo, editors, *Proc. of the 1990 North American Conf. on Logic Programming*, pages 343–358. The MIT Press, Cambridge, MA, 1990.

[Bol93] R. Bol. Loop Checking in Partial Deduction. *Journal of Logic Programming*, 16(1&2):25–46, 1993.

[Boy93] J. Boye. Avoiding Dynamic Delays in Functional Logic Languages. In J. Penjam and M. Bruynooghe, editors, *Proc. of PLILP'93, Tallinn (Estonia)*, pages 12–27. Springer LNCS 714, 1993.

[BPM93] J. Boye, J. Paakki, and J. Maluszynski. Synthesis of Directionality Information for Functional Logic Programs. In *Proc. of Third Int'l Workshop on Static Analysis, WSA'93*, pages 165–177. Springer LNCS 724, 1993.

[Bru91] M. Bruynooghe. A Practical Framework for the Abstract Interpretation of Logic Programs. *Journal of Logic Programming*, 10:91–124, 1991.

[BW95] A. Bockmayr and A. Werner. LSE Narrowing for Decreasing Conditional Term Rewrite Systems. In *Conditional Term Rewriting Systems, CTRS'94, Jerusalem*. Springer LNCS 968, 1995.

[CC77] P. Cousot and R. Cousot. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Proc. of Fourth ACM Symp. on Principles of Programming Languages*, pages 238–252, 1977.

[CC79] P. Cousot and R. Cousot. Systematic Design of Program Analysis Frameworks. In *Proc. of Sixth ACM Symp. on Principles of Programming Languages*, pages 269–282, 1979.

[CC92] P. Cousot and R. Cousot. Abstract Interpretation and Application to Logic Programs. *Journal of Logic Programming*, 13(2&3):103–179, 1992.

[CD93] C. Consel and O. Danvy. Partial Evaluation in Parallel. *LISP and Symbolic Computation*, 5(4):315–330, 1993.

[CDG93] M. Codish, S.K. Debray, and R. Giacobazzi. Compositional Analysis of Modular Logic Programs. In *Proc. of 20th Annual ACM Symp. on*

Principles of Programming Languages, pages 451–464. ACM, New York, 1993.

[CF92] P.H. Cheong and L. Fribourg. A survey of the implementations of narrowing. In J. Darlington and R. Dietrich, editors, *Declarative Programming. Workshops in Computing*, pages 177–187. Springer-Verlag and BCS, 1992.

[CFM94] M. Codish, M. Falaschi, and K. Marriott. Suspension Analyses for Concurrent Logic Programs. *ACM Transactions on Programming Languages and Systems*, 16(3):649–686, 1994.

[Chi93] W. Chin. Towards an Automated Tupling Strategy. In *Proc. of Partial Evaluation and Semantics-Based Program Manipulation, Copenhagen, Denmark, June 1993*, pages 119–132. ACM, New York, 1993.

[Cos91] V.S. Costa *et al.* The Andorra-I Engine: A Parallel Implementation of the Basic Andorra Model. In *Proc. of the 8th Int'l Conf. on Logic Programming, ICLP'91*, pages 825–839, 1991.

[CR91] J. Chabin and P. Réty. Narrowing directed by a graph of terms. In G. Goos and J. Hartmanis, editors, *Proc. of RTA'91*, pages 112–123. Springer LNCS 488, 1991.

[dBP90] F.S. de Boer and C. Palamidessi. On the Asynchronous Nature of Communication in Concurrent Logic Languages: A Fully Abstract Model Based on Sequences. In J.C.M. Baeten and J.W. Klop, editors, *Proc. of CONCUR'90*, pages 175–194. Springer LNCS 458, 1990.

[DG89] J. Darlington and Y. Guo. Narrowing and unification in functional programming: an evaluation mechanism for absolute set abstraction. In *Proc. of the Conf. on Rewrite Techniques and Applications, RTA'89*, pages 92–108. Springer LNCS 355, 1989.

[DJ90] N. Dershowitz and J.-P. Jouannaud. Rewrite Systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, pages 243–320. Elsevier, Amsterdam, 1990.

