# Cálculo Lambda de Primer Orden con Constraints y la Propiedad Church-Rosser

Luis Mandel, Martin Wirsing*
Institut für Informatik
Ludwig-Maximilians-Universität
Leopoldstr. 11b, 80802 München, Germany
Tel: +49 89 2180 6317
Fax: +49 89 2180 6310
{mandel,wirsing}@informatik.uni-muenchen.de

John N. Crossley†
Dept. of Computer Science
Monash University
Clayton, VIC 3168, Australia
Tel: +61 3 9905 5200
Fax: +61 3 9905 5146
jnc@bruce.cs.monash.edu.au

**Resumen**

En [2, 3] fue presentada una extensión al tradicional cálculo lambda con constraints. Los constraints pueden ser usados con dos distintos propósitos: en una forma pasiva, restringiendo el rango de variables, o en forma activa computando soluciones a determinados sistemas. Aquí presentamos una extensión de aquél cálculo, agregándole cuantificadores existenciales de modo de enfatizar las teorías de Henkin y para eliminar el problema de las variables compartidas. También definimos nuevas reglas para la manipulación de los nuevos términos del lenguaje. La semántica denotacional del cálculo es presentada, así como también la demostración de la propiedad Church-Rosser (CR). Además probamos que las reglas de reducción son correctas y que la función semántica está bien definida.

**Palabras Clave:** Cálculo $\lambda$, Constraints, propiedad Church-Rosser, semántica denotacional.

## Introducción

El paradigma de programación con constraints se basa en la idea de manipulación de objetos usando la información parcial disponible. La primer propuesta que usó aproximaciones iterativas para resolver programas con constraints fue Sketchpad [15] en 1963. En 1987 Jaffar y Lassez introdujeron Constraint Logic Programming (CLP). Esta nueva propuesta presentó una forma general para la incorporación de constraints en la programación lógica. Esto fue implementado en el lenguaje CLP($\mathbb{R}$) [8] que contiene una versión del algoritmo simplex para chequear la consistencia de constraints sobre los números reales.

La idea de combinar constraints y programación funcional fue introducida por Darlington et al. en [6]. FALCON [6] es un lenguaje de programación lógico-funcional con constraints.

*Trabajo parcialmente financiado por el proyecto DFG-Constraints.
†Trabajo parcialmente financiado por el acuerdo de investigación con IBM nro. 15760046

Este lenguaje extiende la programación lógica y funcional proveyendo la capacidad de resolver ecuaciones y permitiendo el uso de constraints para programar así como para consultas.

En trabajos previos ([2, 3, 9]) fue presentado el "propositional constrained $\lambda$-calculus". Este cálculo es una extensión del tradicional cálculo $\lambda$ sin tipos. El resultado de una reducción de un término puede ser otro término con constraints. De esta forma "información incompleta" puede ser manipulada. Los constraints pueden ser usados de dos maneras distintas. Por un lado, en forma pasiva, restringiendo el universo de validez de un término. Pero el otro, inferencias provenientes de los constraints causan el reemplazo de subtérminos en el término objetivo. Este es el así denominado uso activo de constraints. En [11, 12] la semántica denotacional de aquel cálculo fue definida y la correctitud de las reglas de reducción fue demostrada.

El objetivo del presente trabajo es desarrollar una nueva extensión al trabajo anterior con el agregado explícito de cuantificadores existenciales a las reglas de formación de términos. El cuantificador existencial $\exists$ liga variables cuyo valor será computado usando la teoría subyacente de constraints mediante un resolvedor de constraints mientras que las variables bajo el alcance del abstractor $\lambda$ serán computadas en la forma tradicional, esto es, usando la $\beta$ reducción. Las variables cuantificadas existencialmente (también llamadas variables lógicas) no pueden ser sombreadas por el abstractor $\lambda$, y viceversa. En la sección 1 se presenta la notación del lenguaje de constraints, las reglas de formación de términos y algunos ejemplos. En la sección 2 se presenta la semántica denotacional así como la demostración de algunos teoremas relativos a esta semántica. En la sección 3 la propiedad CR del cálculo es demostrada y finalmente conclusiones y líneas futuras de investigación son dadas.

