Técnicas de Evaluación Parcial Perezosa en Programas Uniformes

M. Alpuente[1], M. Falaschi[2], P. Julián[3], and G. Vidal[1]

[1] DSIC, U. Politécnica de Valencia, Camino de Vera s/n, 46022 Valencia, Spain.
`{alpuente,gvidal}@dsic.upv.es`
[2] Dip. di Mat. e Informatica, U. Udine, Via delle Scienze 206, 33100 Udine, Italy.
`falaschi@dimi.uniud.it`
[3] Dep. de Informática, U. de Castilla-La Mancha, Ronda de Calatrava s/n, 13071 Ciudad Real, Spain.
`pjulian@inf-cr.uclm.es`

Abstract. La semántica operacional de los lenguajes lógico-funcionales se basa habitualmente en el empleo del *narrowing*, un mecanismo que aúna el principio de reducción de los lenguajes funcionales y el principio de resolución de los lenguajes lógicos. La posibilidad de realizar evaluación perezosa es una característica apreciada en los lenguajes (lógico-)funcionales. Recientemente se ha introducido una nueva estrategia de evaluación perezosa, denominada *narrowing* necesario, que resulta óptima en el sentido de que sólo efectúa cómputos realmente necesarios para la obtención de un resultado. En este informe realizamos un estudio formal y exhaustivo de la relación entre la estrategia de *narrowing* necesario y la estrategia de *narrowing* perezoso. Establecemos, principalmente, los siguientes resultados:

- La equivalencia existente entre las estrategias de *narrowing* necesario y de *narrowing* perezoso sobre una clase particular de programas que definimos a partir de la idea de los programas uniformes de [29, 30], a nuestro entender, la clase de programas más amplia para la cual ambas estrategias computan los mismos resultados y respuestas.
- El refinamiento de la estrategia de *narrowing* perezoso que llamamos *narrowing* perezoso uniforme, que resulta ser correcta y completa sobre la clase de los programas uniformes.

Estos resultados tienen aplicación inmediata en la optimización de las técnicas de evaluación parcial para programas lógico-funcionales con semántica no estricta. Apoyándonos en ellos, introducimos una nueva instancia del marco genérico de evaluación parcial que propusimos en [4] usando *narrowing* perezoso uniforme y para la cual demostramos su corrección y completitud fuerte.

Keywords: integración de programación lógica y funcional, sistemas de reescritura de términos, *narrowing*, estrategias de evaluación perezosa, *narrowing* perezoso, *narrowing* necesario.

1 Introducción.

Los lenguajes de programación lógico-funcional permiten integrar algunas de las mejores características de los paradigmas de programación lógica y funcional.

En este contexto se combinan de forma natural características que provienen del paradigma de programación funcional, como el empleo de tipos, funciones de orden superior, evaluación perezosa y cómputo con estructuras infinitas, con otras procedentes de la programación lógica, como el uso de la unificación, variables lógicas e inversión, manejo de información parcialmente definida y mecanismos predefinidos de búsqueda indeterminista ("built-in search"). En la última decada se han realizado importantes avances en el campo de la integración de lenguajes declarativos, una panorámica de los cuales puede encontrarse en [18, 21, 38].

Es un hecho comúnmente aceptado que los lenguajes lógico-funcionales con una semántica operacional completa se basan en alguna forma de *narrowing*. *Narrowing* es un mecanismo operacional semejante a la reescritura, que va precedido de una instanciación de la expresión a reducir (generalmente producida por un proceso de unificación), con el fin de incluir algunas de las características de la programación lógica, antes mencionadas, en un contexto funcional. Para evitar computaciones innecesarias y posibilitar el empleo de estructuras de datos infinitas, muchos trabajos se han centrado en el estudio de las estrategias de *evaluación perezosa* [12, 16, 19, 33, 40]. De entre las estrategias de evaluación perezosa, la estrategia de *narrowing* necesario [12] se ha postulado como óptima desde diferentes puntos de vista: es correcta y completa, con respecto a ecuaciones estrictas y soluciones constructoras, para la clase de programas inductivamente secuenciales; computa derivaciones necesarias de longitud mínima (en implementaciones basadas en grafos) y no obtiene soluciones redundantes. El *Narrowing* necesario guía sus cómputos mediante el empleo de unas estructuras, denominadas árboles definicionales [8], que contienen toda la información sobre las reglas del programa. Estas estructuras permiten seleccionar una posición del término que se está evaluando y que señala la aparición de un subtérmino que es inevitable reducir para la obtención de un resultado. Sin embargo, como se muestra en [1], la representación explícita y el uso de los árboles definicionales en la implementación del narrowing necesario puede tener un alto coste, tanto en aspectos de eficiencia (en tiempo de ejecución) como en el consumo de memoria, cuando se realizan cómputos con la estrategia de *narrowing* necesario. La introducción de *técnicas incrementales* puede reducir en más de un 50% (como se demuestra en la mayor parte de las pruebas efectuadas) la cantidad de memoria utilizada en la representación de los árboles definicionales; también hemos constatado que se experimenta un aumento de eficiencia en tiempo de ejecución, que si bien no es significativo en todos los casos, no provoca una sobrecarga en ninguno de ellos.

El propósito de la *evaluación parcial* (PE - del inglés *Partial Evaluation*) es la especialización de un programa con respecto a (parte de) sus datos de entrada. El resultado es un nuevo programa, más eficiente que el original cuando se ejecuta para el tipo de entradas para el que se realizó la especialización. La PE ha sido aplicada extensivamente tanto a los lenguajes declarativos como imperativos (ver [25] para una panorámica general sobre el área). La PE de programas lógico-funcionales fue introducida por vez primera en [3] y una visión exhaustiva de sus técnicas puede encontrarse en [4], donde se introduce un marco génerico de PE dirigido por *narrowing*.

Las características de la PE dirigida por *narrowing* dependen de la estrategia de *narrowing* empleada para el desplegado de los árboles locales. La PE usando *narrowing* perezoso presenta varios inconvenientes: i) se obtienen reglas redundantes en el programa especializado; ii) se pierde la ortogonalidad del programa original. Esto impide que la estrategia de *narrowing* perezoso pueda emplearse para ejecutar el programa especializado. Asimismo puede introducirse un empeoramiento, con respecto a la terminación de los cómputos, en comparación con el comportamiento del programa original. La especialización de programas usando *narrowing* perezoso también puede destruir las ventajas de la existencia de cómputos deterministas en el programa original. Como se muestra en [7], estos inconvenientes no aparecen cuando el proceso de PE se basa en la estrategia de *narrowing* necesario.

El presente informe investiga la relación precisa existente entre la estrategia de *narrowing* necesario y la estrategia de *narrowing* perezoso, que no requiere de las costosas estructuras de los árboles definicionales, e intenta identificar la clase de programas más amplia para la cual ambas estrategias tienen el mismo comportamiento operacional, evitándose así los problemas encontrados al usar la estrategia de *narrowing* perezoso en el proceso de PE. Por consiguiente, el objetivo de este trabajo se orienta a conseguir, empleando una estrategia de *narrowing* perezoso, los beneficios de la estrategia de *narrowing* necesario sin el coste de mantener y usar los árboles definicionales. Como resultado de nuestra investigación podemos concluir que dicha clase de programas se concreta en los programas uniformes (para los que damos una definición precisa basada en [28, 30]. A nuestro entender, los programas uniformes constituyen la clase más amplia para la cual ambas estrategias computan no sólo los mismos resultados sino también idénticas respuestas.

El informe se ha organizado en los siguientes apartados. La Sección 2 contiene una serie de notaciones y de conceptos fundamentales acerca de los sistemas de reescritura de términos, que después son empleados a lo largo de todo el trabajo. En la Sección 3 introducimos la programación lógico funcional, centrándonos en la sintaxis de los programas y los mecanismos operacionales del *narrowing* perezoso y del *narrowing* necesario, que se describen con detalle y para los que se enuncian sus propiedades principales. La sección 4 presenta los programas uniformes; en ella estudiamos sus características y, en particular, demostramos que cumplen la propiedad de ser inductivamente secuenciales. En la sección 5 presentamos los principales resultados teóricos del informe; más concretamente, introducimos un refinamiento de la estrategia de *narrowing* perezoso para programas uniformes, que denominamos *narrowing* perezoso uniforme, y probamos la equivalencia de la semántica operacional de respuestas computadas cuando utilizamos *narrowing* perezoso uniforme y *narrowing* necesario sobre programas uniformes. En la sección 6 se define una nueva instancia del marco genérico de evaluación parcial [4] usando *narrowing* perezoso uniforme y demostramos su corrección y completitud fuerte. El informe concluye con un resumen de los resultados obtenidos y las líneas de trabajo futuro.

2 Preliminares

En este apartado resumiremos, brevemente, algunas notaciones y resultados muy conocidos sobre sistemas de reescritura de términos [14, 26]. Aunque las definiciones que siguen a continuación se dan para una signatura homogenea, la extensión de las mismas al caso de signaturas heterogeneas ("*many-sorted*") es inmediata [41].

En este informe, $\mathcal{X}$ denota un conjunto infinito enumerable de *variables* y $\mathcal{F}$ denota el conjunto de *símbolos de función* (que denominamos la *signature*), cada uno de los cuales tiene una aridad previamente fijada. Suponemos que la signatura $\mathcal{F}$ está particionada en dos conjuntos $\mathcal{F} = \mathcal{C} \cup \mathcal{D}$, siendo $\mathcal{C}$ y $\mathcal{D}$ conjuntos disjuntos. Los símbolos de $\mathcal{C}$ se denominan *constructores* y los de $\mathcal{D}$ se denominan *funciones definidas*. $\mathcal{T}(\mathcal{F}, \mathcal{X})$ denota el conjunto de *términos* o *expresiones* construidas a partir de $\mathcal{F}$ y $\mathcal{X}$. $\mathcal{T}(\mathcal{F})$ denota el conjunto de *términos básicos* ("*ground*"), mientras que $\mathcal{T}(\mathcal{C}, \mathcal{X})$ denota el conjunto de *términos constructores*. Si $t \notin \mathcal{X}$, entonces $\mathcal{H}ead(t)$ es el símbolo de función que encabeza el término t, tambien denominado el *símbolo raíz* de t. Un *patrón* es un término de la forma $f(d_1, \ldots, d_n)$ donde $f/n \in \mathcal{D}$ y los argumentos $d_1, \ldots, d_n$ son términos constructores. Un término se dice *lineal* cuando no contiene multiples apariciones de la misma variable. La identidad de objetos sintácticos se denota mediante el operador relacional "$\equiv$". $\mathcal{V}ar(s)$ es el conjunto de variables que ocurren en el objeto sintáctico s.

Una *substitución* es una aplicación de $\mathcal{X}$ a $\mathcal{T}(\mathcal{F}, \mathcal{X})$ tal que su *dominio* $\mathcal{D}om(\sigma) = \{x \in \mathcal{X} \mid \sigma(x) \not\equiv x\}$ es finito. Como es común, representaremos frecuentemente una substitución σ mediante el conjunto $\{x/\sigma(x) \mid x \in \mathcal{D}om(\sigma)\}$. Denotamos la substitución identidad mediante el símbolo "*id*". Decimos que σ es una *substitución básica* si $\sigma(x)$ es un término básico para todo $x \in \mathcal{D}om(\sigma)$. Emplearemos la notación $\leq$ para designar el preorden de *máxima generalidad*, definido sobre el conjunto de las substituciones. Decimos que θ es *más general* que σ (en símbolos, $\theta \leq \sigma$) si y sólo si $(\exists\gamma)\ \sigma = \gamma \circ \theta$. La *restricción*, $\sigma_{\upharpoonright(V)}$, de una substitución σ a un conjunto V de variables se define como $\sigma_{\upharpoonright(V)}(x) = \sigma(x)$ si $x \in V$ and $\sigma_{\upharpoonright(V)}(x) = x$ si $x \notin V$. Un *renombramiento* es una substitución ρ para la cual existe la substitución inversa ρ^{-1} tal que $\rho \circ \rho^{-1} = \rho^{-1} \circ \rho = id$.

Un término t es *más general* que uno s (o también, s es una *instancia* de t), en símbolos $t \leq s$, si $(\exists\sigma)\ s = \sigma(t)$. Dos términos t y t' son *variantes* el uno del otro si existe un renombremiento ρ tal que $t' = \rho(t)$. Un *unificador* de un par de términos $\langle t_1, t_2 \rangle$ es una substitución σ tal que $\sigma(t_1) = \sigma(t_2)$. Un unificador σ se denomina *unificador más general* (*mgu* - del inglés *most general unifier*) si para cualquier otro unificador σ' se cumple que $\sigma \leq \sigma'$.

Los términos se representan como árboles etiquetados a la manera usual, siendo las las etiquetas de los nodos los símbolos que forman el término. Las *posiciones* (también denominadas *ocurrencias*) de un término t se representan por secuencias, posiblemente vacías, de números naturales que se emplean como un camino de acceso a un subtérmino de t. Las posiciones están ordenadas por el orden prefijo "$\leq$": de manera que $p \leq q$, si existe un w tal que $p.w = q$, donde $p.w$ denota la concatenación de secuencias p and w. Cuando dos posiciones no esten relacionadas diremos que son *disjuntas* y lo indicaremos mediante el símbolo "$\not\leq$". Denotamos la secuencia vacia mediante el símbolo "Λ". $\mathcal{P}os(t)$ y $\mathcal{FP}os(t)$ denotan, respectivamente, el conjunto de posiciones y el conjunto de posiciones no variables del término t. $t[p]$ denota la etiqueta asociada al árbol del término t en la posicion $p \in \mathcal{P}os(t)$. $t|_p$ es el subtérmino de t en la posición p. $t[s]_p$ es el término resultante de reemplazar, en la posición p del term t, el subtérmino $t|_p$ por s.

Por simplicidad, limitaremos nuestra discusión a sistemas de reescritura de términos incondicionales. Una *regla de reescritura* es un par de términos $l \to r$, tal que $l \notin \mathcal{X}$, y $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$. l y r se denominan la *parte izquierda* (lhs - del inglés, *left-hand side*) y la *parte derecha* (rhs - del inglés, *right-hand side*), respectivamente, de la regla de reescritura. Un *sistema de reescritura* (TRS, del inglés, *Term Rewrite System*), $\mathcal{R}$, es un conjunto finito de reglas de reescritura.

Las reglas de reescritura de un TRS definen una relación de reescritura $\to$ entre los términos, que se define como sigue: $t \to_{p,l\to r} s$ si existe una posición $p \in \mathcal{P}os(t)$, una regla de reescritura $l \to r$, y una substitución σ con $t|_p = \sigma(l)$ y $s = t[\sigma(r)]_p$. Decimos que $t|_p$ es un *redex* (del inglés - *reducible expression*) de t y que $\sigma(r)$ es el *contracto* de $\sigma(l)$. También decimos que t se reduce a s en un *paso de reescritura* $t \to_{p,l\to r} s$. Una secuencia de cero o más pasos de reescritura se representa por $t \to^* s$. Un término t es *reducible* a un término s si $t \to^* s$; en ocasiones también diremos que s es *derivable* apartir de t. Un término t es *irreducible* o está en *forma normal* si no posee redexes. Un término s posee una forma normal si existe una reducción $s \to^* t$, donde t es una forma normal.

Un TRS es *terminante* o *noetheriano* si no hay secuencias de reescritura, $t_1 \to t_2 \to \ldots$, infinitas. Dado que en este trabajo no imponemos el requisito de que las reglas sean terminantes, no tiene porque existir una forma normal. Un TRS se dice *confluente* si, cuando un término s se reduce a dos términos t_1 y t_2, ambos t_1 y t_2 se reducen al mismo término. Dadas dos reglas $l_1 \to r_1$ y $l_2 \to r_2$ para las que existe una posición $p \in \mathcal{FP}os(l_1)$ tal que $l_1|_p$ y l_2 unifican con *mgu* σ, entonces el par de reductos $\langle \sigma(r_1), l_1[\sigma(r_2)]_p \rangle$ es un *par crítico*.

Un TRS se dice *ortogonal* si cumple las siguientes restricciones sintácticas:

1. *Linealidad por la izquierda*: para toda regla $l \to r$ del TRS, la lhs l es un término lineal.
2. *No ambigüedad (fuerte)*: El TRS no contiene pares cíticos; i.e., las lhs de las reglas del TRS no *solapan*.

Un TRS se dice *débilmente ortogonal* si es lineal por la izquierda y sólamente contiene pares críticos triviales; i,e, si $\langle s, t \rangle$ es un par crítico entonces $s \equiv t$. Es bien conocido que los TRS (débilmente) ortogonales son confluentes ([26]) [22]. Una consecuencia del resultado anterior es que en los TRS ortogonales la forma normal de un término, cuando existe, es única. Una propiedad de los TRS ortogonales es que los descendientes[1] de un redex continuan siendo redexes. Una *posición necesaria* ("needed") [23, 27] es aquella que señala la aparición de un redex tal que él o sus desdendientes deben ser reducidos en cualquier reducción de un término a su forma normal.

Un TRS se dice basado en constructores (CB, del inglés, *Constructor Based*) si los argumentos de las lhs's son términos constructores o variables.

Por ultimo diremos que, una variable es *fresca* si es una variable nueva que no ha sido empleada con anterioridad ni en el programa ni en un paso previo de una derivación.

3 Programación Lógico-Funcional.

Los lenguajes lógico-funcionales pueden considerarse como extensiones de los lenguajes funcionales con principios derivados de la programación lógica [42]. La mayoría de estos lenguajes integrados utilizan sistemas de reescritura como programas y alguna variante del *narrowing* como principio operacional. *Narrowing* es una generalización del mecanismo operacional de la *reescritura*, empleada en los lenguajes funcionales, con el fin de extender éstos con características de la programación lógica: variables lógicas, estructuras de datos parcialmente definidas y mecanismos de búsqueda deductiva no determinista. *Narrowing* extiende el mecanismo de la reescritura sustituyendo ajuste de patrones por unificación, de forma que ambos coinciden cuando se emplean sobre términos que no contienen variables. *Narrowing* proporciona completitud en el sentido de la programación lógica – computación de respuestas – así como también en el de la programación funcional – computación de formas normales –.

En los próximos apartados profundizaremos en los aspectos sintácticos de nuestro lenguaje y en los princios operacionales que lo rigen.

3.1 Programas.

La sintaxis del lenguaje lógico funcional que utilizamos en la elaboración de este trabajo, esencialmente, es equivalente a la de (un subconjunto de) Babel [33, 40], Toy [13] o Curry [19], que recientemente se ha propuesto como un estándar en el área de la integración de los lenguajes declarativos.

Fijamos las características sintácticas del lenguaje diciendo que los programas son TRS's basados en constructores y ortogonales. Los programas con los que tratamos pueden ser no-terminantes. Para esta clase de programas, un término t es una *forma normal en cabeza* (*constructora*) (hnf - del ingles "*head normal form*") si t es una variable o $\mathcal{H}ead(t) \in \mathcal{C}$.

Uno de los intereres primarios de los lenguajes lógico funcionales es la resolución de ecuaciones. Una *ecuación* es un par de términos (s, t). Con el fin de aumentar la expresividad del lenguaje extendemos la signatura $\mathcal{F}$ con un conjunto $\mathcal{P}$ de símbolos de función primitivos. El conjunto $\mathcal{P}$ se de fine como

[1] ver [23] para una definición formal de descendiente

$\mathcal{P} = \{\approx, \wedge, \Rightarrow\}$, lo que permite manipular expresiones complejas conteniendo ecuaciones, que ahora podremos expresar como un término $s \approx t$, conjunciones $b_1 \wedge b_2$, y expresiones condicionales (guardadas) $b \Rightarrow t$, que también expresamos como términos. Así pues, la signatura extendida es $\mathcal{F} = \mathcal{C} \cup \mathcal{D} \cup \mathcal{P}$. Usualmente tratamos los símbolos de $\mathcal{P}$ como operadores infijos. La semántica de estos símbolos primitivos viene determinada por el siguiente conjunto de reglas predefinidas que denotamos por StrEq y que suponemos pertenecen a todo programa:

$$
\begin{array}{rll}
c \approx c & \to \mathit{true} & \%\ c/0 \in \mathcal{C} \\
c(x_1, \ldots, x_n) \approx c(y_1, \ldots, y_n) & \to (x_1 \approx y_1) \wedge \ldots \wedge (x_n \approx y_n) & \%\ c/n \in \mathcal{C} \\
\mathit{true} \wedge x & \to x & \\
(\mathit{true} \Rightarrow x) & \to x &
\end{array}
$$

La extensión de un programa $\mathcal{R}$ con el conjunto de reglas StrEq se denota por $\mathcal{R}_+$, i.e., $\mathcal{R}_+ = (\mathcal{R} \cup$ StrEq). Estas reglas son ortogonales y definen la validez de una ecuación como la *igualdad estricta* entre términos, lo cual es común en los lenguajes funcionales cuando los cómputos pueden no terminar [16, 33, 40]. La igualdad estricta no posee la propiedad reflexiva, i.e., no se cumple que $t \approx t$ para todo término t. La igualdad estricta trata dos términos s y t como idénticos si y solamente si tienen como forma normal el mismo término constructor básico, i.e., si $s \approx t$ se reduce a *true*. La equivalencia entre la reducibilidad al mismo término constructor básico y la reducibilidad a *true*, queda establecida en la siguiente proposición.

Proposition 1. *[11] Sea $\mathcal{R}$ un programa. Un par de términos s y t son reducibles por reescritura al mismo término constructor básico, utilizando las reglas de $\mathcal{R}$ si y solamente si $s \approx t$ se reescribe a true usando las reglas del programa $\mathcal{R}_+$.*

Notad que, aunque el modelo computacional básico solamente emplea reglas incondicionales, todavia es adecuado para soportar programas lógicos, ya que las reglas de reescritura condicionales $l \to r \Leftarrow C$ pueden simularse mediante reglas incondicionales con expresiones guardadas $l \to (C \Rightarrow r)$ utilizando el símbolo de función primitivo '$\Rightarrow$' al igual que en el lenguaje Babel [40].

Por razones de simplicidad, suponemos que el operador '$\wedge$' es asociativo por la derecha y que '$\approx$' tiene más precedencia (i.e., liga más) que '$\wedge$' y '$\wedge$' que '$\Rightarrow$'. Así, por ejemplo, el término $b_1 \wedge (b_2 \wedge b_3)$ puede escribirse como $b_1 \wedge b_2 \wedge b_3$ y el término $((t_1 \approx s_1) \wedge (t_2 \approx s_2)) \Rightarrow t$ se escribe como $t_1 \approx s_1 \wedge t_2 \approx s_2 \Rightarrow t$.

También en lo sucesivo, omitiremos el subindice "+" que denota a los programas extendidos con el conjunto de reglas StrEq. Así pues, salvo que se diga lo contrario $\mathcal{R}$ es indicativo de $\mathcal{R}_+$.

3.2 Narrowing y Estrategias de Narrowing.

Esencialmente, el mecanismo de *narrowing* calcula una sustitución apropiada, σ, que cuando se aplica sobre el término en consideración, t, éste puede ser reducido en un paso de reescritura [18].

Example 1. Sean las reglas que definen el predicado menor o igual "$\leqslant$" sobre los números naturales representados por los términos formados usando los símbolos constructores "0" y sucesor "s":

$$\begin{array}{rcll} 0 & \leqslant & N & \rightarrow \; true \\ s(M) & \leqslant & 0 & \rightarrow \; false \\ s(M) & \leqslant & s(N) & \rightarrow \; M \leqslant N \end{array}$$

El término $s(X) \leqslant Y$ puede reducirse a *true* instanciando Y a $s(Y1)$, para aplicar la tercera regla, seguida de la instanciación de X por 0, para aplicar la primera regla:

$$s(X) \leqslant Y \overset{\{Y/s(Y1)\}}{\rightsquigarrow} X \leqslant Y1 \overset{\{X/0\}}{\rightsquigarrow} true$$

Es habitual definir un paso de *narrowing* de la forma siguiente, que se corresponde con la definición de paso de reescritura, pero sustituyendo el mecanismo de *ajuste de patrones* ("*matching*") por el de *unificación*:

Definition 1 (Paso de Narrowing).
Sea $\mathcal{R}$ un programa. Sean t y s términos. Decimos que t se reduce por narrowing *a s si existe una posicion $p \in \mathcal{FP}os(t)$, una (variante renombrada aparte de una) regla de reescritura $R \equiv l \rightarrow r$ de $\mathcal{R}$ y una substitución σ tal que:*

– $\sigma = mgu(\{t_{|p} = l\})$ *(i.e., σ es el unificador más general de $t_{|p}$ y l), y*
– $s \equiv \sigma(t[r]_p)$.

Escribimos que $t \overset{[p,R,\sigma]}{\rightsquigarrow} s$ o, simplemente, $t \overset{\sigma}{\rightsquigarrow} s$.

Algunos autores [11, 41] no exigen que las substiciones computadas, σ, sean unificadores más generales, basta con que sean unificadores. Con este enfoque, $t \overset{[p,R,\sigma]}{\rightsquigarrow} s$ es un *paso de narrowing* si p es una posición no variable de t y $\sigma(t) \rightarrow_{p,R} s$. Esta es la definición más amplia que puede darse de paso de *narrowing*. La Definición 1 tiene la ventaja de que los unificadores más generales pueden computarse de forma única, mientras que existen muchos unificadores independientes. Sin embargo, eliminar la restricción de que la substitución σ sea un unificador más general es un requisito indispensable en la definición de algunas estrategias de *narrowing* (ver más adelante).

Definition 2 (Derivación de Narrowing).
Sea $\mathcal{R}$ un programa y t un término. Decimos que existe una derivación de narrowing *de t a s, si existe una secuencia de pasos $t \equiv t_0 \overset{[p_1,R_1,\sigma_1]}{\rightsquigarrow} t_1 \overset{[p_2,R_2,\sigma_2]}{\rightsquigarrow} \ldots \overset{[p_n,R_n,\sigma_n]}{\rightsquigarrow} t_n \equiv s$. Escribimos que $t \overset{\sigma}{\rightsquigarrow}{}^{*} s$, donde $\sigma = \sigma_n \circ \ldots \circ \sigma_2 \circ \sigma_1$. Decimos que s es el* resultado *de la derivación, con* respuesta *(parcial) σ. Decimos que el par (s_n, σ) es la* salida *de la derivación.*

La Definición 2 pone de manifiesto que los programas lógico-funcionales permiten tanto el cómputo de un resultado como la obtención de una respuesta. En particular, la semántica esperada de muchos lenguajes lógico-funcionales [13, 16, 19, 40], al igual que sucede con los lenguajes funcionales, consiste en el cómputo de un término constructor básico.

Inicialmente el *narrowing* se utilizó como un medio para resolver *ecuaciones*. Fue en los trabajos pioneros de [43] donde se introdujo por primera vez el *narrowing* como medio para la obtención de un conjunto de soluciones a un problema de E-unificación. Resolver una ecuación $s \approx t$, en general, consiste en hallar los valores de las variables que permiten reducir los términos s y t a dos términos iguales. Para la clase de programas que se consideran en este trabajo, basados en constructores ortogonales y posiblemente no terminantes, esos términos se concretan en un mismo término constructor básico.

Definition 3 (Solución de una Ecuación).
Una substitución σ es una solución *de una ecuación $s \approx t$ si y solo si $\sigma(s)$ y $\sigma(t)$ son reducibles por reescritura al mismo término constructor básico.*

Teniendo en cuenta la Proposición 1, la Definición 3 puede establecerse en los siguientes términos:
Una *solución* de una ecuación $s \approx t$ es una sustitución σ tal que $\sigma(s \approx t)$ se reescribe a *true* usando las reglas del programa $\mathcal{R}_+$.