[DL86] D. DeGroot and G. Lindstrom, editors. *Logic Programming, Functions, Relations and Equations.* Prentice Hall, Englewood Cliffs, NJ, 1986.

[DL90] N. Dershowitz and N. Lindenstrauss. An Abstract Concurrent Machine for Rewriting. In H. Kirchner and W. Wechler, editors, *Proc. of Second Int'l Conf. on Algebraic and Logic Programming*, pages 318–331. Springer LNCS 463, 1990.

[DM85] P. Deransart and J. Maluszyński. Relating Logic Programs and Attribute Grammars. *Journal of Logic Programming*, 2:119–156, 1985.

[DP88a] J. Darlington and H. Pull. A Program Development Methodology Based on a Unified Approach to Execution and Transformation. In D. Bjørner, A.P. Ershov, and N.D. Jones, editors, *Proc. of the Int'l Workshop on Partial Evaluation and Mixed Computation*, pages 117–131. North-Holland, Amsterdam, 1988.

[DP88b] N. Dershowitz and A. Plaisted. Equational Programming. *Machine Intelligence*, 11:21–56, 1988.

[DR93] N. Dershowitz and U. Reddy. Deductive and Inductive Synthesis of Equational Programs. *Journal of Symbolic Computation*, 15:467–494, 1993.

[DS87] N. Dershowitz and G. Sivakumar. Solving Goals in Equational Languages. In S. Kaplan and J. Joaunnaud, editors, *Proc. of First Int'l Workshop on Conditional Term Rewriting*, pages 45–55. Springer LNCS 308, 1987.

[Ech88] R. Echahed. On completeness of narrowing strategies. In *Proc. of CAAP'88*, pages 89–101. Springer LNCS 299, 1988.

[EJ88] A.P. Ershov and N.D. Jones. Two Characterizations of Partial Evaluation and Mixed Computation. In D. Bjørner, A.P. Ershov, and N.D. Jones, editors, *Proc. of the Int'l Workshop on Partial Evaluation and Mixed Computation*, pages xxiii–xxix. North-Holland, Amsterdam, 1988.

[Ers77] A.P. Ershov. On the Partial Computation Principle. *Information Processing Letters*, 6(2):38–41, 1977.

[Ers78a] A.P. Ershov. Mixed Computation in the Class of Recursive Program Schemata. *Acta Cybernetica*, 4(1):19–23, 1978.

[Ers78b] A.P. Ershov. On the Essence of Compilation. In E.J. Neuhold, editor, *Formal Description of Programming Concepts*, pages 391–420. North-Holland, Amsterdam, 1978.

[Ers88] A.P. Ershov. Opening Key-note Speech. In D. Bjørner, A.P. Ershov, and N.D. Jones, editors, *Proc. of the Int'l Workshop on Partial Evaluation and Mixed Computation*, pages xxiii–xxix. North-Holland, Amsterdam, 1988.

[Fay79] M. Fay. First Order Unification in an Equational Theory. In *Proc of 4th Int'l Conf. on Automated Deduction*, pages 161–167, 1979.

[Fer92] M. Fernández. Narrowing Based Procedures for Equational Disunification. *Applicable Algebra in Engineering, Communication and Computing*, 3:1–26, 1992.

[Fri85] L. Fribourg. SLOG: a logic programming language interpreter based on clausal superposition and rewriting. In *Proc. of Second IEEE Int'l Symp. on Logic Programming*, pages 172–185. IEEE, New York, 1985.

[Fut71] Y. Futamura. Partial Evaluation of Computation Process – An Approach to a Compiler-Compiler. *Systems, Computers, Controls*, 2(5):45–50, 1971.

[Fut83] Y. Futamura. Partial Computation of Programs. In E. Goto *et al.*, editor, *RIMS Symposia on Software Science and Engineering, Kyoto, Japan, 1982*, pages 1–35. Springer LNCS 147, 1983.

[Fut88] Y. Futamura. Program Evaluation and Generalized Partial Computation. In *Proc. Int'l Conf. on Fifth Generation Computer Systems*, pages 1–8. ICOT, Tokyo, 1988.