# 1 First-Order Constrained Lambda Calculus

Un lenguaje de constraints está formado por *palabras* sobre el siguiente alfabeto:

- $\pounds = \langle\, \mathcal{K}, \mathcal{V}, \mathcal{F}, \mathcal{P} \,\rangle$ un lenguaje de constraints donde:
  - $\mathcal{K} = \{k_1, \ldots\}$ es un conjunto numerable de *constantes*,
  - $\mathcal{V} = \{x_1, \ldots\}$ es un conjunto numerable de *variables*,
  - $\mathcal{F} = \{f_1^{n_1}, \ldots\}$ es un conjunto numerable de *letras de función*,
  - $\mathcal{P} = \{P_1^{m_1}, \ldots\}$ es un conjunto numerable y no vacío de *letras de predicado*;
- $\Pi = \langle \pounds, \mathcal{A}, \mathcal{R}, \vdash_\pi \rangle$ es un resolvedor de constraints, donde:
  - $\mathcal{A}$ es un conjunto de fórmulas bien formadas sobre $\pounds$,
  - $\mathcal{R}$ es un conjunto de reglas de inferencia, y
  - $\vdash_\pi$ es una relación de inferencia con axiomas $\mathcal{A}$ y reglas $\mathcal{R}$;
- $\lambda$ es el abstractor y $\exists$ denota el cuantificador existencial;
- $\{,\ \},\ (,\ )$ son símbolos auxiliares.

Las reglas de formación de términos que presentamos a continuación son una extensión a las reglas del cálculo $\lambda_c$ definidas en [2, 3] con el agregado del cuantificador existencial $\exists$. Esta extensión es denotada $\Lambda_c^{\exists}$; también escribimos $\Lambda_c^{\exists} = \Lambda_c + \exists$. El conjunto de constraints es denotado Constraint.

**Definición 1.1 (Lambda Términos con Constraints, Constraint)**

(I)
- $\mathsf{fail} \in \Lambda_c^{\exists}$; (Fail)
- $k \in \mathcal{K} \Rightarrow k \in \Lambda_c^{\exists}$; (Constant)
- $x \in \mathcal{V} \Rightarrow x \in \Lambda_c^{\exists}$; (Var)
- $f^n \in \mathcal{F},\ \mathrm{M}_1, \ldots, \mathrm{M}_n \in \Lambda_c^{\exists} \Rightarrow f^n(\mathrm{M}_1, \ldots, \mathrm{M}_n) \in \Lambda_c^{\exists}$; (Functors)
- $x \in \mathcal{V}, \mathrm{M} \in \Lambda_c^{\exists} \Rightarrow (\lambda\, x\, .\, \mathrm{M}) \in \Lambda_c^{\exists}$; (Abstr)
- $\mathrm{M}, \mathrm{N} \in \Lambda_c^{\exists} \Rightarrow (\mathrm{M}\ \mathrm{N}) \in \Lambda_c^{\exists}$; (Applic)
- $C \in \mathsf{Constraint}, \mathrm{M} \in \Lambda_c^{\exists} \Rightarrow (\{C\}\mathrm{M}) \in \Lambda_c^{\exists}$. (Constr)
- $\mathrm{M} \in \Lambda_c^{\exists} \Rightarrow (\exists\, x\, .\, \mathrm{M}) \in \Lambda_c^{\exists}$. (Exist)

(II)
- $P^m \in \mathcal{P}, \mathrm{M}_1 \cdots \mathrm{M}_m \in \Lambda_c^{\exists} \Rightarrow \{P^m(\mathrm{M}_1, \cdots, \mathrm{M}_m)\} \in \mathsf{Constraint}$;
- $\mathrm{M}_1, \mathrm{M}_2 \in \Lambda_c^{\exists} \Rightarrow \{\mathrm{M}_1 = \mathrm{M}_2\} \in \mathsf{Constraint}$;
- $\{C\}, \{D\} \in \mathsf{Constraint} \Rightarrow \{C, D\} \in \mathsf{Constraint}$.