En general, el procedimiento de *narrowing* es indeterminista, debido a la existencia de dos grados de libertad: la elección del subtérmino a reducir y la elección de la regla. Esto conduce a un espacio de búsqueda demasiado amplio. Se han diseñado muchas estrategias para reducir el tamaño del espacio de búsqueda, eliminando algunas derivaciones inútiles. Es habitual definir el concepto de estrategia como una aplicación φ que asigna a cada término t un conjunto de posiciones $\varphi(t) \subseteq \mathcal{FP}os(t)$ para las que puede darse un paso de *narrowing*, i.e., una estrategia es una restricción del espacio de búsqueda. Dado que varias reglas del programa pueden aplicarse sobre una posición p seleccionada por la estrategia φ, es conveniente generalizar el concepto de estrategia para devolver ternas formadas por la propia posición p, la regla del programa R y la sustitución σ aplicada al dar el paso de *narrowing*.

Definition 4 (Estrategia de Narrowing).
Una estrategia de narrowing *es una aplicación φ que, para un término t, computa el conjunto de ternas $\langle p, R, \sigma\rangle$, donde $p \in \mathcal{FP}os(t)$, $R \equiv (l \to r)$ es la regla de programa utilizada para dar el paso de* narrowing *y σ una substitución unificadora de $t|_p$ y l* [2]*.*

Dado un término t y una regla $R \equiv (l \to r)$ del programa $\mathcal{R}$, decimos que $t \overset{[p,R,\sigma]}{\leadsto_\varphi} \sigma(t[r]_p)$ es un paso de *narrowing* acorde con la estrategia φ, si $\langle p, R, \sigma\rangle \in \varphi(t)$. Si el conjunto $\varphi(t)$ contiene un solo elemento, decimos que el paso de *narrowing* es *determinista*. De forma semejante, diremos que una

[2] Habitualmente la substitución σ se exige que sea un unificador más general, i.e., $\sigma = mgu(\{l = t|_p\})$. Hacemos esta salvedad para permitir estrategias, que como la estrategia de *narrowing* necesario, computan substituciones que no son necesariamente unificadores más generales.

derivación $t \equiv t_0 \overset{[p_1,R_1,\sigma_1]}{\leadsto_\varphi} t_1 \overset{[p_2,R_2,\sigma_2]}{\leadsto_\varphi} \ldots \overset{[p_n,R_n,\sigma_n]}{\leadsto_\varphi} t_n \equiv s$, respeta la estrategia φ, y escribimos $t \overset{\sigma}{\leadsto}_\varphi^* s$, si cada uno de sus pasos es un paso de *narrowing* acorde con la estrategia φ. Decimos que un término t es *evaluable de manera determinista* bajo la estrategia φ, si cada paso de *narrowing* en una derivación a partir de t que respeta φ, es determinista.

Una propiedad importante que debe cumplir toda estrategia es que siga manteniendo, bajo determinadas condiciones, la completitud del cálculo. Una clasificación de las diferentes estrategias y estudios detallados sobre las condiciones que debe cumplir un programa para que una determinada estrategia sea correcta y completa, pueden encontrarse en [35, 18].

En el siguiente apartado introducimos la estrategia de *narrowing perezoso*, que es correcta y completa para la clase de programas definida en la Subsección 3.1. Consideraremos que los programas se ejecutan respetando la estrategia de *narrowing* perezoso. Después, estudiaremos la estrategia de *narrowing necesario* a efectos de la comparación de ambas. *Narrowing* necesario es correcto y completo para una subclase de nuestros programas, la subclase de los programas *inductivamente secuenciales*, sobre los cuales satisface una propiedad de optimalidad que formalizaremos posteriormente.

3.3 Narrowing Perezoso

La evaluación perezosa es una característica esencial de los lenguajes lógico funcionales, ya que facilita una mayor expresividad, al permitir la definición de funciones parciales y no estrictas y la utilización de expresiones infinitas. *Narrowing* perezoso reduce las expresiones comenzando por las posiciones más externas ("outermost") sobre las que se puede dar un paso de *narrowing*. Los pasos de *narrowing* sobre posiciones más internas solamente se realizan si son demandados (por el patrón de la lhs de alguna regla) y contribuyen a un paso de *narrowing* posterior sobre una posición más externa. Dado que la noción de "posición demandada" no es única, se han propuesto diferentes estrategias de *narrowing* perezoso [12, 28, 33, 40, 42]. En lo que sigue, especificamos nuestra estrategia de *narrowing* perezoso, que en lo esencial es similar a la presentada en [40].

Las siguientes definiciones son necesarias para nuestra formalización de la estrategia de *narrowing* perezoso. Esta formalización apareció por vez primera en [2].

Primero caracterizamos una clase particular de problema de unificación que se presenta debido al hecho de trabajar con programas CB.

Definition 5 (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 $f(d_1, \ldots, d_n)$ *es un patrón lineal que no comparte variables con* $f(t_1, \ldots, t_n)$*.*

Resolvemos los problemas de unificación lineal utilizando el algoritmo presentado en [40], donde el caso en el que un intento de unificación no tiene éxito debido a que se produce una colisión entre un símbolo constructor c y uno de operación f, no se ve como un fracaso, sino como una demanda de una evaluación posterior de f. Esto difiere con respecto al algoritmo de unificación sintáctica estándar [31]. Lo que sigue es una reformulación del algoritmo de unificación para problemas de unificación lineal presentado en [40]. Debido a la linealidad de los patrones que aparecen en las lhs's de las reglas de los programas, no se precisa de ningún mecanismo de "occur-check".

Definition 6 (Configuración LU).
Una configuración LU *es un par* (U, σ)*, donde* U *es un conjunto* $\{d_1 \downarrow_1 t_1, \ldots, d_n \downarrow_n t_n\}$*, siendo* $d_1, \ldots, dn$ *términos constructores lineales que no comparten variables y* σ *una substitución. Los términos* $d_1, \ldots, dn$ *(respectívamente,* $t_1, \ldots, t_n$*) se denominan* términos lhs *(*términos rhs*) de* U*.*

Generalizamos la aplicación de una substitución σ sobre un conjunto $U \equiv \{d_1 \downarrow_1 t_1, \ldots, d_n \downarrow_n t_n\}$, en la forma obvia: $\sigma(U) = \{\sigma(d_1) \downarrow_1 \sigma(t_1), \ldots, \sigma(d_n) \downarrow_n \sigma(t_n)\}$.

Definition 7 (Relación de Unificación $\to_{\mathsf{LU}}$).
Definimos la relación de unificación $\to_{\mathsf{LU}}$*, entre configuraciones* LU*, como la relación más pequeña que satisface:*

1. $(\{c(d_1, \ldots, d_m) \downarrow_u c(t_1, \ldots, t_m)\} \cup U, \sigma) \to_{\mathsf{LU}} (\{d_1 \downarrow_{u.1} t_1, \ldots, d_m \downarrow_{u.m} t_m\} \cup U, \sigma)$,
   donde $(c/m) \in \mathcal{C}, m \geq 0$.
2. $(\{x \downarrow_u t\} \cup U, \sigma) \to_{\mathsf{LU}} (\{x/t\}(U), \{x/t\} \circ \sigma)$,
   donde $t \notin V$.
3. $(\{d \downarrow_u x\} \cup U, \sigma) \to_{\mathsf{LU}} (\{x/d\}(U), \{x/d\} \circ \sigma)$.
4. $(\{c(d_1, \ldots, d_m) \downarrow_u c'(t_1, \ldots, t_p)\} \cup U, \sigma) \quad \to_{\mathsf{LU}} (\{\mathtt{fail}\}, \sigma)$,
   donde $(c/m), (c'/p) \in \mathcal{C}$*,* $c \neq c'$*, y* $m, p \geq 0$.

Notad que la relación de unificación $\to_{\mathsf{LU}}$ está bien definida, en el sentido de que partiendo de una configuración LU que cumple los requisitos de la Definición 6 se obtiene otra configuración LU que también los cumple.

Definition 8 (Configuración Inicial LU).
Sea un problema de unificación lineal $\langle f(d_1, \ldots, d_n), f(t_1, \ldots, t_n)\rangle$*. La* configuración inicial LU *es:* $(U_0, \sigma_0) \equiv (\{d_1 \downarrow_1 t_1, \ldots, d_n \downarrow_n t_n\}, id)$*.*

Un problema de unificación lineal (LU) puede tener éxito ("success"), fallar ("fail") o quedar suspendido. Cuando queda suspendido, devuelve el conjunto de posiciones que demanda ("demand") para una evaluación posterior. Formalmente, dada una configuración LU irreducible, (U, σ), una posición u es *demandada*, si $(c(d_1, \ldots, d_m) \downarrow_u g(t_1, \ldots, t_p)) \in U$.

Definition 9 (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. Sea* $(U_0, \sigma_0) \equiv (\{d_1 \downarrow_1 t_1, \ldots, d_n \downarrow_n t_n\}, id) \to^*_{\mathsf{LU}} (U, \sigma) \not\to_{\mathsf{LU}}$*. Definimos la función* $\mathsf{LU}(\Gamma)$ *como sigue:*

$$\mathsf{LU}(\Gamma) = \begin{cases} (\textsc{Succ}, \sigma) & \texttt{si } U = \varnothing \\ (\textsc{Fail}, \varnothing) & \texttt{si } U = \{\texttt{fail}\} \\ (\textsc{Demand}, P) & \texttt{en otro caso,} \\ & \textit{donde } P \textit{ es el conjunto de posiciones demandadas} \end{cases}$$

Como en los procedimientos de prueba de la programación lógica, suponemos que las reglas que se utilizan en el proceso de unificación lineal siempre contienen variables frescas (i.e, las reglas se suponen "*renombradas aparte*"). Esto supone que todas las substituciones computadas son idempotentes.

Los siguientes lemas establecen propiedades interesantes de la relación de unificación $\rightarrow_{\mathsf{LU}}$ introducida en la Definición 7.

Lemma 1. *Sea* (U, σ) *una configuración* LU. *Sea la transición de un paso* $(U, \sigma) \rightarrow_{\mathsf{LU}} (U', \sigma')$, *donde* $U' \not\equiv \{\texttt{fail}\}$. *Entonces:*

1. *Las variables en los términos lhs de* U' *están incluidos en el conjunto de variables de los términos lhs de* U.
2. *Si los términos lhs de* U *no comparten variables con los términos rhs de* U *o con* σ, *entonces los términos lhs de* U' *no comparten variables con los otros términos lhs o con los términos rhs de* U', *ni con* σ' [3].

Proof. La primera parte del enunciado de este lema es inmediata por la Definición 6 de configuración LU y la Definición 7 de relación de unificación lineal. Para la prueba de la segunda parte consideramos tres casos:

1. Si se aplica la regla (1) de la Definición 7, entonces el resultado se sigue de forma inmediata.
2. Supongamos que se aplica la regla (2) de la Definición 7, y consideremos la transición $(U, \sigma) \equiv (\{x \downarrow_u t\} \cup U^\star, \sigma) \rightarrow_{\mathsf{LU}} (\{x/t\}(U^\star), \{x/t\} \circ \sigma) \equiv (U', \sigma')$. Ya que los términos lhs de U no comparten variables con los otros términos lhs o con los términos rhs de U, entonces $U' = \{x/t\}(U^\star) = U^\star$, y entonces $U' \subseteq U$ (por lo que $\mathcal{V}ar(U') \subseteq \mathcal{V}ar(U)$). Más aún, ya que los términos lhs de U' no comparten variables con σ, x y t, entonces no comparten variables con σ'.
3. Consideremos ahora que se aplica la regla (3) de la Definición 7, entonces la transición es $(U, \sigma) \equiv (\{d \downarrow_u x\} \cup U^\star, \sigma) \rightarrow_{\mathsf{LU}} (\{x/d\}(U^\star), \{x/d\} \circ \sigma) \equiv (U', \sigma')$. Ya que los términos lhs de U no comparten variables con los otros términos lhs o con los términos rhs de U, o con σ, entonces la sustitución $\{x/d\}$ no instancia los términos lhs de $U^\star$, y así las variables de los términos lhs de U' están incluidas en el conjunto de variables de los términos lhs de U, y los términos lhs de U' no comparten variables con los otros términos lhs o los términos rhs de U', ni con σ'.

Es importante notar que, como consecuencia del Lema 1, la regla (2) de la Definición 7 de relación de unificación $\rightarrow_{\mathsf{LU}}$ puede simplificarse de la siguiente forma: $(\{x \downarrow_u t\} \cup U, \sigma) \rightarrow_{\mathsf{LU}} (U, \{x/t\} \circ \sigma)$, donde $t \notin V$. En efecto, $x \notin \mathcal{V}ar(U)$.

[3] Esto es, no comparten variables ni con el domino ni con el rango de σ'.

Lemma 2. *Sea* $\langle l, s\rangle \equiv \langle f(d_1,\ldots,d_k), f(t_1,\ldots,t_k)\rangle$ *un problema de unificación lineal. Sea* $(U_0,\epsilon) \equiv (\{d_1 \downarrow_1 t_1,\ldots,d_k \downarrow_k t_k\},\epsilon) \to^*_{\mathsf{LU}} (U_n,\sigma_n)$ *una derivación* LU, *donde* $U_n \not\equiv \{\mathtt{fail}\}$. *Entonces, para cada* $x, y \in \mathit{Dom}(\sigma_{n\lceil\mathcal{V}ar(s)})$, *con* $x \neq y$, $\sigma_n(x)$ *es un término constructor lineal que no comparte variables ni con* s *ni con* $\mathcal{V}ar(\sigma_n(y))$, *y* $\mathcal{V}ar(\sigma_n(x)) \subseteq \mathcal{V}ar(l)$.

Proof. Probamos este lema por inducción en el número de pasos n de la derivación LU. Ya que el caso base $n = 0$ es trivial, pasamos a considerar el caso inductivo $n > 0$.

Supongamos $(U_0,\epsilon) \to^*_{\mathsf{LU}} (U_{n-1},\sigma_{n-1}) \to_{\mathsf{LU}} (U_n,\sigma_n)$, $n > 0$, con $U_n \not\equiv \{\mathtt{fail}\}$. En primer lugar, ya que las condiciones del Lema 1 se cumplen para la configuración inicial (U_0,ϵ), mediante la aplicación reiterada de este lema, obtenemos que los términos lhs de U_{n-1} no comparten variables ni con los otros términos lhs ni tampoco con los términos rhs de U_{n-1}, o con σ_{n-1}. Más aún, también por el Lema 1, las variables en los términos lhs de cada configuración U_i, $0 \leq i \leq n$, provienen del término (estandarizado aparte) l que no comparte variables con s. También notad que los términos lhs de cada configuración U_i son términos constructores lineales, ya que son subtérminos de l que no son instanciados a lo largo de la derivación.

Consideremos el último paso $(U_{n-1},\sigma_{n-1}) \to_{\mathsf{LU}} (U_n,\sigma_n)$. Distinguimos tres casos:

1. Si se aplica la regla (1) de la Definición 7, entonces el resultado se sigue directamente por la hipótesis de indución, ya que $\sigma_n = \sigma_{n-1}$.
2. Supongamos que se aplica la regla (2) de la Definición 7, y consideremos la transición $(U_{n-1},\sigma_{n-1}) \equiv (\{x \downarrow_u t\} \cup U^\star,\sigma_{n-1}) \to_{\mathsf{LU}} (U^\star\{x/t\},\sigma_{n-1}\{x/t\}) \equiv (U_n,\sigma_n)$. Ya que $x \notin \mathcal{V}ar(\sigma_{n-1})$, entonces $\sigma_n = \sigma_{n-1} \cup \{x/t\}$ y, puesto que $x \in \mathcal{V}ar(l)$, entonces $\sigma_n = \sigma_{n-1}[\mathcal{V}ar(s)]$, y el resultado se sigue por la hipótesis de indución.
3. Consideremos ahora que se aplica la regla (3) de la Definición 7, y la transición es $(U_{n-1},\sigma_{n-1}) \equiv (\{d \downarrow_u x\} \cup U^\star,\sigma_{n-1}) \to_{\mathsf{LU}} (U^\star\{x/d\},\sigma_{n-1}\{x/d\}) \equiv (U_n,\sigma_n)$. Tenemos que considerar dos casos:
   (a) Sea $x \in \mathcal{V}ar(l)$. Por la hipótesis de indución, los términos en el codominio de $\sigma_{n-1\lceil\mathcal{V}ar(s)}$ son términos constructores lineales que no comparten variables ni con s ni tampoco entre ellos mismos. Además, para toda $x \in \mathit{Dom}(\sigma_{n-1\lceil\mathcal{V}ar(s)})$, $\mathcal{V}ar(x\sigma_{n-1}) \subseteq \mathcal{V}ar(l)$. Por el Lema 1, d no comparte variables con σ_{n-1} y, por lo tanto, los términos en el codominio de $\sigma_{n\lceil\mathcal{V}ar(s)}$ son constructores lineales, no comparten variables y, para toda $x \in \mathit{Dom}(\sigma_{n\lceil\mathcal{V}ar(s)})$, $\mathcal{V}ar(x\sigma_n) \subseteq \mathcal{V}ar(l)$.
   (b) Sea $x \in \mathcal{V}ar(s)$. Por la hipótesis de indución, la variable x no aparece en ningún término en el codominio de $\sigma_{n-1\lceil\mathcal{V}ar(s)}$, y así $\sigma_n = \sigma_{n-1} \cup \{x/d\}[\mathcal{V}ar(s)]$. Ahora, por el Lema 1, d no comparte variables con σ_{n-1}, y por consiguiente el resultado se sigue directamente por hipótesis de indución.

Vamos a definir la estrategia de *narrowing* perezoso que empleamos para computar el conjunto de posiciones perezosas de un término t. Informalmente, la función $\lambda_{lazy}(t)$ devuelve el conjunto de ternas $\langle p, R_k, \sigma\rangle$ tal que $p \in \mathcal{FP}os(t)$ es una posición perezosa de t que puede ser reducida por *narrowing* mediante la regla R_k utilizando la substitución σ. Suponemos que las reglas de $\mathcal{R}$ están numeradas como $R_1, \ldots, R_m$.

Definition 10 (Estrategia de Narrowing Perezoso).
Definimos la estrategia de narrowing perezoso *como una función* λ_{lazy} *que, aplicada a un término* t*, computa el conjunto de ternas* $\langle p, R_k, \sigma\rangle$*, siendo* $p \in \mathcal{FP}os(t)$ *una posición perezosa de* t*,* $R_k \equiv (l_k \to r_k)$ *es una regla (renombrada aparte) de* $\mathcal{R}$ *y* σ *una substitución, como sigue:*

$$
\begin{array}{ll}
\lambda_{lazy}(t) & = \bigcup_{k=1}^{m} \lambda\_(t, \Lambda, R_k) \\
\lambda\_(t, p, R_k) = & \texttt{si } \mathcal{H}ead(l_k) = \mathcal{H}ead(t|_p) \texttt{ entonces} \\
& \quad \texttt{en caso de que } \mathsf{LU}(\langle l_k, t_{|p}\rangle) = \\
& \qquad \begin{cases} (\texttt{Succ}, \sigma): & \{\langle p, R_k, \sigma\rangle\} \\ (\texttt{Fail}, \varnothing): & \varnothing \\ (\texttt{Demand}, P): & \bigcup_{q \in P} \bigcup_{k=1}^{m} \lambda\_(t, p.q, R_k) \end{cases} \\
& \texttt{sino } \varnothing
\end{array}
$$

Como ya se ha mencionado, existen varias fuentes de indeterminismo en el cálculo de *narrowing*: la elección del subtérmino reducible (redex) dado por p y la elección de la regla del programa R_k. Ambas, son fuentes de indeterminismo "don't-know", lo que significa que, en general, todos los posibles puntos de elección deben explotarse para asegurar la completitud del proceso. *Narrowing* perezoso es *fuertemente completo* ("strong complete" [17, 36, 37]) con respecto a substituciones constructoras para TRS's CB y ortogonales [40, 18]. La completitud fuerte significa que la selección de un subconjunto de posiciones perezosas dentro de los subtérminos t_i de un término conjuntivo $t_1 \wedge t_2 \wedge \ldots \wedge t_n$ puede realizarse de forma "don't-care". Por lo tanto, podemos seleccionar las posiciones perezosas del subtérmino t_i y desestimar el resto de posiciones perezosas en los subtérminos t_j, con $j \neq i$. En particular, el subtérmino t_i seleccionado para su inspección puede ser el que, poseyendo un subconjunto de posiciones perezosas, se encuentre más a la izquierda en la conjunción. Nosotros simulamos esta selección "don't-care" introduciendo en los programas una única regla para definir el símbolo de función predefinido "$\wedge$" (i.e., $\mathit{true} \wedge x \to x$), que garantiza que se seleccionará el subconjunto de posiciones perezosas situado más a la izquierda en un término conjuntivo.

Después de este trabajo preparatorio, ya estamos en disposición de definir formalmente qué se entiende por *narrowing* perezoso.

Definition 11 (Narrowing Perezoso).
Definimos narrowing perezoso *como un sistema de transición etiquetado cuya relación de transición* $\leadsto_{LN} \subseteq (\mathcal{T} \times \lambda_{lazy}(\mathcal{T}) \times \mathcal{T})$ *es la relación más pequeña que satisface:*

$$\frac{\langle p, R, \sigma\rangle \in \lambda_{lazy}(t) \wedge \ R \equiv (l \to r) \ll \mathcal{R}}{t \overset{[p,R,\sigma]}{\leadsto}_{LN} \ \sigma(t[r]_p)}$$

Si $s_0 \overset{[p_1,R_1,\sigma_1]}{\leadsto_{LN}} s_1 \overset{[p_2,R_2,\sigma_2]}{\leadsto_{LN}} \ldots \overset{[p_n,R_n,\sigma_n]}{\leadsto_{LN}} s_n$, escribimos, $s_0 \overset{\sigma}{\leadsto}_{LN}^{*} s_n$, siendo $\sigma = \sigma_n \circ \ldots \circ \sigma_2 \circ \sigma_1$ y hablamos de que existe una *derivación de narrowing perezoso* para el término s_0 con *resultado* (parcial) s_n y *respuesta* σ. Principalmente, estamos interesados en aquellas derivaciones que conducen a un término constructor y en la restricción de las respuestas a las variables del término inicial. Una derivación de *narrowing* perezoso $s \overset{\sigma}{\leadsto}_{LN}^{*} t$ tiene *éxito* si y sólamente si $t \in \mathcal{T}(\mathcal{C} \cup \mathcal{X})$, donde $\sigma_{\restriction \mathcal{V}ar(s)}$ es la *substitución de respuesta computada* asociada a esa derivación.

Example 2. Consideremos de nuevo las reglas que definen el predicado "$\leqslant$" en el Ejemplo 1, junto con las siguientes reglas que definen la adición sobre los números naturales:

$$\begin{array}{rcl} 0 + N & \to & N \\ s(M) + N & \to & s(M + N) \end{array}$$

Entonces *narrowing* perezoso evalúa el término $X \leqslant X + X$ aplicando un paso de *narrowing* sobre la posición Λ, con la primera regla de "$\leqslant$", o aplicando un paso de *narrowing* sobre la posición 2, ya que el argumento $X+X$ es demandado por la segunda y la tercera reglas de "$\leqslant$". De este modo, pueden darse tres pasos de *narrowing* perezoso diferentes a partir del término inicial:

$$\begin{array}{ll} X \leqslant X + X \overset{\{X \mapsto 0\}}{\leadsto_{LN}} & true \\ X \leqslant X + X \overset{\{X \mapsto 0\}}{\leadsto_{LN}} & 0 \leqslant 0 \\ X \leqslant X + X \overset{\{X \mapsto s(M)\}}{\leadsto_{LN}} & s(M) \leqslant s(M + s(M)) \end{array}$$

Notad que el segundo paso puede considerarse en cierto sentido superfluo, ya que conduce al mismo resultado (*true*) con la misma respuesta que el primer paso. Este hecho introduce una fuente de ineficiencia en la estrategia de *narrowing* perezoso, que también se pone de manifiesto en otros contextos.

Para programas CB y ortogonales, la estrategia de *narrowing* perezoso es completa con respecto a la igualdad estricta para substituciones constructoras:

Theorem 1. *[40] Sea $\mathcal{R}$ un programa CB y ortogonal, e una ecuación, y σ una substitución constructora que es una solución para e. Entonces existe una derivación de* narrowing *perezoso $e \overset{\sigma'}{\leadsto}_{LN}^{*}$ true tal que $\sigma' \leq \sigma$ $[\mathcal{V}ar(e)]$.*

Para finalizar este apartado, presentamos una propiedad interesante de nuestra estrategia que utilizaremos más adelante.

Proposition 2. *Sea un programa $\mathcal{R}$ y un término s. Sea $\mathcal{D} \equiv (s \overset{\sigma}{\leadsto}_{LN}^{*} t)$ una derivación de* narrowing *perezoso para s en $\mathcal{R}$. Entonces, para toda $x, z \in Dom(\sigma_{\restriction \mathcal{V}ar(s)})$, con $x \neq z$, se cumple que $\sigma(x)$ y $\sigma(z)$ son términos constructores lineales, y $\mathcal{V}ar(\sigma(x)) \cap \mathcal{V}ar(\sigma(z)) = \varnothing$.*

Proof. Sea $\mathcal{D} \equiv (s \equiv s_0 \overset{[p_1,R_1,\sigma_1]}{\rightsquigarrow}_{LN} s_1 \overset{[p_2,R_2,\sigma_2]}{\rightsquigarrow}_{LN} \ldots \overset{[p_n,R_n,\sigma_n]}{\rightsquigarrow}_{LN} s_n \equiv t)$ y $\sigma = \sigma_n \circ \ldots \circ \sigma_1$. Primero hay que notar que $\sigma_{\restriction \mathcal{V}ar(s)} = \sigma_{n \restriction \mathcal{V}ar(s_{n-1})} \circ \ldots \circ \sigma_{1 \restriction \mathcal{V}ar(s_0)}$. Ahora, por el Lema 2, para todo par de variables distintas $x, z \in Dom(\sigma_{i \restriction \mathcal{V}ar(s_{i-1})})$, $\sigma_i(x)$ es un término constructor lineal que no comparte variables ni con s_{i-1} ni con $\sigma_i(z)$, y $\mathcal{V}ar(\sigma_i(x)) \subseteq \mathcal{V}ar(l_i)$ (siendo l_i la lhs de la correspondiente regla R_i), para $i = 1, \ldots, n$. Por consiguiente, los términos en los codominios de $\sigma_{1 \restriction \mathcal{V}ar(s_0)}, \ldots, \sigma_{n \restriction \mathcal{V}ar(s_{n-1})}$ no comparten variables y portanto los términos en el codomino de $\sigma_{\restriction \mathcal{V}ar(s)}$ son también términos constructores lineales que no comparten variables.

3.4 Narrowing Necesario

En lo que sigue vamos a resumir los resultados fundamentales relativos a la estrategia de *narrowing* necesario [6, 11, 12]. *Narrowing necesario* es una estrategia que se basa en la selección de las posiciones necesarias más externas ("outermost") de un término para dar un paso de *narrowing*, de modo que solo se dan pasos inevitables para el cómputo de un resultado o la solución de una ecuación. *Narrowing* necesario es destacable por sus propiedades de optimalidad con respecto a la longitud de las derivaciones de éxito (en implementaciones basadas en grafos) y el número de soluciones computadas.

Narrowing necesario se define para la clase de los *programas inductivamente secuenciales*. Esta clase de programas es un subconjunto de los programas CB ortogonales y recientemente se ha probado que coincide con la clase de los *programas CB fuertemente secuenciales* [20]. Para proporcionar una definición precisa de esta clase de programas y de la estrategia de *narrowing* necesario, introducimos el concepto de árbol definicional. En lugar de la definición original [8], presentamos una definición "declarativa" introducida en [9, 10] que es más útil para la demostración de ciertas propiedades que deseamos estudiar.