[Gal93] J. Gallagher. Tutorial on Specialisation of Logic Programs. In *Proc. of Partial Evaluation and Semantics-Based Program Manipulation, Copenhagen, Denmark, June 1993*, pages 88–98. ACM, New York, 1993.

[GB90] J. Gallagher and M. Bruynooghe. Some Low-Level Source Transformations for Logic Programs. In M. Bruynooghe, editor, *Proc. of 2nd Workshop on Meta-Programming in Logic*, pages 229–246. Department of Computer Science, KU Leuven, Belgium, 1990.

[GHR92] J.C. González-Moreno, M.T. Hortalá-González, and M. Rodríguez-Artalejo. Denotational versus Declarative Semantics for Functional Programming. In *Proc. of CSL'91*, pages 134–148. Springer LNCS 626, 1992.

[GK93] R. Glück and A.V. Klimov. Occam's Razor in Metacomputation: the Notion of a Perfect Process Tree. In P. Cousot, M. Falaschi, G. Filè, and A. Rauzy, editors, *Proc. of 3rd Int'l Workshop on Static Analysis, WSA'93*, pages 112–123. Springer LNCS 724, 1993.

[GL91] M. Gabbrielli and G. Levi. On the Semantics of Logic Programs. In J. Leach-Albert, B. Monien, and M. Rodríguez-Artalejo, editors, *Proc.*

of the 18th Int'l Colloquium on Automata, Languages and Programming, pages 1–19. Springer LNCS 510, 1991.

[GM86] E. Giovannetti and C. Moiso. A completeness result for E-unification algorithms based on Conditional Narrowing. In M. Boscarol, L. Carlucci, and G. Levi, editors, *Foundations of Logic and Functional Programming*, pages 157–167. Springer LNCS 306, 1986.

[GR89] J.H. Gallier and S. Raatz. Extending SLD-resolution to equational Horn clauses using E-unification. *Journal of Logic Programming*, 6:3–43, 1989.

[GS94] R. Glück and M.H. Sørensen. Partial Deduction and Driving are Equivalent. In *Proc. Int'l Symp. on Programming Language Implementation and Logic Programming, PLILP'94*, pages 165–181. Springer LNCS 844, 1994.

[Han90] M. Hanus. Compiling Logic Programs with Equality. In *Proc. of 2nd Int'l Workshop on Programming Language Implementation and Logic Programming*, pages 387–401. Springer LNCS 456, 1990.

[Han91] M. Hanus. Efficient Implementation of Narrowing and Rewriting. In *Proc. Int'l Workshop on Processing Declarative Knowledge*, pages 344–365. Springer LNAI 567, 1991.

[Han92] M. Hanus. Improving Control of Logic Programs by Using Functional Logic Languages. In M. Bruynooghe and M. Wirsing, editors, *Proc. of PLILP'92, Leuven (Belgium)*, volume 631, pages 1–23. Springer LNCS 631, 1992.

[Han94a] M. Hanus. Combining Lazy Narrowing with Simplification. In *Proc. of 6th Int'l Symp. on Programming Language Implementation and Logic Programming*, pages 370–384. Springer LNCS 844, 1994.

[Han94b] M. Hanus. The Integration of Functions into Logic Programming: From Theory to Practice. *Journal of Logic Programming*, 19&20:583–628, 1994.

[Han94c] M. Hanus. Towards the Global Optimization of Functional Logic Programs. In *Proc. of Fifth Int'l Conf. on Compiler Construction*, pages 68–82. Springer LNCS 786, 1994.

[Har77] A. Haraldsson. *A Program Manipulation System Based on Partial Evaluation*. PhD thesis, Linköping University, Sweden, 1977. Linköping Studies in Science and Technology Dissertations 14.

[Har78] A. Haraldsson. A Partial Evaluator, and its Use for Compiling Iterative Statements in Lisp. In *Proc. of 5th ACM Symp. on Principles of Programming Languages, POPL'78*, pages 195–202. ACM, New York, 1978.