Definition 12 (Arbol Definicional).
Un árbol definicional *de un conjunto finito de patones lineales S, es un conjunto no vacío $\mathcal{P}$ de patrones lineales ordenado por el orden $<$ (de generalidad relativa estricta) y que cumple las siguientes propiedades:*

Propiedad de la raíz: Existe un elemento mínimo $pattern(\mathcal{P})$, que denominamos patrón *del árbol definicional.*

Propiedad de las hojas: Los elementos maximales, llamados hojas*, son los elementos de S. Los elementos no maximales se denominan* ramas*.*

Propiedad de los padres: Si $\pi \in \mathcal{P}$, $\pi \neq pattern(\mathcal{P})$, entonces existe un único $\pi' \in \mathcal{P}$, llamado padre *de π (y π se denomina* hijo *de π'), tal que $\pi' < \pi$ y no existe otro patrón $\pi'' \in \mathcal{T}(\mathcal{C} \cup \mathcal{F}, \mathcal{X})$ con $\pi' < \pi'' < \pi$.*

Propiedad inductiva: Dado un patrón $\pi \in \mathcal{P} \backslash S$, existe una posición o en π con $\pi|_o \in \mathcal{X}$ (llamada posición inductiva*) y constructores $c_1, \ldots, c_n \in \mathcal{C}$ con $c_i \neq c_j$ para $i \neq j$, tal que, para todo $\pi_1, \ldots, \pi_n$ que tiene por padre a π, $\pi_i = \pi[c_i(x_1, \ldots, x_{n_i})]_o$ (donde $x_1, \ldots, x_{n_i}$ son variables nuevas y distintas) para todo $1 \leq i \leq n$.*

Si $\mathcal{R}$ es un TRS ortogonal y f/n una función definida, decimos que $\mathcal{P}$ es un *árbol definicional de* f, si $pattern(\mathcal{P}) = f(x_1, \ldots, x_n)$, donde $x_1, \ldots, x_n$ son variables distintas, y las hojas de $\mathcal{P}$ son todas y únicamente variantes de las lhs's de las reglas de $\mathcal{R}$ que definen f.

Definition 13 (Inductivamente Secuencial).
Sea f una función definida en un TRS $\mathcal{R}$, la función f se dice que es inductivamente secuencial *si existe un árbol definicional para f. Un TRS $\mathcal{R}$ se denomina* inductivamente secuencial *si todas las funciones que se definen en $\mathcal{R}$ son inductivamente secuenciales.*

Un TRS inductivamente secuencial puede verse como un conjunto de árboles definicionales, cada uno de los cuales define un símbolo de función. Dado el indeterminismo, en su formación, puede haber más de un árbol definicional para una función inductivamente secuencial. En lo que sigue, para cada función definida inductivamente secuencial, supondremos fijado el árbol definicional entre uno cualquiera de los varios existentes.

Como ya se ha indicado no todo TRS ortogonal y CB es inductivamente secuencial. Notad, por ejemplo, que el TRS de *Berry* [26]

$$
\begin{array}{l}
f(a, b, X) \to c \\
f(b, X, a) \to c \\
f(X, a, b) \to c
\end{array}
$$

donde a, b y c se consideran símbolos constructores, es claramente un TRS CB ortogonal pero no es inductivamente secuencial, ya que es imposible construir un árbol definicional para f.

Una representación gráfica de los árboles definicionales puede facilitar su entendimiento. Es habitual asociar a cada nodo un patrón y marcar, mediante un recuadro, cada posición inductiva de una rama. Finalmente, las hojas contienen las reglas correspondientes. La Figura 1 ilustra el árbol definicional para la función "$\leqslant$" del Ejemplo 1.

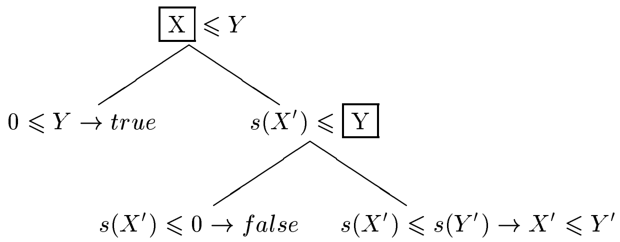

Fig. 1. Arbol definicional para la función "$\leqslant$"

Ahora estamos en disposición de definir qué entendemos por estrategia de *narrowing* necesario. La descripción que realizamos de la estrategia de *narrowing* necesario, extraída de [6] (y similar a la que aparece en [10]), es ligeramente diferente a la presentada en [12], pero conduce a los mismos pasos de needed *narrowing* en una derivación.[4]

Definition 14 (Estrategia de Narrowing Necesario). *Sea $\mathcal{R}$ un TRS inductivamente secuencial. Sea t un término encabezado por un símbolo de operación y $\mathcal{P}$ un árbol definicional con pattern($\mathcal{P}$) $= \pi$ tal que $\pi \leq t$. Definimos una aplicación λ de términos y árboles definicionales a conjuntos de ternas (posición, regla, substitución) como el menor conjunto que satisface las siguientes propiedades. Consideramos dos casos para $\mathcal{P}$:*

1. *Si π es una hoja, i.e., $\mathcal{P} = \{\pi\}$, y $\pi \to r$ es una variante de una regla de $\mathcal{R}$, entonces $\lambda(t, \mathcal{P}) = \{\langle \Lambda, \pi \to r, id\rangle\}$.*
2. *Si π es una rama, considerar la posición inductiva o de π y un hijo $\pi_i = \pi[c_i(x_1, \ldots, x_n)]_o \in \mathcal{P}$. Sea $\mathcal{P}_i = \{\pi' \in \mathcal{P} \mid \pi_i \leq \pi'\}$ el árbol definicional donde todos los patrones son instancias de π_i. Entonces consideramos los siguientes casos para el subtérmino $t|_o$:*

$$\lambda(t,\mathcal{P}) \ni \begin{cases} \langle p, R, \sigma \circ \tau\rangle & \texttt{si } t|_o = x \in \mathcal{X},\ \tau = \{x/c_i(x_1,\ldots,x_n)\}, \\ & \quad \texttt{y } (p,R,\sigma) \in \lambda(\tau(t), \mathcal{P}_i); \\ \langle p, R, \sigma \circ id\rangle & \texttt{si } t|_o = c_i(t_1,\ldots,t_n) \texttt{ y } (p,R,\sigma) \in \lambda(t, \mathcal{P}_i); \\ \langle o.p, R, \sigma \circ id\rangle & \texttt{si } t|_o = f(t_1,\ldots,t_n),\ f \in \mathcal{F} \texttt{ y } (p,R,\sigma) \in \lambda(t|_o, \mathcal{P}') \\ & \quad \text{donde } \mathcal{P}' \text{ es un árbol definicional para } f. \end{cases}$$

Informalmente, el *narrowing* necesario aplica una regla si es posible (caso 1) o comprueba los subtérminos correspondientes a las posiciones inductivas de una rama (case 2): si dicho subtérmino es una variable, queda instanciada con el constructor de un hijo (formando lo que denominamos un enlace de *substitución adelantada*); si ya es un constructor, procede con el hijo que le corresponde (aquél que en la misma posición inductiva presenta un término constructor plano encabezado por el mismo símbolo constructor); si es una función, se evalúa aplicando recursivamente la definición de *narrowing* necesario. Así pues, la estrategia difiere de otras estrategias perezosas en la instanciación de variable libres mediante substituciones adelantadas. Este hecho hace que la estrategia compute respuestas que no son unificadores más generales.

Para computar pasos de *narrowing* necesario para un término t encabezado por un símbolo de función f, tomamos el árbol definicional $\mathcal{P}$ de f y computamos $\lambda(t, \mathcal{P})$. Entonces, para cada $(p, R, \sigma) \in \lambda(t, \mathcal{P})$, el paso $t \overset{p,R,\sigma}{\leadsto}_{NN} t'$ es un *paso de narrowing necesario.*

Example 3. Consideremos, de nuevo, las reglas para "$\leqslant$" y "+" del Ejemplo 2. Entonces, la estrategia λ computa para el término $X \leqslant X + X$ el siguiente conjunto de ternas:

$$\{(\Lambda, 0 \leqslant N \to true, \{X \mapsto 0\}),\ (2, s(M) + N \to s(M+N), \{X \mapsto s(M)\})\}$$

[4] La estrategia λ de la Definición 14 computa las mismas ternas, que la estrategia original, definida en [12], si en las substituciones nos restringimos a las variables del término inicial y no tenemos en cuenta los posibles renombramientos de variables. Para que la Definición 14 se ajuste exáctamente a la original, basta con que, en el punto (1), $\lambda(t, \mathcal{P})$ devuelva $\{\langle \Lambda, \pi \to r, mgu(t, \pi)\rangle\}$ en lugar de $\{\langle \Lambda, \pi \to r, id\rangle\}$.

que se corresponden con los siguientes pasos de *narrowing*:

$$X \leqslant X + X \overset{\{X/0\}}{\leadsto}_{NN} \quad true$$
$$X \leqslant X + X \overset{\{X/s(M)\}}{\leadsto}_{NN} \quad s(M) \leqslant s(M + s(M))$$

La comparación de estos pasos de *narrowing* con los de las derivaciones obtenidas en el Ejemplo 2, nos muestra que needed *narrowing* solamente da pasos necesarios para el cómputo de una respuesta.

En lo que sigue, resumimos una serie de propiedades interesantes de la estrategia de *narrowing* necesario que nos serán muy útiles en el futuro. La primera proposición muestra que cada substitución en un paso de *narrowing* necesario instancia solamente variables que aparecen en el término inicial. Antes de presentar esta proposición, reparemos en que, en cada paso recursivo i durante el cómputo de λ, componemos la subtitución actual ϑ_i (que puede ser la identidad) con las computadas en las llamadas recursivas previas $\vartheta_k \circ \cdots \circ \vartheta_{i-1}$. De este modo, cada paso de *narrowing* necesario puede representarse como $(p, R, \vartheta_k \circ \cdots \circ \vartheta_1)$, que denominamos *representación canónica* de un paso de *narrowing* necesario. Como en los procedimientos de prueba de la programación lógica, suponemos que los árboles definicionales siempre contienen variables frescas cuando se emplean en el cómputo de un paso de *narrowing*. Esto supone que todas las substituciones computadas son idempotentes.

Proposition 3. *[6] Si $(p, R, \vartheta_k \circ \cdots \circ \vartheta_1) \in \lambda(t, \mathcal{P})$ es un paso de* narrowing *necesario, entonces, para $i = 1, \ldots, k$, se tiene que $\vartheta_i = id$ o bien $\vartheta_i = \{x/c(x_1, \ldots, x_n)\}$ (donde $x_1, \ldots, x_n$ son variables distintas) con $x \in \mathcal{V}ar(\vartheta_{i-1} \circ \cdots \circ \vartheta_1(t))$.*

Se debe observar que de la Proposición 3 se deriva que los términos $c(x_1, \ldots, x_n)$ del rango de las substituciones ϑ_i son constructores planos lineales y, puesto que las variables $x_1, \ldots, x_n$ provienen de reglas renombradas aparte del programa, no comparten variables con otros términos del rango de otras substituciones ϑ_j con $i \neq j$. Notad también que, si $\vartheta_i = \{x/c(x_1, \ldots, x_n)\}$, dado que $x \in \mathcal{V}ar(\vartheta_{i-1} \circ \cdots \circ \vartheta_1(t))$, la variable x puede pertenecer a las variables del rango de los componenetes ϑ_j (con $j \leq i$) hallados previamente.

Corollary 1. *Sea $\mathcal{R}$ un programa inductivamente secuencial y t y s dos términos. Si $\mathcal{D} \equiv (t \overset{\sigma}{\leadsto}_{NN}^{*} s)$ es una derivación de* narrowing *necesario, entonces para todo par de variables distintas $x, y \in \mathcal{D}om(\sigma_{\lceil \mathcal{V}ar(t)})$, $\sigma(x)$ y $\sigma(y)$ son términos constructores lineales que no comparten variables.*

Proof. Por inducción en el número de pasos de la derivación $\mathcal{D}$.

1. Caso base ($n = 1$). La derivación $\mathcal{D}$ es de un sólo paso $t \overset{[p,R,\sigma]}{\leadsto}_{NN} s$. Por definición de paso de *narrowing* necesario y la Proposición 3, los términos del rango de la susbtitución σ son constructores lineales que no comparten variables.

2. Caso inductivo ($n > 1$). La derivación $\mathcal{D}$ tiene longitud n

$$t \equiv s_0 \overset{[p_1,R_1,\sigma_1]}{\rightsquigarrow_{NN}} s1 \overset{[p_2,R_2,\sigma_2]}{\rightsquigarrow_{NN}} \ldots \overset{[p_n,R_n,\sigma_n]}{\rightsquigarrow_{NN}} s_n \equiv s,$$

donde $\sigma = \sigma_n \circ \ldots \circ \sigma_2 \circ \sigma_1$. Ahora podemos dividir la derivación $\mathcal{D}$ en dos partes:

$$t \equiv s_0 \overset{\sigma'}{\rightsquigarrow}_{NN}^{*} s_{n-1} \overset{[p_n,R_n,\sigma_n]}{\rightsquigarrow_{NN}} s_n \equiv s,$$

donde $\sigma' = \sigma_{n-1} \circ \ldots \circ \sigma_1$. Notad también que, si $R_i \equiv l_i \to r_i$ entonces $s_{n-1} \equiv \sigma_{n-1}(\ldots(\sigma_2(\sigma_1(s_0[r_1]_{p_1})[r_2]_{p_2} \ldots)[r_{n-1}]_{p_{n-1}})$, por lo que si $x \in \mathcal{D}om(\sigma')$ entonces $x \notin \mathcal{V}ar(s_{n-1})$. Por consiguiente, teniendo en cuenta la aclaración anterior y el hecho de que las reglas del programa se renombran aparte, $\mathcal{D}om(\sigma') \cap \mathcal{D}om(\sigma_n) = \varnothing$. Asi pues, $\sigma_{\restriction \mathcal{V}ar(t)} = \sigma_n \circ (\sigma'_{\restriction \mathcal{V}ar(t)}) \cup \sigma_{n \restriction \mathcal{V}ar(t)}$.
Por hipotesis de inducción, para toda variable x e y distintas de $\mathcal{D}om(\sigma'_{\restriction \mathcal{V}ar(t)})$, los términos $\sigma'(x)$ y $\sigma'(y)$ son constructores lineales que no comparten variables. Además, las variables de los términos del rango de $\sigma'_{\restriction \mathcal{V}ar(t)}$ son variables que provienen de reglas renombradas aparte R_i de $\mathcal{R}$ (con $i < n$).
Por la Proposición 3, los términos del rango de $\sigma_{n \restriction \mathcal{V}ar(s_{n-1})}$ son constructores lineales que no comparten variables entre ellos. Tampoco con los términos del rango de $\sigma'_{\restriction \mathcal{V}ar(t)}$, ya que las variables que aparecen en el rango de $\sigma_{n \restriction \mathcal{V}ar(s_{n-1})}$ son variables nuevas que provienen de la regla renombrada aparte R_n. En particular, los términos del rango de $\sigma_{n \restriction \mathcal{V}ar(t)}$ son constructores lineales que no comparten variables entre ellos, ni con los términos del rango de $\sigma'_{\restriction \mathcal{V}ar(t)}$.
Por otro lado, las variables de $\mathcal{D}om(\sigma_n)$ son variables del término s_{n-1} o variables nuevas que provienen de la regla renombrada aparte R_n. Esto es, $\sigma_n = (\sigma_{n \restriction \mathcal{V}ar(s_{n-1})}) \cup \sigma_{n \restriction \mathcal{V}ar(R_n)}$. Entonces, la composición de σ_n sobre $\sigma'_{\restriction \mathcal{V}ar(t)}$ siempre introduce subterminos del rango de $\sigma_{n \restriction \mathcal{V}ar(s_{n-1})}$ que, como ya hemos dicho, son constructores lineales que no comparten variables ni con los términos del rango de $\sigma'_{\restriction \mathcal{V}ar(t)}$ ni con los otros términos del rango de $\sigma_{n \restriction \mathcal{V}ar(t)}$. Por consiguiente, los términos del rango de $\sigma_{\restriction \mathcal{V}ar(t)}$ son constructores lineales que no comparten variables.

Para programas inductivamente secuenciales, el *narrowing* necesario es correcto y completo con respecto a ecuaciones estrictas y substituciones constructoras como solución de dichas ecuaciones. Más aún, el *narrowing* necesario no computa soluciones redundantes. Estas propiedades se formalizan en el siguiente teorema:

Theorem 2. *[12] Sea $\mathcal{R}$ un programa inductivamente secuencial y e una ecuación.*

1. *(Corrección) Si $e \rightsquigarrow_{\sigma}^{*}$ true es una derivación de* narrowing *necesario, entonces σ es una solución para e.*
2. *(Completitud) Para cada substitución constructora σ que es una solución de e, existe una derivación de* narrowing *necesario $e \rightsquigarrow_{\sigma'}^{*}$ true con $\sigma' \leq \sigma$ $[\mathcal{V}ar(e)]$.*

3. *(Minimalidad) Si* $e \rightsquigarrow^*_\sigma$ *true and* $e \rightsquigarrow^*_{\sigma'}$ *true son dos derivaciones de* narrowing *necesario distintas, entonces* σ *y* σ' *son independientes (i.e., existe al menos un* $x \in \mathcal{V}ar(e)$ *tal que* $\sigma(x) \neq \sigma'(x)$*).*

4 Programas Uniformes.

Los programas uniformes Babel fueron introducidos en [29, 30]. La *uniformidad* es una restricción sintáctica que premite una implementación eficiente de la estrategia de *narrowing* perezoso. A continuación presentamos una definición de programa uniforme ligeramente diferente a la presentada en [30] y adaptada a la clase de programas de primer orden sin tipos con los que trabajamos[5].

Definition 15 (Programa uniforme). *Un programa* uniforme *consiste en un conjunto de reglas* $f(t_1, \ldots, t_n) \rightarrow r$ *que cumplen las siguientes restricciones:*

1. *Patrones* constructores planos*: cada* t_i *es una variable* x *o un constructor* $c(x_1, \ldots, x_n)$*. En el último caso, se dice que* f *demanda el* i*-esimo argumento y que* c *es el* constructor demandante*.*
2. Linealidad por la izquierda*: las partes izquierdas de las reglas,* $f(t_1, \ldots, t_n)$*, no contienen múltiples ocurrencias de la misma variable.*
3. *Restricción de* variables libres*: las partes derechas de las reglas,* r*, no contienen variables libres (i.e., variables que no aparecen en la parte izquierda).*
4. Uniformidad*: sean* $f(t_1, \ldots, t_n)$ *y* $f(s_1, \ldots, s_n)$ *partes izquierdas de las reglas que definen* f*, entonces,* t_i *es una variable si y sólo si* s_i *es una variable.*
5. No-ambigüedad*: Si* $l_1 \rightarrow r_1$ *y* $l_2 \rightarrow r_2$ *son dos reglas distintas cualesquiera que definen* f*,* l_1 *y* l_2 *no son unificables*[6]*.*

Lemma 3 (Estructura de los Programas Uniformes). *Sea* $\mathcal{R}$ *un programa uniforme. Sea* $L_f = \{l \mid l \rightarrow r \in \mathcal{R} \wedge \mathcal{H}ead(l) = f\}$ *el conjunto de las lhs's de las reglas que definen la función* f *en* $\mathcal{R}$*.*

1. *Sea una posición* $k \in \{1, \ldots, n\}$*. Para toda* $l \in L_f$*,* $l|_k$ *es una variable, o bien para toda* $l \in L_f$*,* $l|_k$ *es un constructor lineal plano.*
2. *Si en las lhs's de las reglas que definen* f *aparecen argumentos con términos constructores planos, dadas dos lhs's* $l_1 \in L_f$ *y* $l_2 \in L_f$ *correspondientes a reglas distintas cualesquiera, existe un argumento* $k \in \{1, \ldots, n\}$ *tal que*

[5] Hemos cambiado la condición de *no-ambigüedad débil*, de la definición original en [30], por la condición de *no-ambigüedad (fuerte)*, para impedir que en un programa aparezcan reglas que sean un renombramiento una de otra. Este tipo de reglas son las únicas cuyas lhs's pueden unificar cumpliendo la restricción de no-ambigüedad débil en programas uniformes. Asi pues, para esta clase de programas, considerar que las reglas que son renombramientos son la misma regla, equivale a exigir la condición de no ambiguedad (fuerte).

[6] Esta definición es una particularización de la dada en los preliminares, ya que para TRS CB y ortogonales, al ser los argumentos de las lhs's de las reglas términos constructores, la única posibilidad de que dos reglas solapen es que lo hagan sobre la posición Λ.

$l_1|_k = c_1(x_1, \ldots, x_{n_1})$ y $l_2|_k = c_2(y_1, \ldots, y_{n_2})$ con $c_1 \neq c_2$. En caso contrario, f está definida por una sóla regla $f(x_1, \ldots, x_n) \to r$.

Proof. Demostremos los dos puntos por separado:

1. Procedemos por reducción al absurdo. Supongamos que existen dos lhs's $l_1 \in L_f$ y $l_2 \in L_f$, tal que en la posición k, $l_1|_k \equiv x$ y $l_2|_k \equiv c(x_1, \ldots, x_n)$. Esto viola la restricción (4) de uniformidad, con lo que $\mathcal{R}$ no puede ser un programa uniforme.
2. Procedemos por casos y por reducción al absurdo en cada caso:
   - Consideremos el caso en el que aparecen constructores planos en algunas de las posiciones $k \in \{1, \ldots, n\}$ de las lhs's de las reglas que definen f. Si f está definido por una sola regla, el punto (2) se cumple trivialmente. Si hay más de dos reglas que definen f, tomemos dos lhs's $l_1 \in L_f$ y $l_2 \in L_f$ cualesquiera. Supongamos que no existe un argumento $k \in \{1, \ldots, n\}$ tal que $l_1|_k = c_1(x_1, \ldots, x_n)$ y $l_2|_k = c_2(y_1, \ldots, y_n)$ con $c_1 \neq c_2$. Entonces, teniendo en cuenta el resultado del punto anterior, l_1 y l_2 son identicas salvo renombramiento, σ, y por tanto unifican, siendo σ el unificador más general. Esto viola la condición (5) de no-ambigüedad.
   - En el caso de que no aparezcan constructores planos en ninguna de las posiciones $k \in \{1, \ldots, n\}$ de las lhs's de las reglas que definen f, procedemos de igual forma. Concluimos que f está definida por una sóla regla $f(x_1, \ldots, x_n) \to r$.

El Lema 3 pone de manifiesto la estructura de los programas uniformes. En un programa uniforme, las funciones f/n están definidas por una o más reglas cuyas lhs's son de la forma

$$f(\ldots, c_{k_1}(x_1, \ldots, x_{m_{k_1}}), \ldots, c_{k_p}(y_1, \ldots, y_{m_{k_p}}), \ldots),$$

siendo $\{k_1, \ldots, k_p\} \subseteq \{1, \ldots, n\}$ posiciones fijas en las que aparecen constructores planos mientras en el resto de las posiciones aparecen variables, o bien por una sóla regla de la forma $f(x_1, \ldots, x_n) \to r$. En el primero de los casos, cuando hay más de una regla definiendo el símbolo f, para cada par de reglas, al menos en uno de los argumentos no variables debe aparecer un término constructor plano con un símbolo constructor diferente. Por razones que quedaran claras más adelante, denominaremos a las posiciones fijas $\{k_1, \ldots, k_p\}$ en las que aparecen constructores planos *posiciones inductivas* de f.

Example 4. El programa

$$f(X, a, b, c) \to 1$$
$$f(X, b, b, c) \to 2$$
$$f(X, b, b, b) \to 3$$

donde a, b y c se consideran símbolos constructores, es un ejemplo de programa uniforme.

La posibilidad de realizar evaluación perezosa es una característica importante de los lenguajes lógico-funcionales. Sin embargo, como se muestra en [24, 39], es difícil combinar una estrategia de evaluación perezosa con el uso de variables lógicas y una implementación que utilize una técnica de busqueda en profundidad con *vuelta atrás* (*backtracking*). Pueden destacarse los siguientes problemas relacionados con el uso de la evaluación perezosa:

- Por un lado, la computación de un término t puede no terminar si se demanda un subtérmino $t|_p$, en una posición interna p de t, para el que existen infinitos resultados y ninguno de ellos contribuye al cómputo (proporcionando la posibilidad de unificar una regla del programa con el término) más externo t.
- Por otra parte, cuando se retrasa la evaluación de un término, éste puede evaluarse varias veces, en lugar de una sola vez, como ocurriría si se empleara una estrategia de evaluación impaciente (*innermost*).
- Finalmente, la existencia de puntos de vuelta atrás "adicionales", asociados a la existencia de diversos redexes, dificulta una implementación eficiente (y posiblemente completa) de la estrategia de *narrowing* perezoso, que debe gestionar esos puntos de vuelta atrás "adicionales", junto con los habituales (en programación lógica) asociados a la elección de las diferentes reglas.

El primero de los problemas puede solucionarse mediante el empleo de técnicas de evaluación mixta, combinando *narrowing* perezoso con *narrowing* impaciente para el cómputo de los subtérminos demandados. El segundo puede resolverse utilizando compartición (*sharing*) de variables o alguna técnica de reducción basada en grafos. Los programas uniformes fueron introducidos por vez primera en [29] para paliar el último de los inconvenientes mencionados.

El siguiente ejemplo aclara y profundiza en este último punto.

Example 5. [29] Dado el programa (no uniforme)

$$\begin{aligned}\mathcal{R} = \{\ & R_1 : f(0,0) && \to 0\\ & R_2 : f(s(X),0) && \to s(0)\\ & R_3 : f(X,s(s(Y))) && \to s(s(0))\}\end{aligned}$$

y el término $t \equiv f(f(X,Y),Z)$, puede construirse el árbol de búsqueda de la Figura 2, donde se han subrayado los distintos redexes y se han etiquetado las ramas con las reglas de $\mathcal{R}$ empleadas en cada paso de *narrowing* perezoso. Puede apreciarse que, cuando se evalua el término t, las reglas R_1 y R_2 de $\mathcal{R}$ demandan el primer argumento, ya que $f(X,Y)$ no está suficientemente evaluado. Por lo tanto, se postpone la aplicación de las reglas R_1 y R_2 hasta haberse evaluado el argumento $f(X,Y)$. Sin embargo, la regla R_3 si puede ser aplicada directamente sobre t. Al evaluar el término demandado $f(X,Y)$, nuevamente debe considerarse la aplicación de todas las reglas.

Este ejemplo muestra la aparición de diversos redexes demandados por distintas reglas, lo que obliga a considerar, en una implementación secuencial de la estrategia de *narrowing* perezoso, puntos de elección de vuelta atrás asociados a

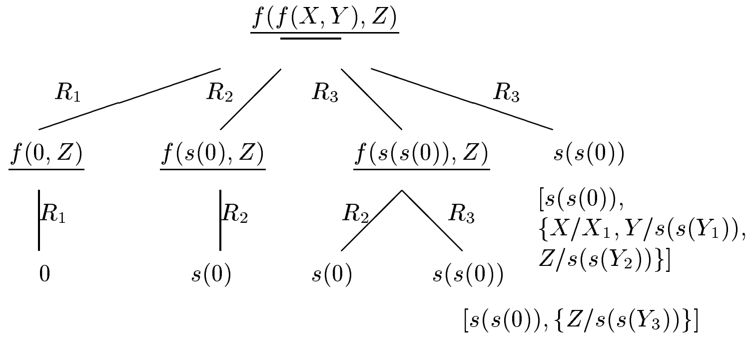

Fig. 2. Arbol de búsqueda para el término $f(f(X,Y),Z)$ utilizando LN. Se muestran las salidas redundantes.

las reglas así como asociados a la existencia de diferentes redexes (para los que no existe algo análogo en el contexto de la programación lógica y que provocan un nuevo intento de unificación con las diferentes reglas del programa). Esta singularidad plantea problemas de eficiencia, no solamente porque ambos tipos de puntos de elección deben ser gestionados en una implementación secuencial de la estrategia de *narrowing* perezoso, sino porque también pueden aparecer cómputos redundantes. Se debe notar que, en el ejemplo anterior, las dos ramas más a la derecha computan el mismo resultado con la misma respuesta. La principal dificultad radica en el hecho de que pueda darse un paso de *narrowing* perezoso sobre una posición de t (Λ) con una regla (R_3), mientras que no es posible dar un paso de *narrowing* perezoso con las otras reglas sobre dicha posición. Los programas uniformes se introdujeron para evitar este tipo de situaciones, en el que diferentes reglas demandan diferentes redexes de un mismo término, lo que conduce a la existencia de puntos de vuelta atrás asociados a diferentes redexes que deben ser explotados. La idea es remplazar los puntos de vuelta atrás debidos a los diferentes redexes por puntos de vuelta atrás debidos a la elección de las diferentes reglas del programa.

En [29] se presenta un algoritmo que transforma cualquier programa (en el sentido definido en el Apartado 3.1) en un programa uniforme. Denotaremos este algoritmo mediante el símbolo $\mathcal{U}_B$. La transformación $\mathcal{U}_B$ es idempotente, esto es $\mathcal{U}_B(\mathcal{U}_B(\mathcal{R})) = \mathcal{U}_B(\mathcal{R})$. El siguiente ejemplo muestra las ventajas de transformar un programa en un programa uniforme.

Example 6. La implementación del lenguaje lógico-funcional BABEL, propuesta en [29], transforma el programa $\mathcal{R}$ del Ejemplo 5 en el programa uniforme:

$$\begin{array}{rll} \mathcal{U}_B(\mathcal{R}) = \{ & f(X,0) & \to h(X) \\ & f(X,s(Y)) & \to g(Y) \\ & g(s(Y)) & \to s(s(0)) \\ & h(0) & \to 0 \\ & h(s(X)) & \to s(0)\} \end{array}$$

donde g y h son símbolos de función nuevos, que no aparecen en el programa original. Ahora, en la evaluación del término $f(f(X,Y),Z)$, todas las reglas del programa que definen f se aplican sobre la posición Λ. En general, para un término $f(s_1,s_2)$, siendo s_2 un término encabezado por un símbolo de función, todas las reglas del programa que definen f demandan el segundo argumento s_2.

Para el programa del Ejemplo 6, los puntos de vuelta atrás debidos a diferentes redexes han sido eliminados. También puede comprobarse una mejora de la eficiendia debido a la desaparición de salidas redundantes. El programa del Ejemplo 5 computa las salidas $[s(s(0)),\{X/X_1,Y/s(s(Y_1)),Z/s(s(Y_2))\}]$ y $[s(s(0)),\{Z/s(s(Y_3))\}]$, donde cláramente la última respuesta es más general que la primera, sin embargo el programa del Ejemplo 6 sólo computa la salida $[s(s(0)),\{Z/s(s(Y_3))\}]$.

A pesar de las ventajas de los programas uniformes, no siempre se consigue con ellos la eliminación de los puntos de vuelta atrás asociados a diferentes redexes demandados, en un mismo término, por diferentes reglas del programa. Esto se pone de manifiesto más adelante, en el Ejemplo 12.

Por otra parte, la transformación $\mathcal{U}_B$ presenta deficiencias, algunas de las cuales ya fueron mencionadas en [29], que detallamos a continuación. La transformación $\mathcal{U}_B$ consta de dos fases: i) *aplanamiento*, que produce reglas cuyas lhs's son patrones constructores planos; ii) *obtención de uniformidad*, que asegura que se cumpla la correspondiente restricción de uniformidad sin violar el resto de las restricciones que debe respetar un programa uniforme. Sin embargo, $\mathcal{U}_B$ no siempre logra preservar la ortogonalidad (débil) del programa original, dando lugar a programas que violan la restricción de no-ambigüedad. Por este motivo en [29] se introduce una tercera fase consistente en "*fusionar*" adecuadamente las reglas que poseen la misma lhs salvo, renombramiento de variables, y que, por lo tanto, incumplen la restricción de no-ambigüedad (débil) que debe respetar todo programa uniforme. Esta tercera fase ha sido empleada para obtener el programa del Ejemplo 6, a partir de un programa transformado intermedio en el que se había perdido la ortogonalidad (débil) del programa original y que, por lo tanto, no era uniforme. Desgraciadamente, no siempre es posible eliminar el problema comentado, a través de la fusión de reglas realizada en la fase final de la transformación sin introducir otros problemas. Los dos ejemplos siguientes ponen de manifiesto este hecho.

Example 7. Sea el programa ortogonal y CB:

$$\begin{array}{ll} \mathcal{R} = \{ & f(a,b,X) \to 1 \\ & f(c,X,c) \to 2 \\ & f(X,a,b) \to 3\}. \end{array}$$

Por ser ortogonal, $\mathcal{R}$ es lineal por la izquierda y no-ambigüo. Asimismo, este programa cumple la restricción de patrones constructores planos. Por lo tanto la primera fase de la transformación $\mathcal{U}_B$ devuelve el programa original $\mathcal{R}$. Si aplicamos la segunda fase de la transformación $\mathcal{U}_B$ a $\mathcal{R}$, obtenemos el programa

$$\begin{array}{lll} \{ & f(Y,Z,X) & \to f_{iv}(Y,Z,X) \\ & f_{iv}(Y,b,X) & \to f_i(Y,b,X) \\ & f_i(a,b,X) & \to 1 \\ & f(Y,X,Z) & \to f_v(Y,X,Z) \\ & f_v(Y,X,c) & \to f_{ii}(Y,X,c) \\ & f_{ii}(c,X,c) & \to 2 \\ & f(X,Y,Z) & \to f_{vi}(X,Y,Z) \\ & f_{vi}(X,Y,c) & \to f_{iii}(X,Y,c) \\ & f_{iii}(X,a,b) & \to 3\}, \end{array}$$

donde los símbolos de función $f_i, \ldots, f_{vi}$ son símbolos nuevos que no aparecen en el programa original. Podemos observar que, ahora, el programa obtenido cumple la restricción de uniformidad pero incumple la restricción de no-ambigüedad, ya que las lhs's de las reglas primera, cuarta y septima unifican. Por lo tanto, el programa no es uniforme. Podemos intentar recuperar la restricción de no-ambigüedad del programa original $\mathcal{R}$ mediante la fusión adecuada de las reglas que plantean el conflicto. Tras la fusión de dichas reglas se obtiene el programa

$$\begin{array}{lll} \mathcal{R}_1 = \{ & f(X,Y,Z) & \to f_1(X,Y,Z) \\ & f_1(Y,b,X) & \to f_i(Y,b,X) \\ & f_i(a,b,X) & \to 1 \\ & f_1(Y,X,c) & \to f_{ii}(Y,X,c) \\ & f_{ii}(c,X,c) & \to 2 \\ & f_1(X,Y,c) & \to f_{iii}(X,Y,c) \\ & f_{iii}(X,a,b) & \to 3\}, \end{array}$$

que tampoco es uniforme, ya que la función definida f_1, introducida para fundir las reglas en conflicto y eliminar el problema de la no-ambigüedad, incumple la restricción de uniformidad. El resultado es un programa que debe ser sometido a un nuevo proceso de transformación. Aplicando un proceso de transformación como el anteriormente descrito al programa $\mathcal{R}_1$, obtenemos el programa

$$\begin{array}{lll} \mathcal{R}_2 = \{ & f(X,Y,Z) & \to f_1(X,Y,Z) \\ & f_1(X,Y,Z) & \to f_2(X,Y,Z) \\ & f_2(Y,b,X) & \to f_i(Y,b,X) \\ & f_i(a,b,X) & \to 1 \\ & f_2(Y,X,c) & \to f_{ii}(Y,X,c) \\ & f_{ii}(c,X,c) & \to 2 \\ & f_2(X,Y,c) & \to f_{iii}(X,Y,c) \\ & f_{iii}(X,a,b) & \to 3\}, \end{array}$$

Podemos apreciar que el programa $\mathcal{R}_2$ simplemente reproduce el programa $\mathcal{R}_1$ (pero ahora con la nueva función definida f_2 y la regla redundante $f_1(X,Y,Z) \to f_2(X,Y,Z)$). Esta claro que después de transformar este programa y fusionar sus reglas n veces obtendriamos el programa

$$
\begin{array}{lll}
\mathcal{R}_n = \{ & f(X,Y,Z) & \to f_1(X,Y,Z) \\
& f_1(X,Y,Z) & \to f_2(X,Y,Z) \\
& . & \\
& . & \\
& . & \\
& f_{n-1}(X,Y,Z) & \to f_n(X,Y,Z) \\
& f_n(Y,b,X) & \to f_i(Y,b,X) \\
& f_i(a,b,X) & \to 1 \\
& f_n(Y,X,c) & \to f_{ii}(Y,X,c) \\
& f_{ii}(c,X,c) & \to 2 \\
& f_n(X,Y,c) & \to f_{iii}(X,Y,c) \\
& f_{iii}(X,a,b) & \to 3\},
\end{array}
$$

que sigue sin ser uniforme.

Debilitar la restricción de no-ambigüedad, permitiendo el uso de programas débilmente no-ambigüos, no mejora el comportamiento de la transformación $\mathcal{U}_B$, como muestra el siguiente ejemplo.

Example 8. Dado el programa débilmente ortogonal y CB:

$$
\begin{array}{ll}
\{ & f(a,Y) \to c \\
& f(X,b) \to c\}.
\end{array}
$$

Si aplicamos la transformación $\mathcal{U}_B$ en sus dos primeras fases, obtenemos el programa

$$
\begin{array}{lll}
\{ & f(X,Y) & \to f'(X,Y) \\
& f(X,Y) & \to f''(X,Y) \\
& f'(a,Y) & \to c \\
& f''(X,b) & \to c\},
\end{array}
$$

donde f' y f'' son símbolos de función nuevos, que no aparecen en el programa original. Podemos observar que el programa obtenido incumple la restricción de no-ambigüedad débil, ya que las lhs's de las dos primeras reglas son iguales, y por lo tanto no es uniforme. Si ahora intentamos fundir las reglas adecuadamente obtenemos el programa

$$
\begin{array}{ll}
\{ & f(X,Y) \to f_1(X,Y) \\
& f_1(a,Y) \to c \\
& f_1(X,b) \to c\},
\end{array}
$$

que tampoco es uniforme, ya que la función definida f_1, introducida para fundir las dos primeras reglas y eliminar el problema de la no-ambigüedad débil, incumple la restricción de uniformidad. El resultado es un programa que debe ser sometido a un nuevo proceso de transformación. Si nos fijamos bien, el programa obtenido simplemente reproduce el programa original (pero ahora con la nueva función definida f_1). Esta claro que después de transformar este programa y fusionar sus reglas n veces obtendriamos el programa

$$
\begin{array}{ll}
\{\ f(X,Y) & \to f_1(X,Y) \\
\ \ f_1(X,Y) & \to f_2(X,Y) \\
\ \ . & \\
\ \ . & \\
\ \ . & \\
\ \ f_{n-1}(X,Y) & \to f_n(X,Y) \\
\ \ f_n(a,Y) & \to c \\
\ \ f_n(X,b) & \to c\},
\end{array}
$$

que sigue sin ser uniforme.

La característica común de los programas del Ejemplo 7 y del Ejemplo 8 es que no son inductivamente secuenciales, contrariamente a lo que sucedia con el programa del Ejemplo 5. A la vista de estos ejemplos podemos concluir que la transformación $\mathcal{U}_B$, compuesta de las tres fases anteriomente mencionadas, no termina obteniendo un programa uniforme cuando se parte, en general, de un programa (débilmente) ortogonal y CB. Conjeturamos que $\mathcal{U}_B$ sólamente es correcta cuando se usa sobre programas inductivamente secuenciales.

Recientemente, Zartmann ha introducido una nueva clase de programas uniformes, obtenidos mediante una transformación, definida en [45], a partir de programas inductivamentes secuenciales [12, 8]. En lo que sigue denotaremos la transformación de Zartmann mediante el símbolo $\mathcal{U}_Z$. La transformación $\mathcal{U}_Z$ es idempotente. Esta transformación conduce a programas que llevan "precompilada" la información almacenada en los árboles definicionales, en el sentido de que para ellos *narrowing* perezoso y *narrowing* necesario coinciden (como mostraremos más adelante). Los programas uniformes definidos por Zartmann se caracterizan porque en las lhs's de las reglas que definen f puede haber como máximo un argumento $k \in \{1, \ldots, n\}$ en el que aparece un constructor plano, siendo el resto de los argumentos $i \neq k$, argumentos en los que aparecen variables. Atendiendo a esta característica, denominaremos a los programas uniformes definidos por Zartmann, *programas uniformes simples*, ya que sólo poseen una posición inductiva. La clase de programas uniformes simples es un subconjunto de los programas uniformes, como se ilustra en la Figura 4.

Example 9. El programa

$$
\begin{array}{l}
f(X,a,Y,Z) \to 1 \\
f(X,b,Y,Z) \to 2 \\
f(X,c,Y,Z) \to 3
\end{array}
$$

donde a, b y c se consideran símbolos constructores, es un ejemplo de programa uniforme simple.

Los programas uniformes simples son inductivamente secuenciales, i.e., cada una de las funciones definidas en el programa tienen asociado un árbol definicional.

Example 10. La función f definida en el programa del Ejemplo 9 tiene asociada el árbol definicional de la Figura 3.

Notad que este árbol es único y sólo posee un nivel, en correspondencia con la única posición inductiva que aparece en la función f definida en el programa del Ejemplo 9. De esta forma, los subárboles que cuelgan de la raíz del árbol definicional son hojas. Este hecho es generalizable al resto de los árboles definicionales asociados a funciones definidas en programas uniformes simples.

$$f(X_1, \boxed{X_2}, X_3, X_4)$$

$$f(X_1, a, X_3, X_4) \to 1 \qquad f(X_1, b, X_3, X_4) \to 2 \qquad f(X_1, c, X_3, X_4) \to 3$$

Fig. 3. Arbol definicional para la función "f" definida en el Ejemplo 9.

La siguiente proposición pone de manifiesto que los programas uniformes también son inductivamente secuenciales. La Figura 4 ilustra la relación existente entre éstos y otras clases de programas ortogonales.

Proposition 4. *Todo programa uniforme es inductivamente secuencial.*

Proof. Basta probar que a cada una de las funciones f definidas en un programa uniforme $\mathcal{R}$ le corresponde un árbol definicional. Para realizar la prueba, definiremos un procedimiento que permita la construccción de un árbol cuya raíz sea el término $f(x_1, \ldots, x_n)$ y cuyas hojas sean los términos del conjunto $L_f = \{l \mid l \to r \in \mathcal{R} \wedge \mathcal{H}ead(l) \equiv f\}$ de las lhs's de las reglas que definen la función f en $\mathcal{R}$. Finalmente, comprobaremos que el árbol construido cumple las restricciones de la Definición 12 de árbol definicional.

Sea $I = \{k \mid l \in L_f \wedge l|_k = c_k(x_1, \ldots, x_{m_k})\}$ el conjunto de posiciones inductivas de f, que por el Lema 3 está fijado por igual para todas las reglas de $\mathcal{R}$ que definen f. Nuestro procedimiento constructivo es el siguiente:

Algoritmo 1
Entrada: *El conjunto L_f de las lhs's que definen f.*
El conjunto I de posiciones en las que aparece un constructor plano.
Salida: *Un conjunto $\mathcal{P}$ de patrones lineales.*
Initialización: $\mathcal{P} = \{\pi_0\}$,
siendo $\pi_0 \equiv f(x_1, \ldots, x_n)$ *y* $x_1, \ldots, x_n$ *variables nuevas*
Mientras $I \neq \varnothing$ **haced**

1) Seleccionar un $k \in I$ arbitrario; $I := I \setminus \{k\}$;
2) $H = \{\pi \mid \pi$ es una hoja de $\mathcal{P}\}$;
3) **Repetir**
3.1) Seleccionar una hoja $\pi \in H$; $H := H \setminus \{\pi\}$;
3.2) $L_\pi = \{l \mid l \in L \wedge \pi \leq l\}$;
3.3) $C = \{c_i(x_1, \ldots, x_{m_i}) \mid (\forall i)(l_i \in L_\pi \wedge \mathcal{H}ead(l_i|_k) = c_i\}$, donde las variables $x_1, \ldots, x_{m_i}$ son nuevas
3.4) Para cada $c_i(x_1, \ldots, x_{m_i}) \in C$,
- *formar una nueva hoja $\pi_i = \pi[c_i(x_1, \ldots, x_{m_i})]_k$,*
- *$\mathcal{P} := \mathcal{P} \cup \{\pi_i\}$;*

Hasta que $H = \varnothing$
FinMientras
Devolver $\mathcal{P}$

Hacemos notar que el algoritmo anterior trata todos los casos posibles relativos a la definición de una función, f, por las reglas del programa. En particular, cuando la función f está definida por una sola regla $f(x_1, \ldots, x_n) \to r$, entonces $L_f = \{f(x_1, \ldots, x_n)\}$ e $I = \varnothing$, obteniendose de forma inmediata el árbol $\mathcal{P} = \{f(x_1, \ldots, x_n)\}$ constituido por un sólo nodo.

El conjunto de patrones lineales $\mathcal{P}$ construido por el algoritmo que acabamos de describir, está ordenado por el orden $<$ (de generalidad relativa estricta) y cumple las propiedades que caracterizan un árbol definicional:

- *Propiedad de la raíz*: Efectivamente, $pattern(\mathcal{P}) = f(x_1, \ldots, x_n)$ es un elemento mínimo, ya que $f(x_1, \ldots, x_n) < \pi$ para todo $\pi \in \mathcal{P}$.
- *Propiedad de las hojas*: Por construcción, las hojas de $\mathcal{P}$ son los elementos del conjunto L_f. Estos se han obtenido a partir del elemento minimal $f(x_1, \ldots, x_n)$, instanciando cada posición inductiva, $k \in I$, con los correspondientes constructores planos (pasos 3.2 a 3.4). Por consiguiente, los elementos de L_f son instancias de sus respectivos ancestros en el árbol construido y por consiguiente los elementos maximales de $\mathcal{P}$
- *Propiedad de los padres*: Por construcción, dado un nodo $\pi = f(\ldots, x_k, \ldots)$ le corresponden diferentes hijos $\pi_i = f(\ldots, c_i(x_1, \ldots, x_{m_i}), \ldots)$. Por consiguiente, a cada nodo $\pi_i \in \mathcal{P}$ distinto de $pattern(\mathcal{P})$ le corresponde un único padre $\pi < \pi_i$, no pudiendo existir otro padre π' tal que $\pi < \pi' < \pi_i$, ya que la variable x_k es sustituida por un constructor plano $c_i(x_1, \ldots, x_{m_i})$, cuando formamos π_i.
- *Propiedad inductiva*: Dado un nodo $\pi \in \mathcal{P} \setminus L_f$ las posiciones inductivas son elementos de I. El paso 3.4 del algoritmo que construye $\mathcal{P}$ nos asegura que los hijos π_i de π cumplen la propiedad inductiva.

Observad que la Proposición 4 asegura que, dado un programa uniforme, para cualquier función definida en el programa es posible construir su árbol definicional, haciendo uso del Aloritmo 1, partiendo de una posición inductiva arbitraria. Esto no sucede para los programas inductivamente secuenciales no uniformes (e.g., es imposible construir el árbol definicional de la función "$\leq$" definida en el Ejemplo 1, si seleccionamos como posición inductiva el segundo argumento).

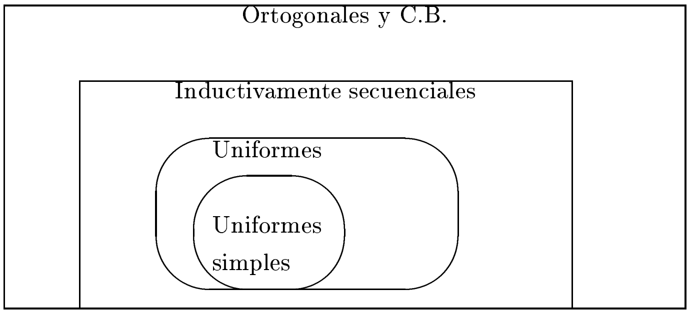

Fig. 4. Una clasificación de los programas

Example 11. El Algoritmo 1 construye el árbol definicional de la Figura 5 para la función f definida en el programa del Ejemplo 4. La figura se ha obtenido concretando la regla de selección de posiciones inductivas, de modo que seleccionamos la posición más a la izquierda de entre las del conjunto I.

Notad que el número de niveles y posiciones inductivas del árbol definicional es igual al número de posiciones inductivas que aparecen en las lhs's de las reglas que definen f. Por construcción, este es un hecho generalizable a todos los árboles definicionales asociados a programas uniformes. Ahora debe de quedar clara la razón por la que elejimos el nombre de "posición inductiva" para las posiciones, del patrón de una regla del programa, en las que aparecen términos constructores planos. Como hemos visto estas posiciones se corresponden con las posiciones inductivas del árbol definicional.

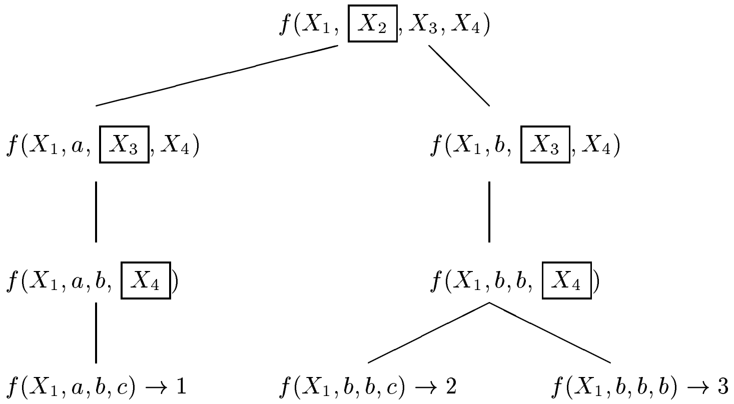

Fig. 5. Arbol definicional para la función "f" definida en el Ejemplo 4.

En lo que sigue, diremos que un árbol definicional se construye de la *forma obvia* cuando en el Algotitmo 1 se concreta la regla de selección de posiciones inductivas, de modo que seleccionamos la posición más a la izquierda de entre las del conjunto I (i.e., seleccionamos de izquierda a derecha los argumentos que son posiciones inductivas de las funciones definidas por el programa).

5 Estrategias de Evaluación Perezosa en Programas Uniformes.

En este apartado estudiamos la equivalencia entre las estrategias de *narrowing* perezoso y *narrowing* necesario para programas uniformes. Como resultado de este estudio definimos un refinamiento de la estrategia de *narrowing* perezoso, que denominamos *narrowing perezoso uniforme*, para el que probamos su corrección y completitud. Comenzamos con una serie de resultados sobre programas uniformes simples que nos sirven de punto de partida.

5.1 Resultados para Programas Uniformes Simples.

Zartmann ha establecido la correspondencia entre las derivaciones que respetan una estrategia de *narrowing* necesario en un programa $\mathcal{R}$ inductivamente secuencial y las que respetan una estrategia perezosa en el correspondiente programa transformado $\mathcal{U}_Z(\mathcal{R})$.

Theorem 3. *[45] Sea $\mathcal{R}$ un programa inductivamente secuencial. Sea $\mathcal{U}_Z(\mathcal{R})$ un programa uniforme simple. Sea t un término encabezado por un símbolo de operación $f \in \mathcal{F}$. Existe una derivación de* narrowing *necesario $t \overset{\sigma}{\leadsto}_{NN}^{*} s$ en $\mathcal{R}$ a una hnf s si y sólo si existe una derivación de* narrowing *perezoso $t \overset{\sigma}{\leadsto}_{LN}^{*} s$ en $\mathcal{U}_Z(\mathcal{R})$.*

El resultado anterior puede entenderse diciendo que la transformación $\mathcal{U}_Z$ es correcta y completa, en el sentido de que el programa uniforme simple transformado, $\mathcal{U}_Z(\mathcal{R})$, cuando se ejecuta empleando *narrowing* perezoso, tiene idéntica semántica que la obtenida para el programa original $\mathcal{R}$, cuando se ejecuta ejecuta empleando la estrategia de *narrowing* necesario.

El siguiente corolario establece la relación existente entre derivaciones de *narrowing* necesario y *narrowing* perezoso en un programa uniforme simple.

Corollary 2. *Sea $\mathcal{R}$ un programa uniforme simple. Sea t un término encabezado por un símbolo de operación $f \in \mathcal{F}$ y s un término en hnf. Existe una derivación de* narrowing *necesario $t \overset{\sigma}{\leadsto}_{NN}^{*} s$ en $\mathcal{R}$ si y sólo si existe una derivación de* narrowing *perezoso $t \overset{\sigma}{\leadsto}_{LN}^{*} s$ en $\mathcal{R}$.*

Proof. Inmediata, por el Teorema 3 y el hecho de que $\mathcal{U}_Z(\mathcal{R}) = \mathcal{R}$.

En el caso de programas uniformes, esta correspondencia en general no es biunívoca, como muestra el siguiente ejemplo.

Example 12. Sea el programa uniforme

$$\begin{aligned}
R_1 &: f(X,a,b) \to 1\\
R_2 &: f(X,a,c) \to 2\\
R_3 &: g(a) \quad\;\;\; \to a\\
R_4 &: g(b) \quad\;\;\; \to b
\end{aligned}$$

Para el término $t \equiv f(g(X), g(Y), g(Z))$ existen las siguientes derivaciones empleando la estrategia de *narrowing* perezoso:

$$\begin{aligned}
f(g(X), g(Y), g(Z)) &\overset{[2,R_3,\{Y/a\}]}{\leadsto_{LN}} f(g(X), a, g(Z))\\
&\overset{[3,R_4,\{Z/b\}]}{\leadsto_{LN}} f(g(X), a, b)\\
&\overset{[\Lambda,R_1,id]}{\leadsto_{LN}} 1,
\end{aligned}$$

y

$$\begin{aligned}
f(g(X), g(Y), g(Z)) &\overset{[3,R_4,\{Z/b\}]}{\leadsto_{LN}} f(g(X), g(Y), b)\\
&\overset{[2,R_3,\{Y/a\}]}{\leadsto_{LN}} f(g(X), a, b)\\
&\overset{[\Lambda,R_1,id]}{\leadsto_{LN}} 1,
\end{aligned}$$

mientras que sólamente es posible una derivación empleando *narrowing* necesario (cuando empleamos el árbol definicional para la función "f" que aparece en la Figura 6):

$$\begin{aligned}
f(g(X), g(Y), g(Z)) &\overset{[2,R_3,\{Y/a\}]}{\leadsto_{NN}} f(g(X), a, g(Z))\\
&\overset{[3,R_4,\{Z/b\}]}{\leadsto_{NN}} f(g(X), a, b)\\
&\overset{[\Lambda,R_1,id]}{\leadsto_{NN}} 1.
\end{aligned}$$

Así pues, la estrategia de *narrowing* perezoso, incluso cuando se aplica a programas uniformes, sigue generando derivaciones redundantes (si bien, ambas computan identicos resultados con las correspondientes respuestas y explotando posiciones necesarias).