[HL96] M. Hanus and S. Lucas. A denotational semantics for needed narrowing. In P. Lucio, M. Martelli, and M. Navarro, editors, *Proc. of the 1996 Joint Conf. on Declarative Programming, APPIA-GULP-PRODE'96*, pages 259–270. U. del País Vasco, 1996.

[Höl89] S. Hölldobler. *Foundations of Equational Logic Programming.* Springer LNAI 353, 1989.

[HP91] P. Van Hentenryck and T. Le Provost. Incremental Search in Constraint Logic Programming. *New Generation Computing*, 9:257–275, 1991.

[HR89] M. Hermenegildo and F. Rossi. On the Correctness and Efficiency of Independent And-Parallelism in Logic Programs. In E. Lusk and R. Overbeck, editors, *Proc. of the 1989 North American Conf. on Logic Programming*, pages 369–389. The MIT Press, Cambridge, MA, 1989.

[HR95] M. Hermenegildo and F. Rossi. Strict and Non-Strict Independent And-Parallelism in Logic Programs: Correctness, Efficiency and Compile-Time Conditions. *Journal of Logic Programming*, 22:1–46, 1995.

[Hul80] J.M. Hullot. Canonical Forms and Unification. In *Proc of 5th Int'l Conf. on Automated Deduction*, pages 318–334. Springer LNCS 87, 1980.

[Hus85] H. Hussmann. Unification in Conditional-Equational Theories. In *Proc. of EUROCAL'85*, pages 543–553. Springer LNCS 204, 1985.

[HW92] W. Hans and S. Winkler. Abstract Interpretation of Functional Logic Languages. Technical Report 92-43, Technical University of Aachen (RWTH Aachen), 1992.

[HZ94] M. Hanus and F. Zartmann. Mode Analysis of Functional Logic Programs. In *Proc. of 1st Int'l Static Analysis Symposium, SAS'94*, pages 26–42. Springer LNCS 864, 1994.

[Jac89] J.-M. Jacquet. *Conclog: A Methodological Approach to Concurrent Logic Programming.* PhD thesis, University of Namur, Belgium, 1989.

[JGS93] N.D. Jones, C.K. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Program Generation.* Prentice-Hall, Englewood Cliffs, NJ, 1993.

[JL87] J. Jaffar and J.-L. Lassez. Constraint Logic Programming. In *Proc. of 14th Annual ACM Symp. on Principles of Programming Languages*, pages 111–119. ACM, 1987.

[JL89] D. Jacobs and A. Langen. Accurate and Efficient Approximation of Variable Aliasing in Logic Programs. In E. Lusk and R. Overbeck, editors, *Proc. of the 1989 North American Conf. on Logic Programming*, pages 154–165. The MIT Press, Cambridge, MA, 1989.

[JLM84] J. Jaffar, J.-L. Lassez, and M.J. Maher. A theory of complete logic programs with equality. *Journal of Logic Programming*, 3:211–223, 1984.

[JLM86] J. Jaffar, J.-L. Lassez, and M.J. Maher. PROLOG-II an instance of the logic programming language scheme. In M. Wirsing, editor, *Formal Descriptions of Programming Concepts*. North-Holland, 1986.

[JM87] J. Jaffar and S. Michaylov. Methodology and Implementation of a CLP System. In J.-L. Lassez, editor, *Proc. of Fourth Int'l Conf. on Logic Programming*. The MIT Press, Cambridge, MA, 1987.

[JMM93] J.A. Jiménez-Martin, J. Mariño-Carballo, and J.J. Moreno-Navarro. Efficient Compilation of Lazy Narrowing into Prolog. In *Proc. of LOPSTR'92*, pages 370–384. Springer LNAI, 1993.

[Jon88] N.D. Jones. Automatic Program Specialization: A Re-Examination from Basic Principles. In D. Bjørner, A.P. Ershov, and N.D. Jones, editors, *Proc. of the Int'l Workshop on Partial Evaluation and Mixed Computation*, pages 225–282. North-Holland, Amsterdam, 1988.

[Jon94] N.D. Jones. The Essence of Program Transformation by Partial Evaluation and Driving. In N.D. Jones, M. Hagiya, and M. Sato, editors, *Logic, Language and Computation*, pages 206–224. Springer LNCS 792, 1994.

[JS87] N.D. Jones and H. Søndergaard. A Semantics-based Framework for the Abstract Interpretation of Prolog. In S. Abramsky and C. Hankin, editors, *Abstract Interpretation of Declarative Languages*, pages 123–142. Ellis Horwood, Chichester, 1987.

[Kap87] S. Kaplan. Simplifying Conditional Term Rewriting Systems: Unification, Termination and Confluence. *Journal of Symbolic Computation*, 4:295–334, 1987.

[KH91] H. Kuchen and W. Hans. An AND-Parallel Implementation of the Functional Logic Language Babel. In *Aachener Informatik-Bericht*, volume 12, pages 119–139, RWTH Aachen, 1991.

[Kle52] S.C. Kleene. *Introduction to Metamathematics*. D. van Nostrand, Princeton, NJ, 1952.

[KLMR90a] H. Kuchen, R. Loogen, J.J. Moreno-Navarro, and M. Rodríguez-Artalejo. Graph-based Implementation of a Functional Logic Language. In *Proc. of ESOP'90*, pages 279–290. Springer LNCS 432, 1990.

[KLMR90b] H. Kuchen, R. Loogen, J.J. Moreno-Navarro, and M. Rodríguez-Artalejo. Lazy Narrowing in a Graph Machine. In *Proc. of the Int'l Conf. on Algebraic and Logic Programming*, pages 298–317. Springer LNCS 463, 1990.

[Klo92] J.W. Klop. Term Rewriting Systems. In S. Abramsky, D. Gabbay, and T. Maibaum, editors, *Handbook of Logic in Computer Science*, volume I, pages 1–112. Oxford University Press, 1992.

[KMH92] H. Kuchen, J.J. Moreno-Navarro, and M. Hermenegildo. Independent AND-Parallel Implementation of Narrowing. In M. Bruynooghe and M. Wirsing, editors, *Proc. of PLILP'92, Leuven, Belgium*, pages 24–38. Springer LNCS 631, 1992.

[KMP77] D.E. Knuth, J.H. Morris, and V.R. Pratt. Fast Pattern Matching in Strings. *SIAM Journal of Computation*, 6(2):323–350, 1977.

[Kom82] H.J. Komorowski. Partial Evaluation as a Means for Inferencing Data Structures in an Applicative Language: A Theory and Implementation in the Case of Prolog. In *Proc. of 9th ACM Symp. on Principles of Programming Languages*, pages 255–267, 1982.

[Kru60] J.B. Kruskal. Well-quasi-ordering, the Tree Theorem, and Vazsonyi's Conjecture. *Transactions of the American Mathematical Society*, 95:210–225, 1960.

[Lan75] D.S. Lankford. Canonical Inference. Technical Report ATP-32, University of Texas at Austin, 1975.

[Llo87] J.W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, Berlin, 1987. Second edition.

[LLR93] R. Loogen, F. López-Fraguas, and M. Rodríguez-Artalejo. A Demand Driven Computation Strategy for Lazy Narrowing. In J. Penjam and M. Bruynooghe, editors, *Proc. of PLILP'93, Tallinn (Estonia)*, pages 184–200. Springer LNCS 714, 1993.

[LMM88] J.-L. Lassez, M. J. Maher, and K. Marriott. Unification Revisited. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 587–625. Morgan Kaufmann, Los Altos, Ca., 1988.

[Lom67] L.A. Lombardi. Incremental Computation. In F.L. Alt and M. Rubinoff, editors, *Advances in Computers, vol. 8*, pages 247–333. Academic Press, New York, 1967.

[LR64] L.A. Lombardi and B. Raphael. Lisp as the Language for an Incremental Computer. In E.C. Berkeley and D.G. Bobrow, editors, *The Programming Language Lisp: Its Operation and Applications*, pages 204–219. The MIT Press, Cambridge, MA, 1964.