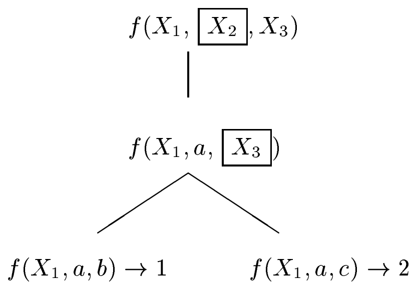

Fig. 6. Arbol definicional para la función "f"

El Ejemplo 12 pone de manifiesto una cierta ineficiencia (ya comentada) de la estrategia de *narrowing* perezoso frente a la de *narrowing* necesario. La causa de ésta radica en el hecho de que, al emplear la estrategia de *narrowing* perezoso sobre programas uniformes, varias reglas que definen una misma función pueden demandar la evaluación de posiciones diferentes de un término. Este era el caso del término $t \equiv f(g(X), g(Y), g(Z))$, para el que se demandan las posiciones 2 y 3, asociadas con las correspondientes posiciones inductivas de las reglas que definen f. En los programas uniformes simples, dado que en cada lhs de una regla sólo puede aparecer una posición inductiva (i.e., un argumento con un término constructor plano y en la misma posición para todas las reglas que definen una determinada función), la estrategia de *narrowing* perezoso sólo demandará una posición de un término t. Esta posición será una posición necesaria para la evaluación del término t, de lo cual se deriva que los pasos de *narrowing* perezoso y de *narrowing* necesario deben coincidir en programas uniformes simples. Este resultado se presenta formalmente en la siguiente proposición, que establece la completa equivalencia entre las estrategias de *narrowing* necesario y de *narrowing* perezoso para programas uniformes simples.

Proposition 5. *Sea $\mathcal{R}$ un programa uniforme simple. Sea t un término encabezado por un símbolo de operación $f \in \mathcal{F}$ y $\mathcal{P}$ el árbol definicional de f. Entonces $\lambda(t, \mathcal{P}) = \lambda_{lazy}(t)$.*

Proof. Por inducción sobre la complejidad del término $t \equiv f(t_1, \ldots, t_n)$, medida por el número de símbolos de función y constructores que aparecen en t. Las sustituciones se consideran módulo renombramiento de variables y restringidas a las variables del término t.

1. Caso base ($n = 1$): En este caso, en t no aparecen otros símbolos de función o constructores que el símbolo f que lo encabeza. Siguiendo la Definición 14 de la estrategia de *narrowing* necesario, λ, distinguimos las siguientes posibilidades.
   (a) $pattern(\mathcal{P}) = \pi$ es una hoja. En este caso, la función f está definida por una única regla $R \equiv f(x_1, \ldots, x_n) \to r$ y es inmediato comprobar que $\lambda(t, \mathcal{P}) = \lambda_{lazy}(t) = \{\langle \Lambda, R, id \rangle\}$.
   (b) $pattern(\mathcal{P}) = \pi$ es una rama. Dado que todos los argumentos del término t son variables, necesariamente, para la única posición inductiva o de $\mathcal{P}$ se cumple que $t|_o \equiv x$.
   Por ser $\mathcal{R}$ uniforme simple, los subárboles $\mathcal{P}_k$ que cuelgan de la posición inductiva o son tales que $pattern(\mathcal{P}_k) = l_k$ es una hoja (i.e., la lhs de una de las m reglas $R_k \equiv l_k \to r_k$ que definen f) y por lo tanto $\lambda(\tau_k(t), \mathcal{P}_k) = \{(\Lambda, R_k, id)\}$, donde $\tau_k = \{x/c_k(x_1, \ldots, x_n)\}$. Aplicando la definición de estrategia de *narrowing* necesario, λ, es inmediato obtener que $\lambda(t, \mathcal{P}) = \bigcup_{k=1}^{m} \{\langle \Lambda, R_k, \tau_k \rangle\}$.
   Por otro lado, $\lambda_{lazy} = \bigcup_{k=1}^{m} \lambda\_(t, \Lambda, k)$. Por ser $\mathcal{R}$ uniforme simple, las reglas que definen f tienen todos sus argumentos variables, salvo el situado en la posición inductiva o, que presenta el constructor plano $c_k(x_1, \ldots, x_n)$. Por consiguiente, el problema de unificación lineal

planteado es tal que $\mathsf{LU}(\langle l_k, t\rangle) = (\text{Succ}, \tau_k)$, módulo renombramiento de variables y si nos restringimos a las variables del término t. De esta forma, tenemos que $\lambda\_(t, \Lambda, k) = \{\langle \Lambda, R_k, \tau_k\rangle\}$ y concluimos que $\lambda_{lazy}(t) = \lambda(t, \mathcal{P})$.

2. Caso inductivo ($n > 1$): Siguiendo la definición de la estrategia de *narrowing* necesario, λ, distinguimos las siguientes posibilidades.
   (a) $pattern(\mathcal{P}) = \pi$ es una hoja. En este caso, la función f está definida por una única regla $R \equiv f(x_1, \ldots, x_n) \to r$ y $\pi = f(x_1, \ldots, x_n)$.
   Por definición, $\lambda(t, \mathcal{P}) = \{\langle \Lambda, R, id\rangle\}$ y $\pi \le t$. Por lo tanto, existe una substitución σ tal que $t = \sigma(\pi)$. Entonces, el problema de unificación lineal planteado entre los términos π y t debe de tener éxito, de forma que $\mathsf{LU}(\langle \pi, t\rangle) = (\text{Succ}, id)$, si la substitución unificadora más general σ se restringe a las variables del término t y se toma módulo renombramiento de variables. Así pues, $\lambda_{lazy}(t) = \{\langle \Lambda, R, id\rangle\}$.
   (b) $pattern(\mathcal{P}) = \pi$ es una rama. Ahora distinguimos entre las siguientes posibilidades, según que para la posición inductiva o de $\mathcal{P}$ aparezca en t una variable, un término encabezado por un constructor o un término encabezado por un símbolo de función:
      i. $t|_o \equiv x$
      En este caso, $\langle p, R_k, \sigma \circ \tau_k\rangle \in \lambda(t, \mathcal{P})$, donde $R_k \equiv l_k \to r_k$ es una de las m reglas que definen f, si $\langle p, R_k, \sigma\rangle \in \lambda(\tau_k(t), \mathcal{P}_k)$, donde $\tau_k = \{x/c_k(x_1, \ldots, x_{n_k})\}$. Dado que no puede aplicarse la hipótesis de inducción, ya que en el cómputo de $\lambda(\tau_k(t), \mathcal{P}_k)$ la complejidad del término $\tau_k(t)$ ha aumentado, procedemos de manera análoga a la del caso base (b). Concluimos que $\lambda_{lazy}(t) = \lambda(t, \mathcal{P}) = \bigcup_{k=1}^{m}\{\langle \Lambda, R_k, \tau_k\rangle\}$.
      ii. $t|_o \equiv c(t_1, \ldots, t_n)$
      Razonamos de manera similar al caso anterior, pero ahora t sólo puede unificar con una de las reglas R_k que definen f, aquella cuya lhs, l_k, cumple que $\mathcal{H}ead(l_k|_o) = c$. Así que, $\lambda(t, \mathcal{P}) = \lambda_{lazy}(t) = \{\langle \Lambda, R_k, id\rangle\}$.
      iii. $t|_o \equiv g(t_1, \ldots, t_n)$
      Entonces, $\langle o.p, R, \sigma\rangle \in \lambda(t, \mathcal{P})$ si $\langle p, R, \sigma\rangle \in \lambda(t|_o, \mathcal{P}')$, donde $\mathcal{P}'$ es el árbol definicional asociado a la operación g.
      Por ser $\mathcal{R}$ un programa uniforme simple, en todas las lhs's l_k de las reglas $R_k \equiv l_k \to r_k$ que definen f aparece un término constructor plano en la posición inductiva o, siendo el resto de los argumentos variables. Al evaluar $\lambda_{lazy}(t)$, obtenemos que $\mathsf{LU}(\langle l_k, t\rangle) = (\text{Demand}, \{o\})$, cualquiera que sea la lhs l_k. Además, por ser $\mathcal{R}$ uniforme, Λ no puede ser una posición perezosa de t, ya que entonces o no habría sido demandada. Por lo tanto, para computar $\lambda_{lazy}(t)$ es necesario computar previamente $\lambda_{lazy}(t|_o)$, cumpliéndose que si $\langle p, R_k, \sigma\rangle \in \lambda_{lazy}(t|_o)$ entonces $(o.p, R_k, \sigma) \in \lambda_{lazy}(t)$. Naturalmente, en cualquier circunstancia, si $\langle o.p, R_k, \sigma\rangle \in \lambda_{lazy}(t)$ entonces $\langle p, R_k, \sigma\rangle \in \lambda_{lazy}(t|_o)$. Por hipótesis de inducción$\lambda(t|_o, \mathcal{P}') = \lambda_{lazy}(t|_o)$, luego $\lambda(t, \mathcal{P}) = \lambda_{lazy}(t)$.

Haciendo uso de la Proposición 5, es inmediato establecer la siguiente relación entre derivaciones que respetan la estrategia de *narrowing* perezoso y las que respetan la estrategia de *narrowing* necesario.

Corollary 3. *Sea $\mathcal{R}$ un programa uniforme simple. Sea t un término encabezado por un símbolo de operación $f \in \mathcal{F}$ y s un término cualquiera. Existe una derivación por* narrowing *necesario $t \overset{\sigma}{\rightsquigarrow}_{NN}^{*} s$ en $\mathcal{R}$ si y sólo si existe una derivación por* narrowing *perezoso $t \overset{\sigma}{\rightsquigarrow}_{LN}^{*} s$ en $\mathcal{R}$. Estas derivaciones utilizan las mismas reglas sobre las mismas posiciones de t y sus sucesores.*

En este punto, es interesante observar que el resultado obtenido en el Corolario 3 es más fuerte que el establecido en el Corolario 2, que sólo asegura la equivalencia para derivaciones a una hnf.

5.2 Resultados para Programas Uniformes.

El problema cuando tratamos con programas uniformes es que la correspondencia entre derivaciones de *narrowing* necesario y *narrowing* perezoso no se puede obtener paso a paso, como vuelve a poner de manifiesto el siguiente ejemplo.

Example 13. Sea el programa uniforme

$$\begin{array}{l} R_1 : f(a,b) \to c \\ R_2 : g(c) \quad\ \to b \end{array}$$

Para el término $t \equiv f(x, g(y))$ y árboles definicionales de f y g construidos en la forma obvia, existe la siguiente derivación empleando *narrowing* necesario:

$$f(x, g(y)) \overset{[2,R_2,\{x/a,y/c\}]}{\rightsquigarrow_{NN}} f(a,b) \overset{[\Lambda,R_1,id]}{\rightsquigarrow_{NN}} c$$

mientras que la derivación empleando la estrategia de *narrowing* perezoso es:

$$f(x, g(y)) \overset{[2,R_2,\{y/c\}]}{\rightsquigarrow_{LN}} f(x,b) \overset{[\Lambda,R_1,\{x/a\}]}{\rightsquigarrow_{LN}} c$$

Sin embargo, existe todavía una relación interesante entre ambas estrategias, que formalizamos en la Proposición 6. Pero antes necesitamos un lema previo y precisar el concepto de *substitución adelantada*.

Como ya se ha comentado, una característica de la estrategia de *narrowing* necesario es que adelanta substituciones, con ello queremos decir que utiliza los árboles definicionales asociados a un programa, no sólamente como una guía para obtener posiciones demandadas de un término, sino que además guarda los enlaces necesarios para acceder a dicha posición. El resultado es que, aplicadas las substituciones adelantadas y una vez evaluado el término que ocupa la posición demandada, reducimos el indeterminismo que entraña toda derivación por *narrowing*. La estrategia de *narrowing* necesario utiliza el cómputo de las substituciones adelantadas para forzar pasos de *narrowing* sobre posiciones necesarias, evitando el cómputo de algunas salidas redundantes que la estrategia de *narrowing* perezoso computa. La utilización de substituciones adelantadas por parte de la estrategia de *narrowing* necesario es lo que esencialmente la diferencia de la estrategia *narrowing* perezoso. Por lo tanto, si deseamos establecer una relación entre ambas, es fundamental precisar el concepto de substitución adelantada.

El siguiente lema establece que cuando el paso de *narrowing* tiene lugar sobre la posición Λ, tanto la estragia de *narrowing* necesario como la de *narrowing* perezoso computan la misma respuesta con la misma regla. Intuitivamente, dado que la parte de substitución adelatada en un paso de *narrowing* necesario es aquella que no se computa en un paso de *narrowing* perezoso, podemos interpretar este resultado en el sentido de que la estrategia de *narrowing* necesario no adelanta substituciones cuando el cómputo se realiza sobre la posición Λ.

Lemma 4. *Sea $\mathcal{R}$ un programa uniforme y R una regla de $\mathcal{R}$. Sea t un término encabezado por un símbolo de operación $f \in \mathcal{F}$ y $\mathcal{P}$ el árbol definicional de f. Entonces, $\langle \Lambda, R, \sigma\rangle \in \lambda(t, \mathcal{P})$ si y sólo si $\langle \Lambda, R, \sigma\rangle \in \lambda_{lazy}(t)$.*

Proof. En esta prueba consideraremos las sustituciones módulo renombramiento de variables y restringidas a las variables del término t.

Supongamos que $\langle \Lambda, R, \sigma\rangle \in \lambda(t, \mathcal{P})$. Nótese que, debido a que $\langle \Lambda, R, \sigma\rangle \in \lambda(t, \mathcal{P})$, los argumentos de t sólo pueden ser variables o términos encabezados por un constructor. De otro modo, Λ no sería una posición necesaria de t. Siguiendo la Definición 14 de la estrategia de *narrowing* necesario λ, distinguimos las siguientes posibilidades.

1. $pattern(\mathcal{P}) = \pi$ es una hoja. En este caso, la función f está definida por una única regla $R \equiv f(x_1, \ldots, x_n) \to r$ y es inmediato comprobar que $\lambda(t, \mathcal{P}) = \lambda_{lazy}(t) = \{\langle \Lambda, R, id\rangle\}$.
2. $pattern(\mathcal{P}) = \pi$ es una rama.
   Por ser $\mathcal{R}$ un programa uniforme, existe una o más posiciones inductivas para cada árbol definicional $\mathcal{P}$. Dada una posición inductiva o_i de $\mathcal{P}$, tendremos dos posibilidades, o $t|_{o_i} \equiv x_{k_i}$ o $t|_{o_i} \equiv c_{k_i}(s_1, \ldots, s_{n_i})$, ya que los argumentos del término t son variables o términos encabezados por un constructor. El Lema 3 asegura que, para cada lhs de las reglas que definen f, en las posiciones inductivas o_i aparece un término constructor plano. Sea $R_k \equiv l_k \to r_k$ una de las reglas que definen f, para las que $l_k|_{o_i} \equiv c_{k_i}(x_1, \ldots, x_{n_i})$. Es fácil comprobar que

   $$\langle \Lambda, R_k, \tau_{k_q} \circ \ldots \circ \tau_{k_1}\rangle \in \lambda(t, \mathcal{P})$$

   donde q es el número de posiciones inductivas de $\mathcal{P}$ y $\tau_{k_i} = id$ o bien $\tau_{k_i} = \{x_{k_i}/c_{k_i}(x_1, \ldots, x_{n_i})\}$.
   Por otro lado, al computar $\lambda_{lazy}(t)$, el problema de unificación lineal planteado con la regla R_k es $\Gamma \equiv \langle l_k, t\rangle$. Una vez construida la correspondiente configuración inicial LU, (U_0, σ_0) podemos comprobar que existe la derivación LU:
   $(U_0, \sigma_0) \to^*_{\mathsf{LU}} (\{l_k|_{o_1} \downarrow_{o_1} t|_{o_1}, \ldots, l_k|_{o_q} \downarrow_{o_q} t|_{o_q}\}, id) \to^*_{\mathsf{LU}} (\varnothing, \tau_{k_q} \circ \ldots \circ \tau_{k_1})$

si sólo tenemos en consideración las variables del término t y que los elementos del tipo $c_{k_i}(x_1, \ldots, x_{n_i}) \downarrow_{o_i} c_{k_i}(s_1, \ldots, s_{n_i})$ se reducen a la substitución id. Entonces, $\mathsf{LU}(\langle l_k, t\rangle) = (\textsc{Succ}, \tau_{k_q} \circ \ldots \circ \tau_{k_1})$, de forma que $\langle \Lambda, R_k, \tau_{k_q} \circ \ldots \circ \tau_{k_1}\rangle \in \lambda_{lazy}(\tau(t))$.

La prueba en el sentido opuesto es completamente análoga.

El Lema 4 justifica la siguiente formalización del concepto de substitución adelantada, en un paso de *narrowing* necesario, que se muestra a continuación.

Definition 16 (Substitución Adelantada). *Sea $\mathcal{R}$ un programa. Sea t un término encabezado por un símbolo de operación y $\mathcal{P}$ un árbol definicional con pattern$(\mathcal{P}) = \pi$ tal que $\pi \leq t$. Las partes de substitución adelantada en el cómputo de $\lambda(t, \mathcal{P})$ se obtienen mediante la aplicación α, de términos y árboles definicionales a substituciones, definida inductivamente como sigue:*

$$\alpha(t,\mathcal{P}) \ni \begin{cases} id & \texttt{si existe una regla } R \texttt{ y una substitución } \sigma \\ & \texttt{tales que } \langle \Lambda, R, \sigma\rangle \in \lambda(t,\mathcal{P}), \\ \tau' \circ \tau & \texttt{si } t|_o = x \in \mathcal{X},\ \tau = \{x/c_i(x_1,\ldots,x_n)\}, \\ & \texttt{y } \tau' \in \alpha(\tau(t), \mathcal{P}_i); \\ \tau' \circ id & \texttt{si } t|_o = c_i(t_1,\ldots,t_n) \texttt{ y } \tau' \in \alpha(t,\mathcal{P}_i); \\ \tau' \circ id & \texttt{si } t|_o = g(t_1,\ldots,t_n),\ g \in \mathcal{F} \texttt{ y } \tau' \in \alpha(t|_o, \mathcal{P}') \\ & \textit{donde } \mathcal{P}' \textit{ es un árbol definicional para } g. \end{cases}$$

Siendo o la posición inductiva de π, $\pi_i = \pi[c_i(x_1, \ldots, x_n)]_o \in \mathcal{P}$ un hijo π, y $\mathcal{P}_i = \{\pi' \in \mathcal{P} \mid \pi_i \leq \pi'\}$ el árbol definicional donde todos los patrones son instancias de π_i.

Example 14. Sea el programa uniforme

$$\begin{array}{ll} R_1 : f(a,b) & \to c \\ R_2 : f(a,c) & \to b \\ R_3 : g(a) & \to c \\ R_4 : g(c) & \to b \end{array}$$

Para el término $t \equiv f(X, f(Y, g(Z)))$ y los árboles definicionales $\mathcal{P}$ de f y $\mathcal{P}'$ de g (construidos seleccionando las posiciones inductivas de izquierda a derecha) es fácil comprobar que $\langle 2.2, R_4, id \circ \{Z/c\} \circ id \circ \{Y/a\} \circ id \circ \{X/a\}\rangle \in \lambda(t, \mathcal{P})$. En este caso, la Definición 16 computa como substitución adelantada: $\tau = id \circ id \circ \{Y/a\} \circ id \circ \{X/a\} \in \alpha(t, \mathcal{P})$.

Intuitivamente, la parte de substitución adelantada τ se compone de los enlaces establecidos en el cómputo de $\lambda(t, \mathcal{P})$ antes de encontrar la ocurrencia demandada 2.2 y calcular: $\langle \Lambda, R_4, id \circ \{Z/c\}\rangle \in \lambda(f(a, f(a, g(c)))|_{2.2}, \mathcal{P}')$, que computa la parte de substitución no adelantada.

En el ejemplo anterior, puede comprobarse que para el término lineal $t \equiv f(X, f(Y, g(Z)))$, $\langle 2.2, R_4, \{Z/c\}\rangle \in \lambda_{lazy}(t)$. Así pues, en ese caso, lo que computa la Definición 16 coincide con nuestro concepto intuitivo de substitución adelantada (aquella parte de la substitución computada en un paso de *narrowing* necesario que no computa el correspondiente paso de *narrowing* perezoso). Sin embargo, el comportamiento de la aplicación α no siempre es tan reconfortante. Basta utilizar en los cómputos un término no lineal, como el término $s \equiv f(X, f(Y, g(Z)))$, para que nos demos cuenta de que el concepto definido por la aplicación α no se corresponde exáctamente con el concepto intuitivo de substitución adelantada.

Example 15. Para el programa uniforme del Ejemplo 14 y el término $s \equiv f(Z, f(Y, g(Z)))$ obtenemos que $\langle 2.2, R_3, id \circ id \circ id \circ \{Y/a\} \circ id \circ \{Z/a\}\rangle \in \lambda(t, \mathcal{P})$. En este caso, la Definición 16 computa como substitución adelantada: $\tau = id \circ id \circ \{Y/a\} \circ id \circ \{Z/a\} \in \alpha(t, \mathcal{P})$, mientras que $\langle 2.2, R_3, \{Z/a\}\rangle \in \lambda_{lazy}(s)$. Vemos que el correspondiente paso de *narrowing* perezoso no "desecha" alguno de los enlaces establecidos en el cómputo de $\lambda(s, \mathcal{P})$ antes de encontrar la ocurrencia demandada 2.2.

Sin embargo ésto no es un problema, ya que lo que pretendemos con la Definición 16 es disponer de una herramienta operacional con la que establecer una relación entre un paso de *narrowing* necesario y el correspondiente paso de *narrowing* perezoso.

Proposition 6. *Sea $\mathcal{R}$ un programa uniforme y R una regla de $\mathcal{R}$. Sea t un término encabezado por un símbolo de operación $f \in \mathcal{F}$ y $\mathcal{P}$ el árbol definicional de f. Entonces, $\langle p, R, \sigma \circ \tau\rangle \in \lambda(t, \mathcal{P})$ si y sólo si $\langle p, R, \sigma\rangle \in \lambda_{lazy}(\tau(t))$, donde $\tau \in \alpha(t, \mathcal{P})$ es la parte de sustitución adelantada en el cómputo de $\lambda(t, \mathcal{P})$.*

Proof. Por inducción sobre la complejidad del término $t \equiv f(t_1, \ldots, t_n)$, medida por el número de símbolos de función que aparecen en t, e inducción estructural sobre el árbol definicional $\mathcal{P}$. Las sustituciones se consideran módulo renombramiento de variables y restringidas a las variables del término t.

1. Caso base ($n = 1$): En este caso, en t no aparecen otros símbolos de función que f, así que los argumentos de t son variables o términos constructores. Por consiguiente, la única posibilidad para que $\langle p, R, \sigma \circ \tau\rangle \in \lambda(t, \mathcal{P})$ es que $p = \Lambda$. El Lema 4 y la Definición 16 de substitución adelentada nos aseguran que $\langle p, R, \sigma \circ \tau\rangle \in \lambda_{lazy}(t)$ y que $\tau = id$ es la parte de substitución adelantada (i.e., no hay substitución adelantada), por lo que el enunciado se cumple trivialmente.
2. Caso inductivo ($n > 1$): Siguiendo la definición de la estrategia de *narrowing* necesario, λ, distinguimos los siguientes casos.
   (a) $pattern(\mathcal{P}) = \pi$ es una hoja. En este caso, la función f está definida por una única regla $R \equiv f(x_1, \ldots, x_n) \to r$ y es inmediato comprobar que $\lambda(t, \mathcal{P}) = \lambda_{lazy}(t) = \{\langle \Lambda, R, id\rangle\}$. La idempotencia de la sustitución identidad nos permite afirmar que $\langle p, R, id \circ id\rangle \in \lambda(t, \mathcal{P})$ si y sólo si $\langle p, R, id\rangle \in \lambda_{lazy}(t)$.
   (b) $pattern(\mathcal{P}) = \pi$ es una rama. Sea o la posición inductiva de π. Ahora distinguimos entre las siguientes posibilidades:

i. $t|_o \equiv x$
Sea $\tau = x/c_i(x_1, \ldots, x_{n_i})$. Entonces $\langle p, R, \sigma' \circ \tau' \circ \tau\rangle \in \lambda(t, \mathcal{P})$ si $\langle p, R, \sigma' \circ \tau'\rangle \in \lambda(\tau(t), \mathcal{P}_i)$, donde τ' es la parte sustitución adelantada en el cómputo de $\lambda(\tau(t), \mathcal{P}_i)$. Por hipótesis de inducción $\langle p, R, \sigma'\rangle \in \lambda_{lazy}(\tau'(\tau(t)))$, donde, por Definición 16, $\tau' \circ \tau$ es la parte sustitución adelantada en el cómputo de $\lambda(t, \mathcal{P})$.

ii. $t|_o \equiv c(t_1, \ldots, t_n)$
Si $\langle p, R, \sigma' \circ \tau'\rangle \in \lambda(t, \mathcal{P}_i)$, donde τ' es la parte de sustitución adelantada en el cómputo de $\lambda(t, \mathcal{P}_i)$, entonces $\langle p, R, \sigma' \circ \tau' \circ id\rangle \in \lambda(t, \mathcal{P})$. Por hipótesis de inducción $\langle p, R, \sigma'\rangle \in \lambda_{lazy}(\tau'(t))$, donde $\tau' \circ id = \tau'$, por Definición 16, es la parte de sustitución adelantada en el cómputo de $\lambda(t, \mathcal{P})$.

iii. $t|_o \equiv g(t_1, \ldots, t_n)$
Entonces, $\langle o.p, R, \sigma' \circ \tau' \circ id\rangle \in \lambda(t, \mathcal{P})$ si $\langle p, R, \sigma' \circ \tau'\rangle \in \lambda(t|_o, \mathcal{P}')$, donde $\mathcal{P}'$ es el árbol definicional asociado a la operación g y τ' es la parte de sustitución adelantada en el cómputo de $\lambda(t|_o, \mathcal{P}')$.
Al evaluar $\lambda_{lazy}(t)$, por ser $\mathcal{R}$ uniforme, obtenemos que $\mathsf{LU}(\langle l_k, t\rangle) = (\textsc{Demand}, P)$, cualquiera que sea la lhs l_k. Además, $o \in P$ y no existe una posición $u < o$ que sea una posición perezosa de t, ya que entonces o no habría sido demandada. Luego para computar $\lambda_{lazy}(t)$ es necesario computar previamente $\lambda_{lazy}(t|_o)$, cumpliéndose que $\langle p, R_k, \sigma\rangle \in \lambda_{lazy}(t|_o)$ si y sólo si $\langle o.p, R_k, \sigma\rangle \in \lambda_{lazy}(t)$. Por hipótesis de inducción $\langle p, R, \sigma'\rangle \in \lambda_{lazy}(\tau'(t|_o))$, luego $\langle o.p, R, \sigma'\rangle \in \lambda_{lazy}(\tau'(t))$, donde $\tau' \circ id = \tau'$, por Definición 16, es la parte de sustitución adelantada en el cómputo de $\lambda(t, \mathcal{P})$.

Ahora es fácil establecer el análogo de la Proposición 5 para pasos de *narrowing* perezoso y pasos de *narrowing* necesario sobre programas uniformes.