[LS75] G. Levi and F. Sirovich. Proving Program Properties, Symbolic Evaluation and Logical Procedural Semantics. In *Proc. of MFCS'75*, pages 294–301. Springer LNCS 32, 1975.

[LS87] J.W. Lloyd and J.C. Shepherdson. Partial Evaluation in Logic Programming. Technical Report CS-87-09, Computer Science Department, University of Bristol, 1987. Revised version 1989, in [LS91].

[LS91] J.W. Lloyd and J.C. Shepherdson. Partial Evaluation in Logic Programming. *Journal of Logic Programming*, 11:217–242, 1991.

[Mah88] M.J. Maher. Complete Axiomatizations of the Algebras of Finite, Rational and Infinite Trees. In *Proc. of Third IEEE Symp. on Logic In Computer Science*, pages 348–357. Computer Science Press, New York, 1988.

[Mah90] M.J. Maher. On Parameterized Substitutions. Technical Report RC 16042, IBM - T.J. Watson Research Center, Yorktown Heights, NY, 1990.

[Mar93] K. Marriott. Frameworks for Abstract Interpretation. *Acta Informatica*, 30:103–129, 1993.

[McC64] J. McCarthy. A Basis for a Mathematical Theory of Computation. In P. Brafford and D. Hirschberg, editors, *Computer Programming and Formal Systems*, pages 33–70. North-Holland, Amsterdam, 1964.

[MG94] B. Martens and J. Gallagher. Ensuring Global Termination of Partial Deduction while Allowing Flexible Polyvariance. Technical Report CSTR-94-16, Computer Science Department, University of Bristol, December 1994.

[MG95] B. Martens and J. Gallagher. Ensuring Global Termination of Partial Deduction while Allowing Flexible Polyvariance. In K. Furukawa and K. Ueda, editors, *Proc. of ICLP'95*, pages 597–611, 1995.

[MH89] K. Muthukumar and M. Hermenegildo. Determination of Variable Dependence Information at Compile-Time Through Abstract Interpretation. In E. Lusk and R. Overbeck, editors, *Proc. of the 1989 North American Conf. on Logic Programming*, pages 166–189. The MIT Press, Cambridge, MA, 1989.

[MH94] A. Middeldorp and E. Hamoen. Completeness Results for Basic Narrowing. *Applicable Algebra in Engineering, Communication and Computing*, 5:213–253, 1994.

[MJ86] A. Mycroft and N.D. Jones. A Relational Framework for Abstract Interpretation. In *Programs as Data Objects*, pages 536–547. Springer LNCS 154, 1986.

[MKLR90] J.J. Moreno-Navarro, H. Kuchen, R. Loogen, and M. Rodríguez-Artalejo. Lazy Narrowing in a Graph Machine. In *Proc. of the 2nd Int'l Conf. on Algebraic and Logic Programming, ALP'90*, pages 298–317. Springer LNCS 463, 1990.

[MKM+93] J.J. Moreno-Navarro, H. Kuchen, J. Mariño-Carballo, S. Winkler, and W. Hans. Efficient Lazy Narrowing using Demandedness Analysis. In J. Penjam and M. Bruynooghe, editors, *Proc. of PLILP'93, Tallinn (Estonia)*, pages 167–183. Springer LNCS 714, 1993.

[Mor89] J.J. Moreno-Navarro. *Babel: diseño, semántica e implementación de un lenguaje que integra la programación funcional y lógica.* PhD thesis, Facultad de Informática, Universidad Complutense de Madrid, 1989.

[MR92] J.J. Moreno-Navarro and M. Rodríguez-Artalejo. Logic Programming with Functions and Predicates: the Language BABEL. *Journal of Logic Programming*, 12(3):191–223, 1992.

[MS89] K. Marriott and H. Søndergaard. Semantics-based Dataflow Analysis of Logic Programs. In G. Ritter, editor, *Information Processing 89*, pages 601–606. North-Holland, 1989.

[MS92] K. Marriott and H. Søndergaard. Bottom-up Dataflow Analysis of Normal Logic Programs. *Journal of Logic Programming*, 13:181–204, 1992.

[Nie88] F. Nielson. Strictness Analysis and Denotational Abstract Interpretation. *Information and Computation*, 76:29–92, 1988.

[NRS89] W. Nutt, P. Réty, and G. Smolka. Basic narrowing revisited. *Journal of Symbolic Computation*, 7:295–317, 1989.

[OMI95] S. Okui, A. Middeldorp, and T. Ida. Lazy Narrowing: Strong Completeness and Eager Variable Elimination. In *Proc. of the 20th Colloquium on Trees in Algebra and Programming*, pages 394–408. Springer LNCS 915, 1995.

[Pad88] P. Padawitz. *Computing in Horn Clause Theories*, volume 16 of *EATCS Monographs on Theoretical Computer Science.* Springer-Verlag, Berlin, 1988.

[Pal90] C. Palamidessi. Algebraic Properties of Idempotent Substitutions. In M.S. Paterson, editor, *Proc. of 17th Int'l Colloquium on Automata, Languages and Programming*, pages 386–399. Springer LNCS 443, 1990.

[Plo81] G. Plotkin. A structured approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.

[Red85] U.S. Reddy. Narrowing as the Operational Semantics of Functional Languages. In *Proc. of Second IEEE Int'l Symp. on Logic Programming*, pages 138–151. IEEE, New York, 1985.

[Rét87] P. Réty. Improving basic narrowing techniques. In *Proc. of the Conf. on Rewriting Techniques and Applications*, pages 228–241. Springer LNCS 256, 1987.

[RF93] M.J. Ramírez and M. Falaschi. Conditional Narrowing with Constructive Negation. In E. Lamma and P. Mello, editors, *Proc. of Third Int'l Workshop on Extensions of Logic Programing ELP'92*, pages 59–79. Springer LNCS 660, 1993.

[RKKL85] P. Réty, C. Kirchner, H. Kirchner, and P. Lescanne. NARROWER: A new algorithm for unification and its applications to logic programming. In *Proc. of RTA'85*, pages 141–157. Springer LNCS 202, 1985.

[Rom91] A. Romanenko. Inversion and metacomputation. In *Partial Evaluation and Semantics-Based Program Manipulation*, pages 12–22. Sigplan Notices, 26(9), ACM, New York, September 1991.

[San95a] D. Sands. Proving the Correctness of Recursion-Based Automatic Program Transformations. In P. Mosses, M. Nielsen, and M. Schwartzbach, editors, *Proc. of 6th Int'l Joint Conf. on Theory and Practice of Software Development, TAPSOFT'95*, pages 681–695. Springer LNCS 915, 1995.

[San95b] D. Sands. Total Correctness by Local Improvement in Program Transformation. In *Proc. of the 22nd Annual ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages, POPL'95*, pages 221–232. ACM, New York, 1995.

[Sar91] V.A. Saraswat. Semantic Foundation of Concurrent Constraint Programming. In *Proc. of 18th ACM Symp. on Principles of Programming Languages*. ACM, New York, 1991.

[Sco82] D. Scott. Domains for Denotational Semantics. In *Proc. of ICALP'82*, pages 577–613. Springer LNCS 140, 1982.

[SG95] M.H. Sørensen and R. Glück. Generalization in Positive Supercompilation. In J.W. Lloyd, editor, *Proc. of ILPS'95*, pages 465–479. The MIT Press, Cambridge, MA, 1995.

[SGJ94] M.H. Sørensen, R. Glück, and N.D. Jones. Towards Unifying Partial Evaluation, Deforestation, Supercompilation, and GPC. In D. Sannella, editor, *Proc. of the 5th European Symp. on Programming, ESOP'94*, pages 485–500. Springer LNCS 788, 1994.

[Sie89] J.H. Siekmann. Unification Theory. *Journal of Symbolic Computation*, 7:207–274, 1989.

[Sla74] J.R. Slagle. Automated Theorem-Proving for Theories with Simplifiers, Commutativity and Associativity. *Journal of the ACM*, 21(4):622–642, 1974.