Corollary 4. *Sea $\mathcal{R}$ un programa uniforme y R una regla de $\mathcal{R}$. Sea t un término encabezado por un símbolo de operación $f \in \mathcal{F}$ y $\mathcal{P}$ el árbol definicional de f. Entonces, existe el paso de* narrowing *necesario $t \overset{[p,R,\sigma\circ\tau]}{\leadsto}_{NN} s$ si y sólo si existe el paso de* narrowing *perezoso $\tau(t) \overset{[p,R,\sigma]}{\leadsto}_{LN} s$, donde $\tau \in \alpha(t, \mathcal{P})$ es la parte de sustitución adelantada en la computación $\lambda(t, \mathcal{P})$.*

Proof. Si $t \overset{[p,R,\sigma\circ\tau]}{\leadsto}_{NN} s$, por definición de paso de *narrowing*, $s \equiv \sigma(\tau(t[r]_p))$. Por la Proposición 6, $\langle p, R, \sigma\rangle \in \lambda_{lazy}(\tau(t))$, entonces puede darse el paso de *narrowing* $\tau(t) \overset{[p,R,\sigma]}{\leadsto}_{LN} \sigma(\tau(t)[r]_p)$. Dado que la substitución adelantada τ está restringida a las variables del término t, se cumple que

$$\sigma(\tau(t)[r]_p) = \sigma(\tau(t)[\tau(r)]_p) = \sigma(\tau(t[r]_p)) \equiv s.$$

La prueba en el otro sentido puede realizarse mediante un razonamiento completamente similar al anterior.

La idea intuitiva subyacente en el enunciado del Corolario 4 se ilustra mediante el siguiente ejemplo.

Example 16. Volviendo al Ejemplo 13 y centrándonos en el primer paso de la derivación de *narrowing* necesario para el término $t \equiv f(X, g(Y))$, tenemos que:

$$f(X, g(Y)) \overset{[2,R_2,id\circ\{Y/c\}\circ id\circ\{X/a\}]}{\leadsto_{NN}} f(a, b)$$

donde la respuesta computada se ha representado en forma canónica, para poner de manifiesto la relación existente entre este paso y el correspondiente paso empleando la estrategia de *narrowing* perezoso:

$$f(X, g(Y)) \overset{[2,R_2,\{Y/c\}]}{\leadsto_{LN}} f(X, b).$$

La parte de subtitución adelantada en el paso de *narrowing* necesario ha sido $\tau \equiv id \circ \{x/a\} \in \alpha(t, \mathcal{P})$, donde $\mathcal{P}$ es el árbol definicional de f. Si aplicamos esta subtitución τ al término t y damos un paso *narrowing* perezoso, obtenemos:

$$f(a, g(Y)) \overset{[2,R_2,\{Y/c\}]}{\leadsto_{LN}} f(a, b).$$

en concordancia con lo enunciado en el Corolario 4.

Naturalmente, la relación expresada por el Corolario 4 también se cumple para términos lineales, a pesar de que el concepto intuitivo de substitución adelantada no se corresponde con lo que computa la aplicación α, como ilustra el siguiente ejemplo.

Example 17. Para el programa uniforme del Ejemplo 14 y el término objetivo $s \equiv f(Z, f(Y, g(Z)))$, hemos visto que $\langle 2.2, R_3, id \circ id \circ id \circ \{Y/a\} \circ id \circ \{Z/a\}\rangle \in \lambda(s, \mathcal{P})$ donde $\tau = id \circ \{Y/a\} \circ id \circ \{Z/a\}$ es la parte de substitución adelantada computada por la aplicación α. Es fácil comprobar que, empleando *narrowing* necesario, puede darse el siguiente paso:

$$f(Z, f(Y, g(Z))) \overset{[2.2,R_3,id\circ id\circ id\circ\{Y/a\}\circ id\circ\{Z/a\}]}{\leadsto_{NN}} f(a, f(a, c))$$

donde, nuevamente, la respuesta computada se representa en forma canónica, para poner de manifiesto la relación existente entre este paso y el correspondiente paso empleando la estrategia de *narrowing* perezoso cuando aplicamos la subtitución τ al término s. El paso de *narrowing* perezoso correspondiente es:

$$f(a, f(a, g(a))) \overset{[2.2,R_3,id]}{\leadsto_{LN}} f(a, f(a, c))$$

en concordancia con lo enunciado en el Corolario 4.

Adviértase que la Proposición 6 fue probada para un árbol definicional fijado arbitráriamente. Terminamos este apartado haciendo notar que existe una identidad entre pasos de *narrowing* perezoso y pasos de *narrowing* necesario cuando seleccionamos un árbol definicional adecuado. Se cumple que, dado un término t, para cada paso de *narrowing* perezoso dado sobre una posición p de t, con una regla R y una substitución σ, existe un árbol definicional que permite dar un paso de *narrowing* necesario sobre la misma posición p de t, con la misma regla R y substitución σ. Motivamos este resultado mediante el siguiente ejemplo.

Example 18. Consideremos nuevamente el programa uniforme del Ejemplo 13:

$$\begin{aligned} R_1 &: f(a,b) \to c \\ R_2 &: g(c) \quad\ \to b \end{aligned}$$

Para el término $t \equiv f(x, g(y))$ puede darse el siguiente paso de *narrowing* perezoso

$$f(x,g(y)) \overset{[2,R_2,\{y/c\}]}{\leadsto_{LN}} f(x,b).$$

Ahora bien, si empleamos los árboles definicionales que se muestran en la Figura 7, en lugar de emplear árboles definicionales para f y g construidos en la forma obvia, puede darse el mismo paso empleando *narrowing* necesario:

$$f(x,g(y)) \overset{[2,R_2,\{y/c\}]}{\leadsto_{NN}} f(x,b).$$

Así pues, existe una representación adecuada de los árboles definicionales para la cual la estrategia de *narrowing* necesario coincide con la de *narrowing* perezoso en programas uniformes.

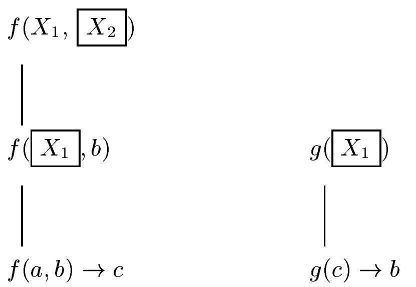

Fig. 7. Arboles definicionales para las funciones "f" y "g" del Ejemplo 18.

Proposition 7. *Sean $\mathcal{R}$ un programa uniforme y R una regla de $\mathcal{R}$. Sea t un término encabezado por un símbolo de operación $f/n \in \mathcal{F}$. Si $\langle p, R, \sigma\rangle \in \lambda_{lazy}(t)$ entonces existe un árbol definicional $\mathcal{P}$ de f, tal que $\langle p, R, \sigma\rangle \in \lambda(t, \mathcal{P})$.*

Proof. Realizamos esta prueba mostrando la posibilidad de construir un árbol definicional $\mathcal{P}$ para el que se cumple el enunciado. Emplearemos inducción sobre la estructura de la posición p.

1. Caso base ($p = \Lambda$): Por el Lema 4, $\langle \Lambda, R, \sigma\rangle \in \lambda_{lazy}(t)$ si y sólo si $\langle \Lambda, R, \sigma\rangle \in \lambda(t, \mathcal{P})$, cualquiera que sea el árbol definicional $\mathcal{P}$ de f. Por lo tanto el enunciado de la proposición se cumple de forma inmediata.
2. Caso inductivo ($p > \Lambda$): Por ser $\mathcal{R}$ un programa uniforme, la posición perezosa $p = i.q$, con $i \in \{1, \dots, n\}$, donde i es una posición inductiva de f. Dado que según el Algoritmo 1, para un programa uniforme la posición inductiva sobre la que se comienza a formar el árbol definicional puede ser elegida

de forma arbitraria, entonces sea $\mathcal{P}$ el árbol definicional resultante de elegir como primera posición inductiva la posición i.
Para que p sea una posición perezosa, debe cumplirse que $t|_i \equiv g(t_1, \ldots, t_m)$. Por otro lado, como se ha argumentado en el caso inductivo (b-iii) de la Proposición 6, por ser $\mathcal{R}$ un programa uniforme, $\langle q, R, \sigma \rangle \in \lambda_{lazy}(t|_i)$ si y sólo si $\langle i.q, R, \sigma \rangle \in \lambda_{lazy}(t)$. Por hipótesis de inducción existe un árbol definicional $\mathcal{P}'$ de g, tal que $\langle q, R, \sigma \rangle \in \lambda(t|_i, \mathcal{P}')$. Por la definición de estrategia de *narrowing* necesario λ, si $t|_i \equiv g(t_1, \ldots, t_m)$ y $\langle q, R, \sigma \rangle \in \lambda(t|_i, \mathcal{P}')$ entonces $\langle i.q, R, \sigma \rangle \in \lambda(t, \mathcal{P})$.
Así pues, posiblemente, existe un conjunto de árboles definicionales, aquellos que tienen como primera posición inductiva a la posición i, para los que se cumplirá el enunciado.

El siguiente corolario expresa el resultado anterior en términos de pasos de *narrowing*.

Corollary 5. *Sean $\mathcal{R}$ un programa uniforme y R una regla de $\mathcal{R}$. Sea t un término encabezado por un símbolo de operación $f \in \mathcal{F}$. Si $t \overset{[p,R,\sigma]}{\rightsquigarrow}_{LN} s$, entonces existe el árbol definicional $\mathcal{P}$ de f, tal que $t \overset{[p,R,\sigma]}{\rightsquigarrow}_{NN} s$.*

5.3 Narrowing Perezoso Uniforme.

Los resultados del apartado anterior, en especial las proposiciones 6 y 7, ponen de manifiesto que, para programas uniformes, las posiciones de un término computadas por la estrategia λ_{lazy} son *posiciones necesarias* de dicho término. Esta importante observación nos va a permitir introducir un refinamiento de la estrategia de *narrowing* perezoso, para el cual, ésta todavia puede probarse completa. La idea intuitiva detrás de este refinamiento es que si las posiciones que computa la estrategia de *narrowing* perezoso son posiciones necesarias, entonces puede elegirse cualquiera de ellas de una forma "don't care", ya que todas ellas deben ser explotadas para alcanzar el resultado. El orden en el que sean explotadas generará distintas derivaciones, pero sin afectar el resultado ni la respuesta obtenida.

Denominamos *narrowing perezoso uniforme* (ULN) a un refinamiento de la estrategia de *narrowing* perezoso que consiste en seleccionar el subconjunto de ternas asociados a una posición perezosa (por ejemplo, la más a la izquierda) para dar los correspondientes pasos de *narrowing*. Formalizamos este concepto mediante la siguiente definición:

Definition 17 (Estrategia de Narrowing Perezoso Uniforme).
Sea $\mathcal{R}$ un programa uniforme. Definimos la estrategia de *narrowing* perezoso uniforme *como una función λ_{ulazy} que, aplicada a un término t, computa el conjunto de ternas $\langle p, R_k, \sigma \rangle$, siendo $p \in \mathcal{FP}os(t)$ una posición perezosa de t, $R_k \equiv (l_k \to r_k)$ es una regla (renombrada aparte) de $\mathcal{R}$ y σ una substitución, como sigue:*

$$
\begin{array}{ll}
\lambda_{ulazy}(t) & = \bigcup_{k=1}^{m} \lambda\_(t, \Lambda, R_k) \\
\lambda\_(t, p, R_k) = & \texttt{si } \mathcal{H}ead(l_k) = \mathcal{H}ead(t|_p) \texttt{ entonces} \\
& \quad \texttt{en caso de que } \mathsf{LU}(\langle l_k, t_{|p}\rangle) = \\
& \quad \left\{ \begin{array}{ll} (\texttt{Succ}, \sigma): & \{\langle p, R_k, \sigma\rangle\} \\ (\texttt{Fail}, \varnothing): & \varnothing \\ (\texttt{Demand}, P): & \bigcup_{k=1}^{m} \lambda\_(t, p.q, R_k) \\ & \texttt{con } p = select\_don't\_care(P) \end{array} \right. \\
& \texttt{sino } \varnothing
\end{array}
$$

Donde la función $select\_don't\_care(S)$ elige arbitráriamente un elemento del conjunto S.

Dado un término t y una regla $R \equiv (l \to r)$ del programa $\mathcal{R}$, decimos que $t \overset{[p,R,\sigma]}{\leadsto}_{ULN} \sigma(t[r]_p)$ es un *paso de narrowing uniforme*, si $\langle p, R, \sigma\rangle \in \lambda_{ulazy}(t)$.

La Definición 17 asegura que sólamente se seleccionará, de forma "don't care", el subconjunto de ternas asociado a una posición perezosa. La estrategia que acabamos de introducir rompe la discrepancia entre el indeterminismo "don't care" en la selección inicial de los árboles definicionales que emplea la estrategia de *narrowing* necesario y el hecho de que las posiciones perezosas demandadas por la estrategia de *narrowing* perezoso se exploten todas ellas de forma "don't know".

Example 19. Volvamos sobre el programa uniforme del Ejemplo 12. Si en lugar de emplear el árbol definicional para la función "f" que ilustra la Figura 6, empleamos el representado en la Figura 8, puede obtenerse la siguiente derivación empleando *narrowing* necesario:

$$
\begin{array}{ll}
f(g(X), g(Y), g(Z)) \overset{[3,R_4,\{Z/b\}]}{\leadsto}_{NN} & f(g(X), g(Y), b) \\
\overset{[2,R_3,\{Y/a\}]}{\leadsto}_{NN} & f(g(X), a, b) \\
\overset{[\Lambda,R_1,id]}{\leadsto}_{NN} & 1.
\end{array}
$$

Podemos apreciar que esta derivación se corresponde paso a paso con la derivación de *narrowing* perezoso "redundante", obtenida en el Ejemplo 12.

El ejemplo anterior muestra que la derivación de *narrowing* perezoso "redundante", obtenida en el Ejemplo 12, se corresponde con la que calcularía la estrategia de *narrowing* necesario eligiendo la representación alternativa para el árbol definicional de f mostrada en la Figura 8. Por lo tanto, cada una de las derivaciones "redundantes" de *narrowing* perezoso que aparecen al computar un término en un programa uniforme están asociadas a una de las posibles derivaciones de *narrowing* necesario para un árbol definicional fijado.

Justificamos el refinamiento introducido mediante los siguientes resultados. El primero de ellos indica que para programas uniformes, dado un término y fijado un árbol definicional para la raíz de dicho término, la estrategia de *narrowing* necesario computa una única posición necesaria (a la que posiblemente hay asociadas varias ternas correspondientes a diferentes reglas).

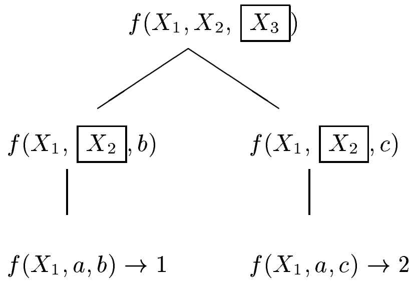

Fig. 8. Arbol definicional para la función "f"

Proposition 8. *Sea $\mathcal{R}$ un programa uniforme. Sean t un término encabezado por un símbolo de operación $f \in \mathcal{F}$ y $\mathcal{P}$ un árbol definicional de f. Si $\lambda(t, \mathcal{P}) = \{\langle p_1, R_1, \sigma_1 \rangle, \ldots, \langle p_k, R_k, \sigma_k \rangle\}$ entonces $p_1 = \ldots = p_k = p$.*

Proof. Inmediata, apartir de la Definición 15 y la estructura de un programa uniforme, Lema 3.

La Proposición 7 y la Proposición 8 aseguran que, para programas uniformes, las posiciones perezosas de un término t computadas por la estrategia λ_{lazy} son disjuntas, formando subconjuntos de ternas asociados, cada uno de ellos, a un paso de *narrowing* necesario $\lambda(t, \mathcal{P})$, dado con un árbol definicional $\mathcal{P}$ de f, siendo f el símbolo que encabeza el término t.

Proposition 9. *Sean $\mathcal{R}$ un programa uniforme y t un término. Si $\langle p, R, \sigma \rangle \in \lambda_{lazy}(t)$ y $\langle p', R', \sigma' \rangle \in \lambda_{lazy}(t)$ entonces $p = p'$ o $p \not\leq p'$.*

Proof. Inmediata, haciendo uso de la Proposición 7 y de la Proposición 8.

La Proposición 8 justifica, intuitivamente, la corrección de nuestro refinamiento: dado que la estrategia de *narrowing* necesario es correcta y completa para programas inductivamente secuenciales (de los que los programas uniformes son una subclase), cuando empleamos la estrategia de *narrowing* perezoso basta con explotar uno de los subconjuntos de ternas disjuntos asociados a las posiciones perezosas.

La estrategia de *narrowing* perezoso uniforme es una estrategia "intermedia" entre la de *narrowing* necesario y la de *narrowing* perezoso, que tiene las buenas propiedades de evitar: i) el exceso de instanciación del *narrowing* necesario; ii) las derivaciones redundantes del *narrowing* perezoso.

5.4 Equivalencia de las Estrategias de Evaluación Perezosa y Corrección del Narrowing Perezoso Uniforme.

Hasta el momento hemos establecido la relación existente entre 'pasos' correspondientes a las estrategias de *narrowing* perezoso y de *narrowing* necesario. A continuación investigaremos la relación existente entre derivaciones de longitud arbitraria. Aunque en este apartado nos ceñimos a las relaciones entre la

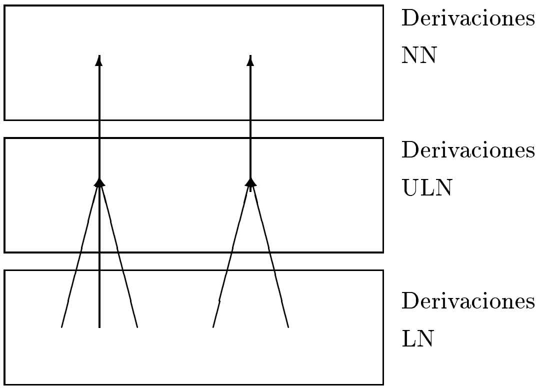

Fig. 9. Relación entre las estrategias de evaluación perezosa sobre programas uniformes

estrategia de *narrowing* necesario y la de *narrowing* perezoso uniforme, queremos hacer notar que los principales resultados pueden extenderse a la estrategia de *narrowing* perezoso, si bien, en este caso, a una derivación de *narrowing* necesario podrán corresponderle varias derivaciones de *narrowing* perezoso sólo distinguibles por el orden en el que se dan los pasos en la derivación.

Nuestro interes primordial es probar la equivalencia de estas estrategias y demostrar la corrección y completitud de la estrategia de *narrowing* perezoso uniforme que acabamos de introducir. El siguiente ejemplo muestra que, para árboles definicionales fijados, estas estrategias no son equivalentes (en el sentido de que no computan la misma salida) a menos que evaluemos un término hasta alcanzar la hnf.

Example 20. Sea el programa uniforme

$$\begin{array}{ll} R_1 : f(c(X), a, b) & \to b \\ R_2 : g(c(Y)) & \to Y \\ R_3 : h(b) & \to b \end{array}$$

Para el término $t \equiv f(X, g(Z), h(W))$ existe una única derivación empleando *narrowing* perezoso uniforme (seleccionando el subconjunto de ternas asociado a la posición perezosa más a la izquierda):

$$f(X, g(Z), h(W)) \overset{[2,R_2,\{Z/c(Y)\}]}{\rightsquigarrow}_{ULN} f(X, Y, h(W)) \overset{[3,R_3,\{W/b\}]}{\rightsquigarrow}_{LN} f(X, Y, b),$$

que produce la salida $[f(X, Y, b), \{W/b, Z/c(Y)\}]$. Por el contrario, la derivación obtenida construyendo los árboles definicionales de las funciones f, g y h en la forma obvia y empleando la estrategia de *narrowing* necesario es:

$$\begin{array}{ll} f(X, g(Z), h(W)) \overset{[2,R_2,\{Z/c(Y_2)\}\circ\{X/c(X_4)\}]}{\rightsquigarrow}_{NN} & f(c(X_4), Y_2, h(W)) \\ \qquad\qquad \overset{[3,R_3,\{W/b\}\circ\{Y_2/a\}]}{\rightsquigarrow}_{NN} & f(c(X_4), a, b), \end{array}$$

que produce la salida $[f(c(X_4), a, b), \{X/c(X_4), W/b, Z/c(a)\}]$. Sin embargo, al dar el siguiente paso (que lleva los términos a la hnf), ambas derivaciones computan el mismo resultado con la misma respuesta (salvo renombramientos). Hacemos notar también que la estrategia de *narrowing* perezoso permite una derivación adicional en la que primero se da un paso con la regla R_3 y después con la R_2.

El siguiente teorema establece la equivalencia de la semántica operacional, para respuestas computadas, cuando empleamos las estrategias de *narrowing* necesario y *narrowing* perezoso uniforme.

Theorem 4. *Sea $\mathcal{R}$ un programa uniforme. Sea t un término encabezado por un símbolo de operación $f \in \mathcal{F}$.*

1. *Existe una derivación de* narrowing *necesario $\mathcal{D} \equiv (t \overset{\sigma}{\leadsto}_{NN}^{*} s)$, donde s es una hnf, si y sólo si existe una derivación de* narrowing *perezoso uniforme $\mathcal{D}' \equiv (t \overset{\sigma}{\leadsto}_{ULN}^{*} s)$.*
2. *$\mathcal{D}$ y $\mathcal{D}'$ son derivaciones con el mismo número de pasos y que emplean las mismas reglas sobre las mismas posiciones en los pasos correspondientes de cada derivación.*

Para demostrar el Teorema 4 necesitaremos los siguientes resultados axiliares.

Primero probamos que, para programas uniformes, se cumple la siguiente propiedad entre los distintos componentes de la representación canónica de un paso de *narrowing* necesario.

Lemma 5. *Sea $\mathcal{R}$ un programa uniforme. Sea t un término encabezado por el símbolo de función f y $\mathcal{P}$ el árbol definicional de f. Si $\langle p, R, \vartheta_k \circ \cdots \circ \vartheta_1 \rangle \in \lambda(t, \mathcal{P})$ es un paso de* narrowing *necesario, entonces:*

1. *para $i, j \in \{1, \ldots, k\}$, con $i \neq j$, $\mathcal{V}ar(\vartheta_i) \cap \mathcal{V}ar(\vartheta_j) = \varnothing$.*
2. *para $i \in \{1, \ldots, k\}$, si $\vartheta_i = \{x/c_i(x_1, \ldots, x_{n_i})\}$ entonces $x \in \mathcal{V}ar(t)$.*

Proof. Procedemos por inducción sobre k e inducción estructural sobre el árbol definicional $\mathcal{P}$

1. Caso base ($k = 1$): En este caso, necesariamente $pattern(\mathcal{P}) = \pi$ es una hoja, de otro modo $k > 1$ en contra de lo supuesto. Por Definición 14 de estrategia de *narrowing* necesario, $\lambda(t, \mathcal{P}) = \{\langle \Lambda, \pi \to r, id \rangle\}$ y el enunciado del lema se cumple trivialmente.
2. Caso inductivo ($k > 1$): De acuerdo con la Definición 14 de estrategia de *narrowing* necesario, tenemos que $pattern(\mathcal{P}) = \pi$ es una rama y distinguimos las siguientes posibilidades: Sea o_1 la posición inductiva correspondiente a esta rama, entonces $\langle p, R, \vartheta_k \circ \cdots \circ \vartheta_1 \rangle \in \lambda(t, \mathcal{P})$ si
   (a) $t|_{o_1} \equiv x$, $\vartheta_1 = \{x/c_1(x_1, \ldots, x_{n_1})\}$ y $\langle p, R, \vartheta_k \circ \cdots \circ \vartheta_2 \rangle \in \lambda(\vartheta_1(t), \mathcal{P}_1)$. Por hipótesis de inducción las substituciones ϑ_j con $j \in \{2, \ldots, k\}$ cumplen el enunciado. Además, $x \in \mathcal{V}ar(t)$, con lo que ϑ_1 cumple también cumple el punto (2) del enunciado y por lo tanto se cumple en

general. Para probar que en este caso se cumple el punto (1) razonamos del siguiente modo:
Por ser $\mathcal{R}$ un programa uniforme, las posiciones inductivas de $\mathcal{P}$ se corresponden con los argumentos, en los que aparecen constructores planos, de los términos lhs que definen f (Proposición 4). Por lo tanto si o es una posición inductiva y o' es una posición de t con $o < o'$ entonces o' no es una posición inductiva. En otras palabras las posiciones inductivas en $\mathcal{P}$ se corresponden con posiciones disjuntas del término t.
Sea o_2 una posición inductiva de $\mathcal{P}$ y por lo tanto disjunta de o_1. Así que, es imposible que $\vartheta_1(t)|_{o_2} \equiv y$ con $y \in \{x_1, \ldots, x_{n_1}\}$. Se presentan las siguientes posibilidades:

i. $t|_{o_2} \equiv x$.
Entonces, $\vartheta_1(t)|_{o_2} \equiv c_1(x_1, \ldots, x_{n_1})$ y por consiguiente $\vartheta_2 = id$. Así que, de manera trivial ϑ_1 no comparte variables con ϑ_2.

ii. $t|_{o_2} \equiv y$.
Entonces, $\vartheta_1(t)|_{o_2} \equiv y$. En este caso, $\vartheta_2 = \{y/c_2(y_1, \ldots, y_{n_2})\}$ y por construcción del árbol definicional $\mathcal{P}$, las variable $y_1, \ldots, y_{n_2}$ son frescas. Así que, ϑ_1 no comparte variables con ϑ_2.

iii. $t|_{o_2} \equiv c_2(t_1, \ldots, t_{n_2})$.
Entonces, $\vartheta_1(t)|_{o_2} \equiv c_2(\vartheta_1(t_1), \ldots, \vartheta_1(t_{n_2}))$ y por consiguiente $\vartheta_2 = id$. Así que, de manera trivial ϑ_1 no comparte variables con ϑ_2.
Notad que el caso en el que $t|_{o_2} \equiv c(t_1, \ldots, t_n)$, con $c \neq c_2$, no puede darse, ya que, entonces $\lambda(t, \mathcal{P})$ no sería un paso de *narrowing* necesario.

iv. $t|_{o_2} \equiv g(t_1, \ldots, t_{n_2})$.
Idéntico al caso anterior.

Ahora podemos repetir este razonamiento, de forma que podemos probar que ϑ_1 no comparte variables con el resto de las substituciones ϑ_j que forman la representación canónica de $\lambda(t, \mathcal{P})$.

(b) $t|_{o_1} \equiv c_1(t_1, \ldots, t_n)$, $\vartheta_1 = id$ y $\langle p, R, \vartheta_k \circ \cdots \circ \vartheta_2 \rangle \in \lambda(t, \mathcal{P}_1)$.
Por hipótesis de inducción las substituciones ϑ_j con $j \in \{2, \ldots, k\}$ cumplen el enunciado. Por otro lado, en este caso trivialmente ϑ_1 no comparte variables con el resto de los ϑ_j con $j \in \{2, \ldots, k\}$ y el punto (2) del enunciado se cumple por vacuidad para ϑ_1.

(c) $t|_{o_1} \equiv g(t_1, \ldots, t_n)$, $\vartheta_1 = id$ y $\langle p, R, \vartheta_k \circ \cdots \circ \vartheta_2 \rangle \in \lambda(t|_{o_1}, \mathcal{P}')$, donde $\mathcal{P}'$ es un árbol definicional para g.
Idéntico al caso anterior.

Como consecuencia del lema anterior, los elementos de la substitución computada en un paso de *narrowing* necesario, representada en forma canónica, están restringidos a las variables del término t. Además, debido a que sus componentes no comparten variables, puede entenderse como formando una partición, de manera que esos componentes pueden ser reordenados sin afectar al resultado final de la composición. En otras palabras, los operadores de composición "$\circ$" y de unión de conjuntos "$\cup$" son intercambiables para los componentes de una substitución computada en un paso de *narrowing* necesario. Este resultado se formaliza mediante el siguiente corolario y haremos uso extensivo de él en adelante.

Corollary 6. *Sea $\mathcal{R}$ un programa uniforme. Sea t un término encabezado por el símbolo de función f y $\mathcal{P}$ el árbol definicional de f. Si $\langle p, R, \vartheta_k \circ \cdots \circ \vartheta_1 \rangle \in \lambda(t, \mathcal{P})$ es un paso de* narrowing *necesario, entonces $\vartheta_k \circ \cdots \circ \vartheta_1 = \vartheta_k \cup \cdots \cup \vartheta_1$*

Al comentar el Ejemplo 15 indicamos que algunas de las substituciones computadas por la aplicación α no eran desechadas en un paso de *narrowing* perezoso. Apoyándonos en el Corolario 6 podemos escribir $(\tau_2 \cup \tau_1) \in \alpha(t, \mathcal{P})$ distinguiendo entre la parte que es desechada, τ_1, y la que no es desechada, τ_2, en un paso de *narrowing* perezoso a partir de t. El siguiente lema pone de manifiesto que la parte desechada en el paso de *narrowing* perezoso vuelve a ser computada por la estrategia λ en el paso siguiente.

Lemma 6. *Sea $\mathcal{R}$ un programa uniforme. Sea t un término encabezado por el símbolo de función f y $\mathcal{P}$ el árbol definicional de f. Sea $(\tau_2 \cup \tau_1) \in \alpha(t, \mathcal{P})$. Si $\langle p, R, \sigma \cup \tau_2 \cup \tau_1 \rangle \in \lambda(t, \mathcal{P})$ y $t \overset{[p,R,\sigma\cup\tau_2]}{\leadsto_{LN}} t'$ un paso de* narrowing *perezoso y $p \not\equiv \Lambda$ entonces $\langle p', R', \sigma' \cup \tau_1 \rangle \in \lambda(t', \mathcal{P})$.*

Proof. Inmediata, sin más que darse cuenta de que las posiciones disjuntas o por encima de la posición p permanecen inalteradas después del paso de *narrowing*, salvo las posibles instanciaciones de las variables $x \in \mathcal{D}om(\sigma \cup \tau_2)$. Además, por el Lema 5 $\mathcal{D}om(\sigma \cup \tau_2) \cap \mathcal{D}om(\tau_1) = \varnothing$. Por lo tanto, $\lambda(t', \mathcal{P})$ computa de nuevo la parte de substitución τ_1.

El siguiente lema relaciona derivaciones de *narrowing* perezoso uniforme a partir de un término t y la correspondiente derivación obtenida cuando al término t se le aplica, previamente, la partición desechada de la substitución adelantada $\alpha(t, \mathcal{P})$.

Lemma 7. *Sea $\mathcal{R}$ un programa uniforme. Sea t un término encabezado por un símbolo de operación $f \in \mathcal{F}$. Sea $\mathcal{P}$ el árbol definicional de f. Sea $\tau = (\tau_2 \cup \tau_1) \in \alpha(t, \mathcal{P})$ la substitución adelantada en un paso de* narrowing *necesario. Sea $\langle p_1, R_1, \sigma_1 \cup \tau_2 \cup \tau_1 \rangle \in \lambda(t, \mathcal{P})$ y $\langle p_1, R_1, \sigma_1 \cup \tau_2 \rangle \in \lambda_{ulazy}(t)$. Existe una derivación de* narrowing *perezoso uniforme $t \overset{\theta\circ\tau_1}{\leadsto_{ULN}}{}^{*} s$ si y sólo si existe una derivación de* narrowing *perezoso uniforme $\tau_1(t) \overset{\theta}{\leadsto_{ULN}}{}^{*} s$, donde el término s es una hnf y $\theta = \sigma \circ (\sigma_1 \cup \tau_2)$.*

Proof. Por inducción sobre el número de pasos, n, de las derivaciones. Las sustituciones se consideran módulo renombramiento de variables y restringidas a las variables del término t.

1. Caso base ($n = 1$): En este caso el paso de *narrowing* perezoso, $t \overset{[p_1,R_1,\sigma_1\cup\tau]}{\leadsto_{ULN}} s$, tiene que darse sobre la posición $p_1 = \Lambda$; sino, el paso se daría sobre una posición $p_1 > \Lambda$ y por definición de paso de *narrowing*, el término s estaría encabezado por un símbolo de función f, en contra de lo supuesto en el enunciado. Por el Lema 4, $\tau = id$, $\langle \Lambda, R, \sigma_1 \rangle \in \lambda(t, \mathcal{P})$ y $\langle \Lambda, R, \sigma_1 \rangle \in \lambda_{ulazy}(t)$. Así pues, el enunciado del lema se cumple trivialmente.

2. Caso inductivo ($n > 1$): Si existe la derivación de *narrowing* perezoso uniforme $t \overset{\theta\circ\tau_1}{\leadsto}{}^*_{ULN}\ s$, podemos dividir esta derivación, en dos partes, de la siguiente forma:

$$t \overset{[p_1,R_1,\sigma_1\cup\tau_2]}{\leadsto}_{ULN}\ t_1 \overset{\sigma\circ\tau_1}{\leadsto}{}^*_{ULN}\ s.$$

El caso en el que $p_1 = \Lambda$, es inmediato, ya que por el Lema 4 $\tau = id$, así que discutamos el caso en el que $p_1 \neq \Lambda$. Por el Lema 6, $\langle p_2, R_2, \sigma_2 \cup \tau_1\rangle \in \lambda(t_1, \mathcal{P})$, donde p_2 y R_2 son, respectivamente, la posición y la regla con las que se da el segundo paso de *narrowing* en la derivación. Ahora distinguimos dos casos, según se cumpla:

(a) $\langle p_2, R_2, \sigma_2 \cup \tau_1\rangle \in \lambda_{ulazy}(t_1)$.
Entonces, el enlace adelantado se computa en este paso y de forma evidente, por definición de paso de *narrowing* y estar τ_1 restringida a las variables de t, puede darse el paso de *narrowing* $\langle p_2, R_2, \sigma_2\rangle \in \lambda_{ulazy}(\tau_1(t_1))$. Por lo tanto, si la derivación $t_1 \overset{\sigma\circ\tau_1}{\leadsto}{}^*_{ULN}\ s$ está constituida por los pasos de *narrowing*

$$t_1 \overset{[p_2,R_2,\sigma_2\cup\tau_1]}{\leadsto}_{ULN}\ t_2 \overset{\sigma'}{\leadsto}{}^*_{ULN}\ s,$$

donde $\sigma = \sigma' \circ \sigma_2$, ahora puede formarse la derivación alternativa

$$\tau_1(t_1) \overset{[p_2,R_2,\sigma_2]}{\leadsto}_{ULN}\ t_2 \overset{\sigma'}{\leadsto}{}^*_{ULN}\ s.$$

Esto es, existe la derivación $\tau_1(t_1) \overset{\sigma}{\leadsto}{}^*_{ULN}\ s$.

(b) $\langle p_2, R_2, \sigma_2\rangle \in \lambda_{ulazy}(t_1)$.
Esto es, el enlace adelantado se computará en alguno de los pasos de *narrowing* posteriores. En este caso se cumplen las condiciones para aplicar la hipótesis de inducción. Por lo tanto, dado que existe la derivación $t_1 \overset{\sigma\circ\tau_1}{\leadsto}{}^*_{ULN}\ s$, con menos de n pasos, por hipótesis de inducción, existe la derivación $\tau_1(t_1) \overset{\sigma}{\leadsto}{}^*_{ULN}\ s$.

Por otro lado, dado que se puede dar el paso de *narrowing* perezoso uniforme $t \overset{[p_1,R_1,\sigma_1\cup\tau_2]}{\leadsto}_{ULN}\ t_1 = \sigma_1(\tau_2((t[r_1]_{p_1})))$, supueto que $R_1 \equiv l_1 \to r_1$, podemos asegurar que el correspondiente problema de unificación lineal tiene éxito, esto es, $\mathsf{LU}(\langle l_1, t|_{p_1}\rangle) = (\textsc{Succ}, \sigma_1 \cup \tau_2)$. En lo que sigue comprobaremos que, $\mathsf{LU}(\langle l_1, \tau_1(t)|_{p_1}\rangle) = (\textsc{Succ}, \sigma_1 \cup \tau_2)$:

(a) Primero notad que al ser τ_1 una substitución constructora lineal (Proposición 3), no introduce nuevos redexes cuando se aplica al término t. Por consiguiente, el problema de unificación lineal, $\langle l_1, \tau_1(t)|_{p_1}\rangle$ también tendrá éxito.

(b) Sean $l_1 = f(d_1, \ldots, d_k)$ y $t|_{p_1} = f(s_1, \ldots, s_k)$. Que el problema de unificación lineal $\langle l_1, t|_{p_1}\rangle$ tiene éxito significa que la configuración inicial LU

$$(\{d_1 \downarrow_1 s_1, \ldots, d_k \downarrow_k s_k\}, id)$$

es reducible mediante la relación $\to_{\mathsf{LU}}$ a la configuración irreducible $(\varnothing, \sigma_1 \cup \tau_2)$. Estudiemos las características de esta reducción, fijándonos en el elemento $d_i \downarrow_i s_i$ de la configuración LU inicial. Por ser $\mathcal{R}$ un programa uniforme, d_i es una variable o un término constructor plano y lineal.

En el primer caso, si $d_i \equiv z_i$, obtenemos la substitución $\{z_i/s_i\}$ (i.e., id, cuando nos restringimos a las variables del término t). En el segundo caso, si $d_i \equiv c_i(z_1, \ldots, z_m)$, pueden darse las siguientes posibilidades:

i. $s_i \equiv x \in \mathcal{V}ar(t)$.
Obtenemos la substitución $\{x/c_i(z_1, \ldots, z_m)\}$. Dado que l_1 es un patrón lineal, cuyos argumentos son variables o constructores planos, y las reglas se toman renombradas aparte, las variables $z_1, \ldots, z_m$ no forman parte del dominio de ninguna otra substitución que computemos en la derivación LU. Así pues, esta substitución es un elemento de $\sigma_1 \cup \tau_2$.

ii. $s_i \equiv c_i(s'_1, \ldots, s'_m)$, con $c_i \in \mathcal{C}$.
Con lo que el elemento $d_i \downarrow_i s_i$ de la configuración LU inicial se transforma en $c_i(z_1, \ldots, z_m) \downarrow_i c_i(s'_1, \ldots, s'_m)$ y la configuración LU inicial, después de un paso de reducción LU, quedaría:
$$(\{d_1 \downarrow_1 s_1, \ldots, z_1 \downarrow_{i.1} s'_1, \ldots, z_m \downarrow_{i.m} s'_m, \ldots, d_k \downarrow_k s_k\}, id).$$
En el siguiente paso obtendríamos las substituciones $\{z_i/s'_i\}$ (i.e., id, cuando nos restringimos a las variables del término t).

Notad que los casos en los que $s_i \equiv c_j(s'_1, \ldots, s'_m)$, con $c_j \in \mathcal{C}$ y $c_j \neq c_i$, o $s_i \equiv g(s'_1, \ldots, s'_m)$, con $g \in \mathcal{F}$, no pueden darse, ya que entonces el problema de unificación lineal no tendria éxito, en contra de lo supuesto.

(c) Si aplicamos la substitución τ_1 al término $t|_{p_1}$, el problema de unificación lineal planteado $\langle l_1, \tau_1(t|_{p_1})\rangle$ conduce a la configuración inicial LU
$$(\{d_1 \downarrow_1 \tau_1(s_1), \ldots, d_k \downarrow_k \tau_1(s_k)\}, id)$$
que debe ser reducible mediante la relación $\rightarrow_{\mathsf{LU}}$ a una configuración ireducible. Estudiemos, nuevamente, las características de esta reducción, fijándonos en un elemento arbitrario $d_i \downarrow_i \tau_1(s_i)$ de la configuración LU inicial y realizando un análisis de casos como el anterior. Si $d_i \equiv z_i$ obtenemos la substitución $\{z_i/\tau_1(s_i)\}$ (i.e., id, cuando nos restringimos a las variables del término t). Si $d_i \equiv c_i(z_1, \ldots, z_m)$, pueden darse las siguientes posibilidades:

i. $s_i \equiv x \in \mathcal{V}ar(t)$, con $x \notin \mathcal{D}om(\tau_1)$.
Entonces $\tau_1(s_i) = x$ y obtenemos la substitución $\{x/c_i(z_1, \ldots, z_m)\}$. Dado que l_1 es un patrón lineal, cuyos argumentos son variables o constructores planos, y las reglas se toman renombradas aparte, esta substitución es el elemento de $\sigma_1 \cup \tau_2$ que se computó en el punto (2.b.i).

ii. $s_i \equiv x \in \mathcal{V}ar(t)$, con $x \in \mathcal{D}om(\tau_1)$.
Entonces $\tau_1(s_i) = \tau_1(x)$, con lo que el elemento $d_i \downarrow_i \tau_1(s_i)$ de la configuración LU inicial se transforma en $c_i(z_1, \ldots, z_m) \downarrow_i \tau_1(x)$. Como el problema de unificación lineal tiene éxito, $\tau_1(x)$ debe ser un término encabezado por el símbolo constructor c_i. Esto es, $\tau_1(x) = c_i(s'_1, \ldots, s'_m)$ y la configuración LU inicial, después de un paso de reducción LU, quedaría:
$$(\{d_1 \downarrow_1 \tau_1(s_1), \ldots, z_1 \downarrow_{i.1} s'_1, \ldots, z_m \downarrow_{i.m} s'_m, \ldots, d_k \downarrow_k \tau_1(s_k)\}, id).$$
En el siguiente paso obtendríamos las substituciones $\{z_i/s'_i\}$ (i.e., id, cuando nos restringimos a las variables del término t).

iii. $s_i \equiv c_i(s'_1, \ldots, s'_m)$, con $c_i \in \mathcal{C}$.
También obtendríamos la substitución id, cuando nos restringimos a las variables del término t.

Mediante este análisis, podemos concluir que $\mathsf{LU}(\langle\langle l_1, \tau_1(t)|_{p_1}\rangle\rangle) = (\textsc{Succ}, \sigma_1 \cup \tau_2)$ y por lo tanto que $\langle p_1, R_1, \sigma_1 \cup \tau_2\rangle \in \lambda_{ulazy}(\tau_1(t))$. Luego, podemos dar el paso de *narrowing* perezoso uniforme

$$\begin{aligned}\tau_1(t) \overset{[p_1,R_1,\sigma_1\cup\tau_2]}{\leadsto_{ULN}} & \ \sigma_1(\tau_2((\tau_1(t)[r_1]_{p_1}))) \\ &= \sigma_1(\tau_2(\tau_1(t[r_1]_{p_1}))) \\ &= \tau_1(\sigma_1(\tau_2(t[r_1]_{p_1}))) \\ &= \tau_1(t_1)\end{aligned}$$

ya que la substitución τ_1 está restringida a las variables del término t y σ_1, τ_1 y τ_2 son substituciones constructoras lineales que no comparte variables entre ellas (Lema 5), lo que nos permite conmutar las substituciones. Ahora, podemos construir la siguiente derivación de *narrowing* perezoso:

$$\tau_1(t) \overset{[p_1,R_1,\sigma_1\circ\tau_2]}{\leadsto_{ULN}} \tau_1(t_1) \overset{\sigma'}{\leadsto_{ULN}}{}^{*} s.$$

La prueba en el sentido opuesto es completamente análoga.

Corollary 7. *Sea $\mathcal{R}$ un programa uniforme. Sea t un término encabezado por un símbolo de operación $f \in \mathcal{F}$. Sea $\mathcal{P}$ el árbol definicional de f. Sea $\tau \in \alpha(t, \mathcal{P})$ la substitución adelantada en un paso de* narrowing *necesario. Existe una derivación de* narrowing *perezoso uniforme $t \overset{\sigma\circ\tau}{\leadsto_{ULN}}{}^{*} s$, donde el término s es una hnf, si y sólo si existe una derivación de* narrowing *perezoso uniforme $\tau(t) \overset{\sigma}{\leadsto_{ULN}}{}^{*} s$.*

Proof. Inmediata, por el Lema 7 y la definición de paso de *narrowing*.

Sea $\tau = (\tau_2 \cup \tau_1) \in \alpha(t, \mathcal{P})$, donde τ_2 representan los enlaces que se establecen en el primer paso de *narrowing* perezoso uniforme y que por lo tanto no son desechados. Por el Lema 7, dado que existe la derivación de *narrowing* perezoso uniforme $t \overset{[\theta\circ\tau_1]}{\leadsto_{ULN}}{}^{*} s$ entoces existe una derivación de *narrowing* perezoso uniforme $\tau_1(t) \overset{\theta}{\leadsto_{ULN}}{}^{*} s$, donde el término s es una hnf y $\theta = \sigma \circ (\sigma_1 \cup \tau_2)$. Ahora bien, por definición de paso de *narrowing* si existe el paso de *narrowing* perezoso uniforme

$$\tau_1(t) \overset{[p_1,R_1,\sigma_1\cup\tau_2]}{\leadsto_{ULN}} t_2 \equiv \sigma_1(\tau_2(\tau_1(t)[r_1]_{p_1}))$$

existe el paso de *narrowing* perezoso uniforme

$$\begin{aligned}\tau_2(\tau_1(t)) \overset{[p_1,R_1,\sigma_1]}{\leadsto_{ULN}} & \ \sigma_1(\tau_2(\tau_1(t))[r_1]_{p_1}) \\ &= \sigma_1(\tau_2(\tau_1(t)[r_1]_{p_1})) \\ &= t_2\end{aligned}$$

ya que la substitución τ_1 está restringida a las variables del término t. Ahora, podemos construir la siguiente derivación de *narrowing* perezoso uniforme:

$$\tau_2(\tau_1(t)) \overset{[p_1,R_1,\sigma_1]}{\leadsto_{ULN}} t_2 \overset{\sigma}{\leadsto_{ULN}}{}^{*} s.$$

La prueba en el sentido contrario se puede establecer de forma analoga.

A continuación estamos en disposición de probar el Teorema 4 y de establecer la equivalencia entre las semánticas operacionales por *narrowing* necesario y por *narrowing* perezoso uniforme.

Proof. 1. Demostramos el punto (1) del enunciado por inducción sobre el número de pasos, n, de la derivación $\mathcal{D}$. Las sustituciones se consideran módulo renombramiento de variables.

(a) Caso base ($n = 1$):
Idéntico al caso base del Lema 7.

(b) Caso inductivo ($n > 1$):
Sea
$$\mathcal{D} \equiv (t \overset{[p_1,R_1,\sigma_1\circ\tau_1]}{\leadsto_{NN}} t_1 \overset{[p_2,R_2,\sigma_2\circ\tau_2]}{\leadsto_{NN}}, \ldots \overset{[p_{n-1},R_n,\sigma_n\circ\tau_n]}{\leadsto_{NN}} s),$$
donde $\theta_i = (\sigma_i \circ \tau_i)$ y $\tau_i \in \alpha(s_{i-1}, \mathcal{P}_{i-1})$ es la substitución adelantada en cada paso $i \in \{1, \ldots, n\}$. La derivación $\mathcal{D}$ puede dividirse en dos partes como, indicamos a continuación:
$$t \overset{[p_1,R_1,\theta_1]}{\leadsto_{NN}} t_1 \overset{\theta_n\circ\cdots\circ\theta_2}{\leadsto_{NN}}{}^{*}\ s.$$
Dado que existe el paso de *narrowing* necesario $t \overset{[p_1,R_1,\theta_1]}{\leadsto_{NN}} t_1$, por el Corolario 4, existe el paso de *narrowing* perezoso uniforme $\tau_1(t) \overset{[p_1,R_1,\sigma_1]}{\leadsto_{ULN}} t_1$. También, dado que existe la derivación de *narrowing* necesario, $t_1 \overset{\theta_n\circ\cdots\circ\theta_2}{\leadsto_{NN}}{}^{*}$ s, con menos de n pasos, por hipótesis de induccción existe la derivación de *narrowing* perezoso uniforme $t_1 \overset{\theta_n\circ\cdots\circ\theta_2}{\leadsto_{ULN}}{}^{*}\ s$. Ahora, podemos formar la derivación de *narrowing* perezoso uniforme:
$$\tau_1(t) \overset{[p_1,R_1,\sigma_1]}{\leadsto_{ULN}} t_1 \overset{\theta_n\circ\cdots\circ\theta_2}{\leadsto_{ULN}}{}^{*}\ s,$$
y por el Corolario 7, existe la derivación de *narrowing* perezoso uniforme
$t \overset{\theta_n\circ\cdots\circ\theta_2\circ\theta_1}{\leadsto_{ULN}}{}^{*}\ s$.

La prueba en el sentido opuesto puede realizarse fácilmente ya que el razonamiento empleado es completamente reversible.

2. El punto (2) del enunciado de este teorema es una consecuencia inmediata de la Proposición 6.

Corollary 8. *Sea $\mathcal{R}$ un programa uniforme. Sea e una ecuación. Existe una derivación de* narrowing *necesario $e \overset{\sigma}{\leadsto}_{NN}^{*}$ true si y sólo si existe una derivación de* narrowing *perezoso uniforme $e \overset{\sigma}{\leadsto}_{ULN}^{*}$ true.*

Proof. Sean t_1 y t_2 términos y $e \equiv t_1 \approx t_2$. Por la Definición 3, existe una derivación de *narrowing* necesario $e \overset{\sigma}{\leadsto}_{NN}^{*}$ *true* si y sólo si existen las derivaciones de *narrowing* necesario $t_1 \overset{\sigma}{\leadsto}_{NN}^{*}\ s$ y $t_2 \overset{\sigma}{\leadsto}_{NN}^{*}\ s$, donde s es un término constructor básico. Por el Teorema 4, existen las derivaciones de *narrowing* necesario $t_1 \overset{\sigma}{\leadsto}_{NN}^{*}\ s$ y $t_2 \overset{\sigma}{\leadsto}_{NN}^{*}\ s$, si y sólo si existen las derivaciones *narrowing* perezoso uniforme $t_1 \overset{\sigma}{\leadsto}_{ULN}^{*}\ s$ y $t_2 \overset{\sigma}{\leadsto}_{ULN}^{*}\ s$. Finalmente, de nuevo por la Definición 3, existen las derivaciones de *narrowing* perezoso uniforme $t_1 \overset{\sigma}{\leadsto}_{ULN}^{*}\ s$ y $t_2 \overset{\sigma}{\leadsto}_{ULN}^{*}\ s$ si y sólo si existe una derivación *narrowing* perezoso uniforme $e \overset{\sigma}{\leadsto}_{ULN}^{*}$ *true*.

Theorem 5. *Sea $\mathcal{R}$ un programa uniforme y e una ecuación.*

1. *(Corrección) Si* $e \overset{\sigma}{\rightsquigarrow}_{ULN}^{*}$ *true es una derivación de* narrowing *perezoso uniforme, entonces σ es una solución para e.*
2. *(Completitud) Para cada substitución constructora σ que es una solución de e, existe una derivación de* narrowing *perezoso uniforme* $e \overset{\sigma'}{\rightsquigarrow}_{ULN}^{*}$ *true con* $\sigma' \leq \sigma \; [\mathcal{V}ar(e)]$.

Proof. Inmediato, por el Corolario 8 y el Teorema 2.

6 Evaluación Parcial de Programas Uniformes.

En esta sección formulamos una instancia del marco de evaluación parcial definido en [4] y estudiamos algunas de las ventajas y problemas que plantea.

6.1 Evaluación Parcial de Programas Lógico Funcionales.

En esta sección recordamos nuestro procedimiento genérico para la evaluación parcial de programas lógico funcionales. Las definiciones de este apartado son adaptaciones inmediatas de las que aparecen en [4, 5].

Definition 18 (Resultante). *Sea s un término y $\mathcal{R}$ un programa. Dada una derivación de* narrowing *que respeta la estrategia φ,* $s \overset{\sigma}{\rightsquigarrow}_{\varphi}^{*} t$, *su* resultante *asociada es la regla de reescritura $\sigma(s) \to t$.*

La intuición que se esconde detrás del concepto de resultante queda patente en el siguiente ejemplo.

Example 21. Sea el programa

$$\begin{array}{l} R_1 : f(X) \;\to\; (g(X) \approx 0 \;\Rightarrow\; X) \\ R_2 : g(0) \;\to\; 0 \\ R_3 : 0 \approx 0 \;\to\; true \\ R_4 : (true \;\Rightarrow\; X) \;\to\; X \end{array}$$

Dada la derivación de *narrowing* (perezoso):

$$\begin{array}{ll} \underline{f(x)} & \overset{[\Lambda,R_1,id]}{\rightsquigarrow_{\varphi}} \; (\underline{g(x)} \approx 0 \;\Rightarrow\; x) \\ & \overset{[1.1,R_2,\{X/0\}]}{\rightsquigarrow_{\varphi}} \; (\underline{0 \approx 0} \;\Rightarrow\; 0) \\ & \overset{[1,R_3,id]}{\rightsquigarrow_{\varphi}} \; (\underline{true \;\Rightarrow\; 0}) \\ & \overset{[\Lambda,R_4,id]}{\rightsquigarrow_{\varphi}} \; 0 \end{array}$$

Intuitivamente, el término $f(X)$ se ha reducido a 0 al aplicar la substitución $\{X/0\}$, asi que la resultante asociada a esta derivación es: $\{X/0\}(f(X)) \;\to\; 0$ (i.e., $f(0) \;\to\; 0$). Podemos interpretar que la resultante comprime una serie de pasos de *narrowing* convirtiendolos en una regla de programa, que cuando se emplea para evaluar el término $f(X)$ lo evalua a 0 en un único paso de *narrowing*.

Una vez comprendido el concepto de resultante es fácil definir que entendemos por PE de un programa con respecto a un término s. La idea básica consiste en la construcción de un árbol de búsqueda incompleto para el término inicial s, del que extraemos las resultantes asociadas con las derivaciones que parten desde la raíz s a las hojas que no son de fallo.

Definition 19 (Evaluación Parcial). *Sea $\mathcal{R}$ un programa y s un término. Sea τ un árbol de* narrowing *(posiblemente incompleto) para s en $\mathcal{R}$. Sea $\{t_1, \ldots, t_n\}$ los términos en las hojas de τ. Entonces, el conjunto de resultantes para las derivaciones de* narrowing $\{s \overset{\sigma_i}{\leadsto}_{\varphi}^{+} t_i \mid i = 1, \ldots, n\}$[7] *se denomina una* evaluación parcial *de s en $\mathcal{R}$.*

La evaluación parcial de un conjunto de términos S en $\mathcal{R}$ se define como la unión de las evaluaciones parciales para los términos de S.

La PE de un conjunto de términos se considera módulo renombramientos. También, en ocasiones, la PE de un conjunto de términos S en $\mathcal{R}$ la denominaremos evaluación parcial de $\mathcal{R}$ con respecto a S.

Como fue establecido por Lloyd y Shepherdson en [32], se necesita una condición de *cierre* ("closedness") para asegurar la completitud del proceso de PE. Nosotros extendemos la noción plana de cierre de la decucción parcial, introduciendo una condición de cierre recursiva que garantiza que todas las llamadas que puedan ocurrir durante la ejecución del *programa residual* (resultado de la PE) son cubiertas por alguna regla de éste.