[Sør94] M.H. Sørensen. Turchin's Supercompiler Revisited: An Operational Theory of Positive Information Propagation. Technical Report 94/7, Master's Thesis, DIKU, University of Copenhagen, Denmark, 1994.

[SR92] F. Sáenz and J.J. Ruz. Parallelism Identification in BABEL Funtional Logic Programs. In *Proc. of 7th Italian Conf. on Logic Programming GULP'92*, pages 409–423, 1992.

[SR93] F. Sáenz and J.J. Ruz. Análisis de Independencia de Programas Babel. In *Proc. of ProDe'93*, pages 21–38, 1993.

[SS88] P. Sestoft and H. Søndergaard. A bibliography on partial evaluation. *Sigplan Notices*, 23(2):19–27, Feb 1988.

[Sto77] J.E. Stoy. *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory.* The MIT Press, Cambridge, MA, 1977.

[TS84] H. Tamaki and T. Sato. Unfold/Fold Transformations of Logic Programs. In S. Tärnlund, editor, *Proc. of Second Int'l Conf. on Logic Programming*, pages 127–139, 1984.

[Tur74] V.F. Turchin. Equivalent Transformations of Refal Programs. In *Automatizirovannaya Sistema Upravlenya Stroitelstvom, vol. VI*, pages 36–68. TsNIPIASS, Moscow, 1974. (In Russian).

[Tur77] V.F. Turchin, editor. *Basic Refal and its Implementation on Computers.* GOSSTROI SSSR, TsNIPIASS, Moscow, 1977. (In Russian).

[Tur79] V.F. Turchin. A Supercompiler System Based on the Language Refal. *SIGPLAN Notices*, 14(2):46–54, February 1979.

[Tur80a] V.F. Turchin. Semantic definitions in Refal and automatic production of compilers. In N.D. Jones, editor, *Semantics-Directed Compiler Generation*, pages 441–474. Springer LNCS 94, 1980.

[Tur80b] V.F. Turchin. The Use of Metasystem Transition in Theorem Proving and Program Optimization. In J. De Bakker and J. van Leeuven, editors, *Proc of 7th Int'l Conf. on Automata, Languages and Programming, Noordwijkerhout, The Netherlands*, pages 645–657. Springer LNCS 85, 1980.

[Tur86] V.F. Turchin. The Concept of a Supercompiler. *ACM Transactions on Programming Languages and Systems*, 8(3):292–325, July 1986.

[Tur88] V.F. Turchin. The Algorithm of Generalization in the Supercompiler. In D. Bjørner, A.P. Ershov, and N.D. Jones, editors, *Proc. of the Int'l Workshop on Partial Evaluation and Mixed Computation*, pages 531–549. North-Holland, Amsterdam, 1988.

[Tur89] V.F. Turchin. *Refal-5, Programming Guide & Reference Manual.* New England Publishing Co., Holyoke, MA, 1989.

[vEY87] M.H. van Emden and K. Yukawa. Logic Programming with Equations. *Journal of Logic Programming*, 4:265–288, 1987.

[Wad88] P.L. Wadler. Deforestation: transforming programs to eliminate trees. In *Proc of the European Symp. on Programming, ESOP'88*, pages 344–358. Springer LNCS 300, 1988.

[WS90] B. Wang and R.K. Shyamasundar. Methodology for Proving the Termination of Logic Programs. In *Proc. of PLILP'90*, pages 204–221. Springer LNCS 456, 1990.

[Yan87] R. Yang. P-Prolog: A Parallel Logic Language. *World Scientific*, 1987.

[You89] Y.H. You. Enumerating Outer Narrowing Derivations for Constructor-Based Term Rewriting Systems. *Journal of Symbolic Computation*, 7:319–343, 1989.

[Zha94] K. Zhang. A Review of Exploitation of AND-Parallelism and Combined AND/OR-Parallelism in Logic Programs. *ACM Sigplan Notices*, 29(2):25–32, 1994.

Índice de Materias