Definition 20 (Cierre). *Sea S un conjunto de términos y t un término. Decimos que el término t es S-cerrado si se cumple $closed(S,t)$, donde el predicado closed se define inductivamente como sigue:*

$$closed(S,t) \Leftrightarrow \begin{cases} true & si\ t \in \mathcal{X} \\ closed(S,t_1) \wedge \ldots \wedge closed(S,t_n) & si\ t \equiv c(t_1, \ldots, t_n) \\ (\exists s \in S)\ \theta(s) = t \wedge \bigwedge\limits_{x/t' \in \theta} closed(S,t') & si\ t \equiv f(t_1, \ldots, t_n) \end{cases}$$

donde $c \in \mathcal{C}$, $f \in (\mathcal{D} \cup \mathcal{P})$. Decimos que un conjunto de términos T es S-cerrado, en símbolos $closed(S,T)$, si se cumple $closed(S,t)$ para todo $t \in T$, y decimos que un programa $\mathcal{R}$ es S-cerrado si se cumple $closed(S, \mathcal{R}_{calls})$. Aquí $\mathcal{R}_{calls}$ denota el conjunto de términos en las rhs's de las reglas de $\mathcal{R}$.

Example 22. Sea t el término $f(g(0))$ y S el conjunto $\{f(X), g(X)\}$. Entonces, t es S-cerrado ya que:

1. $(\exists\sigma)\sigma = \{X/g(0)\}$. $t = \sigma(f(X))$, i.e. el propio término t está cubierto por $f(X)$.
2. $(\exists\theta)\theta = \{X/0\}$. $g(0) = \theta(g(X))$, i.e. $g(0)$ está cubierto por $g(X)$.
3. y 0 is constructor, que por definición está cubierto.

[7] "$\leadsto_{\varphi}{}^{+}$" representa la clausura transitiva de la relación "$\leadsto_{\varphi}$".

Ahora introducimos la condición de *independencia*. La condición de independencia garantiza que dos términos especializados diferentes no serán confundidos y que el programa residual $\mathcal{R}'$ no produce respuestas adicionales. En el caso de la deducción parcial de la programación lógica, sólamente se comprueba la no unificabilidad (dos a dos) de los átomos evaluados parcialmente de S para asegurar la condición de independencia y garantizar la corrección fuerte de la transformación. En nuestro caso necesitamos una condición más compleja, más concretamente, necesitamos considerar los posibles *solapamientos* ("overlaps") entre las llamadas especializadas (ver [5]).

Definition 21 (Solapamiento). *Un término s* solapa *con un término t si existe un subtérmino no variable $s_{|u}$ de s tal que $s_{|u}$ y t unifican. Si $s \equiv t$, requerimos que t sea unificable con un subtérmino propio no variable de s.*

Definition 22 (Independencia). *Un conjunto de términos S es* independiente *si no existen términos s y t en S tal que s solapa con t.*

En lo que sigue, formalizamos un algoritmo genérico para la PE de programas lógico funcionales basado en el *narrowing* y que asegura que el conjunto de términos evaluados parcialmente es cerrado. Nuestro algoritmo es genérico con respecto a:

1. la *estrategia de narrowing* que construye los árboles de búsqueda locales;
2. la regla de desplegado (*unfolding rule*) que determina cuando y como parar la construcción de los árboles locales (atendiendo a algún criterio de parada);
3. el *operador de abstracción* empleado para garantizar la terminación global del proceso de PE.

Debido al empleo del *narrowing* en el desplegado de los árboles de búsqueda locales, hemos convenido en denominar a nuestro método evaluación parcial dirigida por *narrowing* (NPE) (del inglés, *Narrowing-driven Partial Evaluation*).

La regla de desplegado determina las expresiones con las que se dará un paso de *narrowing* (respetando una estrategia φ) y asegura que el proceso de desplegado termina devolviendo un conjunto finito de resultantes. Esto es, la regla de desplegado garantiza la denominada *terminación local* del proceso de NPE.

Definition 23 (Regla de Desplegado). *Una* regla de desplegado $U_\varphi(s, \mathcal{R})$ *(o simplemente U_φ) es una aplicación que devuelve una PE para el término s en el programa $\mathcal{R}$ utilizando la relación de* narrowing $\leadsto_\varphi$ *(i.e., un conjunto de resultantes).*

Dado un conjunto de términos S, denotamos por $U_\varphi(S, \mathcal{R})$ la unión de los conjuntos $U_\varphi(s, \mathcal{R})$, para s en S.

Por lo tanto, $U_\varphi(S, \mathcal{R}))$ denota una *evaluación parcial de S en $\mathcal{R}$ utilizando U_φ.*

El *operador de abstracción* garantiza la terminación del proceso de NPE alcanzando la condición de cierre y asegura la finitud del conjunto de términos evaluados parcialmente junto con el adecuado grado de especialización (se denomina *polivarianza* a esta habilidad de producir varias definiciones especializadas a partir de una única definición original).

Definition 24 (Operador de Abstracción). *Dado un conjunto finito de términos T y un conjunto de términos S (que forman la actual configuración de términos evaluados parcialmente), un* operador de abstracción *es una función que devuelve un conjunto finito de términos $abstract(S,T)$ tal que:*

1. *si $s \in abstract(S,T)$, entonces existe un $t \in (S \cup T)$ tal que $t|_p = \theta(s)$ para alguna posición p y substitución θ;*
2. *para todo $t \in (S \cup T)$, t es cerrado con respecto al conjunto de términos en $abstract(S,T)$.*

Intuitivamente, la primera condición garantiza que el operador de abstracción no introduce nuevos símbolos de función que no aparezcan en el conjunto de entrada S y en T, mientras que la segunda condición asegura que el conjunto de términos resultante "cubre" las llamadas previamente especializadas y que la condición de cierre se preserva a traves de las sucesivas operaciones de abstracción.

Nuestro algoritmo genérico para la NPE está parametrizado por la regla de desplegado U_φ y el operador de abstracción *abstract* en el estilo de [15].

Algoritmo 2
Entrada: *un programa $\mathcal{R}$ y un conjunto de términos T*
Salida: *un conjunto de términos S*
Initialización: $i := 0$; $T_0 := abstract(\varnothing, T)$
Repetir
    1. $\mathcal{R}' := U_\varphi(T_i, \mathcal{R})$;
    2. $T_{i+1} := abstract(T_i, \mathcal{R}'_{calls})$;
    3. $i := i + 1$;
Hasta que $T_i = T_{i-1}$ *(módulo renombramientos)*
Devolver $S := T_i$

De forma semejante a [34], aplicando *abstract* en cada iteración del Algoritmo 2, ajustamos el control de la polivarianza tanto como sea necesario.

Notad que la salida del algoritmo de NPE, para un programa $\mathcal{R}$, no es una evaluación parcial, sino un conjunto de términos S a partir del cual $U_\varphi(S, \mathcal{R})$ determina de forma no ambigüa el programa evaluado parcialmente $\mathcal{R}'$ en $\mathcal{R}$ empleando U_φ.

6.2 Un Evaluador Parcial Basado en Narrowing Perezoso Uniforme.

En esta sección presentamos una instancia del Algoritmo 2 y estudiamos su conveniencia, así como sus problemas asociados.

La instancia que vamos a proponer se justifica advirtiendo una de las dificultades [7] que presenta la evaluación parcial de programas cuando empleamos la estrategia de *narrowing* perezoso: la perdidad de la ortogonalidad del programa original. Como muestra el siguiente ejemplo, la evaluación parcial de programas, utilizando la estrategia de *narrowing* perezoso para el desplegado de los árboles locales, puede conducir a programas que no son siquiera ortogonales, aún en el caso de partir de programas uniformes.

Example 23. Sea $\mathcal{R}$ el programa uniforme del Ejemplo 12. La evaluación parcial de $\mathcal{R}$ con respecto al término $f(g(X), g(a), g(b))$, usando una regla de desplegado de un paso, conduce al siguiente resultado[8]:

El programa especializado $\mathcal{R}' = U_{\lambda_{lazy}}(S, \mathcal{R})$ es

$$\begin{array}{ll}
\{f(g(X), g(a), g(b)) & \to f(g(X), a, g(b)) \\
f(g(X), g(a), g(b)) & \to f(g(X), g(a), b) \\
f(g(X), a, g(b)) & \to f(g(X), a, b) \\
f(g(X), g(a), b) & \to f(g(X), a, b) \\
f(g(X), a, b) & \to 1\}.
\end{array}$$

que tras realizarse el postproceso de renombramiento, de acuerdo al siguiente renombramiento independiente

$$\begin{array}{ll}
S = \{ & f(g(X), g(a), g(b)) \mapsto h(X), \\
& f(g(X), a, g(b)) \mapsto h_1(X), \\
& f(g(X), g(a), b) \mapsto h_2(X), \\
& f(g(X), a, b) \mapsto h_3(X)\},
\end{array}$$

produce el programa renombrado $\mathcal{R}''$

$$\begin{array}{ll}
\{h(X) & \to h_1(X) \\
h(X) & \to h_2(X) \\
h_1(X) & \to h_3(X) \\
h_2(X) & \to h_3(X) \\
h_3(X) & \to 1\}.
\end{array}$$

que no es ortogonal ya que no cumple la condición de no ambigüedad[9]. Evidentemente, el programa derivado tampoco es uniforme.

El resultado obtenido en el ejemplo anterior es consecuencia de que la estrategia de *narrowing* perezoso computa derivaciones redundantes, en las que el orden de las reglas simplemente se intercambia cuando se aplican sobre d-iferentes redexes perezosos. De esta forma, derivaciones parciales que finalmente computarían el mismo resultado con la misma respuesta, dan lugar a distintas resultantes. Este ejemplo muestra la necesidad de emplear durante el proceso de PE una estrategia de *narrowing* perezoso refinada, como la estrategia de *narrowing* perezoso uniforme, que no presenta este tipo de problemas para programas uniformes.

[8] Después de la tercera iteración, el conjunto de términos evaluados parcialmente es: $S = \{f(g(X), g(a), g(b)),\ f(g(X), a, g(b)),\ f(g(X), g(a), b),\ f(g(X), a, b)\}$.

[9] Notad que tampoco cumple la condición de no ambigüedad débil

Example 24. Consideremos de nuevo el programa uniforme $\mathcal{R}$ del Ejemplo 12. La evaluación parcial de $\mathcal{R}$ con respecto al término $f(g(X), g(a), g(b))$, usando también una regla de desplegado de un paso, pero la estrategia de *narrowing* perezoso uniforme, en lugar de la estrategia de *narrowing* perezoso, produce el siguiente programa especializado $\mathcal{R}''$

$$\begin{aligned}&\{h(X) \to h_1(X)\\ &h_1(X) \to h_3(X)\\ &h_3(X) \to 1\}.\end{aligned}$$

que es un programa uniforme.

A la luz de los dos últimos ejemplos, proponemos como instancia concreta la consiste en elegir la estrategia de *narrowing* perezoso uniforme, anteriormente caracterizada, y un criterio de parada basado en el orden de "embedding" [44] para la definición de la regla de desplegado. Como operador de abstracción elegimos el operador concreto definido en [5]. Para preservar la completitud del proceso de PE nos vemos obligados a no desplegar los árboles locales de *narrowing* más alla de un término en hnf. Definimos una hnf-PE $\mathcal{R}'$ de un término s en $\mathcal{R}$ como una PE de s en $\mathcal{R}$ tal que cada derivación utilizada para construir el programa residual $\mathcal{R}'$, $s \overset{\sigma}{\leadsto}_{ULN}^{*} s_i \overset{\sigma'}{\leadsto}_{ULN}^{*} s'$, cumple que ningún s_i en hnf ha sido narroweado. En lo que sigue nos referiremos a la instancia concreta de nuestro algoritmo de NPE con el acrónimo ULN-PE (del inglés, *Uniform Lazy Narrowing-driven Partial Evaluation*), para distinguirla de la instancia estudiada en [2] a la que denominaremos LN-PE (del inglés, *Lazy Narrowing-driven Partial Evaluation*).

Como estudiamos en [2], en ocasiones, cuando realizamos una LN-PE de un programa puede perderse la disciplina de constructores en las lhs's de las reglas del programa transformado, lo que impide que la estrategia de *narrowing* perezoso pueda emplearse en la evaluación de términos con el programa transformado. Se necesita un postproceso de renombramiento tanto para recuperar la disciplina de constructores del programa original y asegurar que un término podra ejecutarse en el programa transformado, como para lograr la condición de independencia, es decir que no habrá interferencias entre las reglas cuando se ejecute el programa transformado. Lograr la independencia del conjunto de términos parcialmente evaluados es vital para la corrección de la LN-PE.

Para programas uniformes, además del postproceso de renombramiento se necesita un postproceso adicional, como pone de relieve el siguiente ejemplo, en el que se muestra como la uniformidad del programa especializado puede perderse debido al anidamiento de constructores en las lhs de las reglas.

Example 25. Sea el programa uniforme

$$\begin{aligned}\mathcal{R} = \{\ &R_1 : f(c(X)) \to g(X)\\ &R_2 : g(a) \qquad \to a\}\end{aligned}$$

Para el término $t \equiv f(X)$, sólamente existe la siguiente derivación a un término constructor en cabeza, cuando se empleando la estrategia de *narrowing* perezoso uniforme:

$$f(x) \overset{[\Lambda, R_1, \{X/c(Y)\}]}{\leadsto_{ULN}} g(Y) \overset{[\Lambda, R_2, \{Y/a\}]}{\leadsto_{ULN}} a,$$

que conduce al programa especializado

$$\mathcal{R}' = \{\ R_1' : f(c(a)) \to a\},$$

después de aplicar una iteración del algoritmo de evaluación parcial. Este programa $\mathcal{R}'$ (que coincide con el programa renombrado $\mathcal{R}''$) no es un programa uniforme, ya que incumple la primera restricción de la Definición 15 de programa uniforme.

Adviertase que este mismo problema también afecta a la estrategia de *narrowing* necesario, ya para el programa $\mathcal{R}$ del Ejemplo 25 la estrategia de *narrowing* necesario conduce al mismo programa especializado $\mathcal{R}'$

Recuperar la uniformidad del programa original es importante si queremos ejecutar el programa transformado utilizando *narrowing* perezoso uniforme, lo que es indispensable para asegurar la corrección del proceso de ULN-PE. A continuación proponemos una fase de postproceso adicional consistente en utilizar el siguiente algoritmo para lograr el aplanamiento de las lhs's de las reglas del programa evaluado parcialmente.

Algoritmo 3 (Aplanamiento, F)
Entrada*: Un programa $\mathcal{R}$.*
Salida*: Un programa en forma plana $F(\mathcal{R})$.*
$F(\mathcal{R}) =$ `Si` *las lhs's de las reglas de* $\mathcal{R}$ *son planas* `entonces` $\mathcal{R}$
`sino Sea` $R \equiv (f(t_1, \ldots, t_n) \to [E \Rightarrow]r) \in \mathcal{R}$ *tal que*
$(\exists i, j, m, : 1 \leq i \leq n, 1 \leq j \leq m)(t_i = c(s_1, \ldots, s_m) \wedge r \notin \mathcal{X})$;
`En` $F(\mathcal{R}')$
`Donde`
$$\begin{aligned} \mathcal{R}' = \ & (\mathcal{R} \setminus \{R\}) \cup \\ & \{f(t_1, \ldots, t_{i-1}, c(z_1, \ldots, z_m), t_{i+1}, \ldots, t_n) \to \\ & (z_1 \approx s_1 \wedge \ldots \wedge z_m \approx s_m[\wedge E] \Rightarrow r)\}. \end{aligned}$$
Las variables $z_1, \ldots, z_m$ son variables frescas y E es una condición, posiblemente vacía.

Example 26. Volviendo al Ejemplo 25, si aplicamos el Algoritmo 3 al programa especializado

$$\mathcal{R}' = \{\ R_1' : f(c(a)) \to a\},$$

obtenemos el programa

$$\mathcal{R}'' = F(\mathcal{R}') = \{\ R_1'' : f(c(X)) \to (X = a \Rightarrow a)\},$$

para el que se ha recuperado la restricción de patrones constructores planos y por lo tanto la uniformidad del programa original.

Hacemos notar que un algoritmo de aplanamiento como el que aparece en [29], en este ejemplo, restablecería la uniformidad al precio de perder toda la especialización conseguida en el proceso de ULN-PE, ya que obtendríamos nuevamente el programa original.

Concluimos este apartado demostrando la corrección y completitud fuerte de la intancia ULN-PE. Para establecer este resultado necesitamos un lema previo, que formalizamos de la siguiente manera.

Lemma 8. *Sean $\mathcal{R}$ un programa uniforme y R una regla de $\mathcal{R}$. Sean t y s términos. Si $t \overset{\sigma}{\rightsquigarrow}_{ULN}^{*} s$ entonces existe una selección de árboles definicionales de las funciones definidas en $\mathcal{R}$, tales que $t \overset{\sigma}{\rightsquigarrow}_{NN}^{*} s$.*

Proof. Inmediata, por inducción sobre el número de pasos de la derivación y haciendo uso del Corolario 5.

Theorem 6 (strong soundness and completeness).
Sea $\mathcal{R}$ un programa uniforme, e una ecuación, y S un conjunto finito de términos. Sea $\mathcal{R}'$ una hnf-PE de $\mathcal{R}$ con respecto a S tal que $\mathcal{R}' \cup \{e\}$ es S-cerrado. Sea S' un renombramiento independiente de S, $\mathcal{R}''$ un renombramiento y aplanamiento de $\mathcal{R}'$ w.r.t. S', y $e'' = ren(e, S')$. Entonces θ es una respuesta computada para e en $\mathcal{R}$ si y sólo si θ es una respuesta computada para e'' en $\mathcal{R}''$.

Proof. Por el Lema 8 si $\mathcal{R}''$ es una ULN-PE es también una NN-PE. Ahora por la corrección y completitud fuerte de la NN-PE [7], se sigue el resultado.

7 Conclusiones.

En este informe hemos realizado un estudio exhaustivo de la relación existente entre la estrategia de *narrowing* necesario [12] y la estrategia de *narrowing* perezoso presentada en [2], que en esencia es una formalización de la aparece en [29] sin recurrir a un *look ahead* de las posiciones demandadas. Algunas de las contribuciones originales de este trabajo se resumen en los siguientes puntos:

– Hemos caracterizado formalmente los programas uniformes como una subclase de los inductivamente secuenciales (Proposición 4).
– Para programas uniformes simples, las estrategias de *narrowing* perezoso y de *narrowing* necesario son equivalentes paso a paso (Proposición 5).
– Hemos dado una definición operacional del concepto de substitución adelantada en un paso de *narrowing* necesario.
– Hemos precisado la relación existente entre los pasos de *narrowing* necesario y los pasos de *narrowing* perezoso, para programas uniformes (Proposición 4 y Proposición 7).
– Hemos caracterizado una nueva estrategia de evaluación perezosa para programas uniformes, que hemos denominado *narrowing* perezoso uniforme.
– Hemos probado la equivalencia de las estrategias de *narrowing* perezoso uniforme y de *narrowing* necesario para programas uniformes (Teorema 4).

– Hemos demostrado que la estrategia de *narrowing* perezoso uniforme es correcta y completa para programas uniformes (Teorema 5).

Estos resultados tienen aplicación inmediata a la optimización de las técnicas de evaluación parcial. En [7], se demuestra que la evaluación parcial basada en *narrowing* necesario (NN-PE) hereda las buenas propiedades del mecanismo de base (e.g., preserva la estructura inductiva de los programas, las computaciones deterministas, las propiedades de optimalidad, etc). Por otro lado, la LN-PE puede aplicarse a una clase más amplia de programas, si bien presenta varios inconvenientes (e.g., se obtienen reglas redundantes en el programa especializado y se pierde la ortogonalidad del programa original). Los resultados de esta investigación caracterizan los programas uniformes como la clase más amplia de programas sobre la que pueden obtenerse ventajas comparables a las de la NN-PE sin la necesidad de usar estructuras como los árboles definicionales que utiliza el *narrowing* necesario para guiar la ejecución. La representación explícita de los árboles definicionales tiene un coste elevado para la eficiencia de la ejecución de los programas así como para el consumo de memoria. La idea es evitar el problema que introduce la gestión de los árboles definicionales obteniendo todavía las ventajas de la PE basada en *narrowing* necesario, haciendo uso de la estrategia de *narrowing* perezoso. Las buenas propiedades de los programas uniformes nos han permitido:

– definir una nueva instancia de nuestro marco general para PE de programas lógico funcionales: ULN-PE; y
– probar que la ULN-PE es fuertemente correcta y completa y preserva la uniformidad del programa original (tras un postproceso).

Nuestra intención es seguir profundizando en esta línea de investigación, estudiando la mejora práctica que los resultados obtenidos suponen para las técnicas de evaluación parcial de programas integrados perezosos.

References

1. M. Alpuente, S. Escobar, and S. Lucas. UPV-Curry: An Incremental Curry Interpreter with Polymorphic Types and Monadic I/O. Technical Report DSIC-II/12/99, UPV, 1999. Available from URL: `http://www.dsic.upv.es/users/elp/papers.html`.
2. M. Alpuente, M. Falaschi, P. Julián, and G. Vidal. Specialization of Lazy Functional Logic Programs. In *Proc. of the ACM SIGPLAN Conf. on Partial Evaluation and Semantics-Based Program Manipulation, PEPM'97*, volume 32, 12 of *Sigplan Notices*, pages 151–162, New York, 1997. ACM Press.
3. 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.
4. M. Alpuente, M. Falaschi, and G. Vidal. Partial Evaluation of Functional Logic Programs. *ACM Transactions on Programming Languages and Systems*, 20(4):768–844, 1998.

5. M. Alpuente, M. Falaschi, and G. Vidal. Partial Evaluation of Functional Logic Programs. Technical Report DSIC-II/11/98, UPV, 1998.
6. M. Alpuente, M. Hanus, S. Lucas, and G. Vidal. Specialization of Functional Logic Programs Based on Needed Narrowing. Technical report DSIC-II/7/99, DSIC, 1999. Available from URL: `http://www.dsic.upv.es/users/elp/papers.html`.
7. M. Alpuente, M. Hanus, S. Lucas, and G. Vidal. Specialization of Inductively Functional Logic Programs. In *Proc. of ICFP'99, Paris (France)*. ACM, New York, 1999. (To appear).
8. S. Antoy. Definitional trees. In *Proc. of the 3rd Int'l Conference on Algebraic and Logic Programming, ALP'92*, volume 632 of *Lecture Notes in Computer Science*, pages 143–157. Springer-Verlag, Berlin, 1992.
9. S. Antoy. Needed Narrowing in Prolog. Technical Report TR 96-2, Portland State University, 1996.
10. S. Antoy. Optimal non-deterministic functional logic computations. In *Proc. of the Int'l Conference on Algebraic and Logic Programming, ALP'97*, volume 1298 of *Lecture Notes in Computer Science*, pages 16–30. Springer-Verlag, Berlin, 1997.
11. S. Antoy, R. Echahed, and M. Hanus. A Needed Narrowing Strategy. Technical report MPI-I-93-243, Max-Planck-Institut für Informatik, Saarbrücken, 1993.
12. 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.
13. R. Caballero-Roldán, F.J. López-Fraguas, and J. Sánchez-Hernández. User's manual for Toy. Technical Report SIP-5797, UCM, Madrid (Spain), April 1997.
14. 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.
15. 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.
16. E. Giovannetti, G. Levi, C. Moiso, and C. Palamidessi. Kernel Leaf: A Logic plus Functional Language. *Journal of Computer and System Sciences*, 42:363–377, 1991.
17. M. Hamada and A. Middeldorp. Strong Completeness of a Lazy Conditional Narrowing Calculus. In *Proc. of the Second Fuji International Workshop on Functional and Logic Programming, Shonan Village*, 1996. To appear.
18. M. Hanus. The Integration of Functions into Logic Programming: From Theory to Practice. *Journal of Logic Programming*, 19&20:583–628, 1994.
19. M. Hanus, H. Kuchen, and J.J. Moreno-Navarro. Curry: A Truly Functional Logic Language. In *Proc. ILPS'95 Workshop on Visions for the Future of Logic Programming*, pages 95–107, 1995.
20. M. Hanus, S. Lucas, and A. Middeldorp. Strongly sequential and inductively sequential term rewriting systems. *Information Processing Letters*, 67(1):1–8, 1998.
21. S. Hölldobler. *Foundations of Equational Logic Programming*. Springer LNAI 353, 1989.
22. G. Huet. Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems. *Journal of the ACM*, 27(4):797–821, 1980.
23. G. Huet and J.J. Lévy. Computations in orthogonal rewriting systems, Part I + II. In J.L. Lassez and G.D. Plotkin, editors, *Computational Logic – Essays in Honor of Alan Robinson*, pages 395–443, 1992.

24. J.A. Jiménez-Martin, J. Mariño-Carballo, and J.J. Moreno-Navarro. Efficient Compilation of Lazy Narrowing into Prolog. In *Proc. of LOPSTR'92*, Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1992.
25. N.D. Jones, C.K. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice-Hall, Englewood Cliffs, NJ, 1993.
26. 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.
27. J.W. Klop and A. Middeldorp. Sequentiality in Orthogonal Term Rewriting Systems. *Journal of Symbolic Computation*, pages 161–195, 1991.
28. 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.
29. 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*, volume 463 of *Lecture Notes in Computer Science*, pages 298–317, 1990.
30. H. Kuchen, F.J. López-Fraguas, J.J. Moreno-Navarro, and M. Rodríguez-Artalejo. Implementing a Functional Logic Language with Disequality Constrains. In K. Apt, editor, *Proc. of the Joint International Conference and Symposium on Logic Programming, JICSLP'93*, pages 207–221. The MIT Press, Cambridge, MA, 1993.
31. 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.
32. J.W. Lloyd and J.C. Shepherdson. Partial Evaluation in Logic Programming. *Journal of Logic Programming*, 11:217–242, 1991.
33. 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.
34. B. Martens and J. Gallagher. Ensuring Global Termination of Partial Deduction while Allowing Flexible Polyvariance. In L. Sterling, editor, *Proc. of ICLP'95*, pages 597–611. MIT Press, 1995.
35. A. Middeldorp and E. Hamoen. Completeness Results for Basic Narrowing. *Applicable Algebra in Engineering, Communication and Computing*, 5:213–253, 1994.
36. A. Middeldorp and S. Okui. A Deterministic Lazy Narrowing Calculus. In *Proc. of the Fuji Int'l Workshop on Functional and Logic Programming*, pages 104–118. World Scientific, 1995.
37. A. Middeldorp, S. Okui, and T. Ida. Lazy Narrowing: Strong Completeness and Eager Variable Elimination. *Theoretical Computer Science*, 167(1,2):95–130, 1996.
38. J.J. Moreno-Navarro. Expressivity of Functional-Logic Languages and their Implementation. In R. Barbuti I. Ramos, M. Alpuente, editor, *Tutorials of the 1994 Joint Conf. on Declarative Programming, GULP-PRODE94*, pages 11–42. Servicio de Publicaciones de la Universidad Politécnica de Valencia, SPUPV-94.2049, 1994.
39. 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.
40. J.J. Moreno-Navarro and M. Rodríguez-Artalejo. Logic Programming with Functions and Predicates: The language Babel. *Journal of Logic Programming*, 12(3):191–224, 1992.

41. P. Padawitz. *Computing in Horn Clause Theories*, volume 16 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, Berlin, 1988.
42. 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.
43. J.R. Slagle. Automated Theorem-Proving for Theories with Simplifiers, Commutativity and Associativity. *Journal of the ACM*, 21(4):622–642, 1974.
44. M.H. Sørensen and R. Glück. An Algorithm of Generalization in Positive Supercompilation. In J.W. Lloyd, editor, *Proc. of ILPS'95*, pages 465–479. The MIT Press, Cambridge, MA, 1995.
45. F. Zartmann. Denotational Abstract Interpretation of Functional Logic Programs. In P. Van Hentenryck, editor, *Proc. of the 4th Int'l Static Analysis Symposium, SAS'97*, pages 141–159. Springer LNCS 1302, 1997.