□

Para simplificar la notación, y además de las abreviaturas standard, términos de la forma $\exists\, x_1\, .\, (\exists\, x_2\, .\, \mathrm{M})$ serán escritos $\exists\, x_1\, x_2\, .\, \mathrm{M}$. Esto es, los términos:

$$(\exists\, x_1\, .\, (\exists\, x_2\, .\, \mathrm{M})) \equiv (\exists\, x_1\, x_2\, .\, \mathrm{M}) \equiv (\exists\, x_2\, x_1\, .\, \mathrm{M}) \equiv (\exists\, x_2\, .\, (\exists\, x_1\, .\, \mathrm{M}))$$

son equivalentes. Estas equivalencias (así como la regla $\alpha$) son a nivel sintáctico. Más aún, si los términos $\mathrm{M}_1, \ldots, \mathrm{M}_n$ ocurren en un cierto contexto, entonces variables libres y ligadas que ocurren en esos términos tienen distintos nombres.

**Definición 1.2 (Alcance, Variables Libres y Ligadas)** Dado un término $(\lambda\, x\, .\, \mathrm{M})$ el *alcance* del abstractor $\lambda$ es M. Dado un término $(\exists\, x\, .\, \mathrm{N})$ el *alcance* del cuantificador existencial $\exists$ es N. Una ocurrencia de una variable se dice *libre* en un término si no ocurre bajo el alcance del abstractor $\lambda$ ni ocurre bajo el alcance del cuantificador $\exists$. Toda ocurrencia *libre* de una variable $x$ en M es una ocurrencia *ligada* en $(\lambda\, x\, .\, \mathrm{M})$ y toda ocurrencia *libre* de una variable $x$ en N es una ocurrencia *ligada* en $(\exists\, x\, .\, \mathrm{N})$. □

## 1.1 Reglas de Reducción

Como en [2, 3], sólo consideremos soluciones únicas. La regla clave, que introduce reducciones implicadas por la teoría de constraints, es la siguiente:

($\exists$) si $\mathcal{A} \vdash (\exists!\ x\ .\ C) \wedge (\forall\ x\ .\ C \Rightarrow x = \mathrm{N})$ y N es una forma normal minimal entonces $\exists\ x\ .\ \{C\}\mathrm{M} \longrightarrow_{\exists} (\{C\}\mathrm{M})[x := \mathrm{N}]$

donde $\mathcal{A}$ es el conjunto de axiomas del resolvedor de constraints. Nótese que las variables libres de N deben satisfacer la convención de variables, esto es, variables libres y ligadas tienen nombres distintos.

Esta regla es motivada por las teorías de Henkin. Estas son las teorías $\mathcal{T}$ para las que dada una sentencia de la forma $(\exists\ x\ .\ \mathrm{M})$, existe una constante $k$ en el lenguaje tal que: $\mathcal{T} \models (\exists\ x\ .\ \mathrm{M}) \supset (\mathrm{M}[x := k])$

Esta reducción presenta similitudes con la regla $\beta$ como se ve en la figura 1.1.

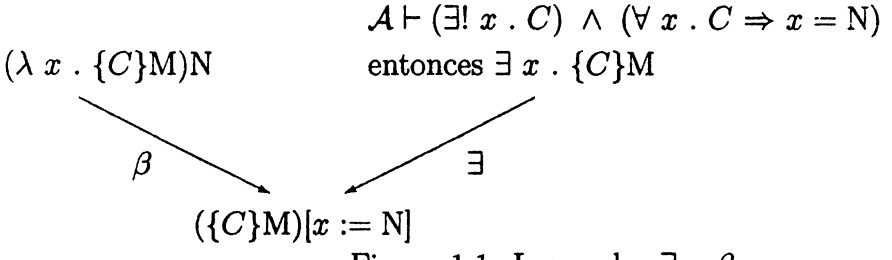



Figura 1.1: Las reglas $\exists$ y $\beta$

Para la $\beta$ reducción, la sustitución es inducida por la presencia de un parámetro formal en la abstracción. En el caso de la reducción $\exists$, el valor N para $x$ es calculado por el resolvedor de constraints como el único valor válido para $x$ en la teoría de constraints bajo las condiciones en $\{C\}$.

Resumiendo, las reglas del cálculo $\lambda_c^{\exists}$ son las siguientes:

($\beta$) $(\lambda\ x\ .\ \mathrm{M})\ \mathrm{N} \longrightarrow_{\beta} \mathrm{M}[x := \mathrm{N}]$;
($\exists$) $\exists\ x\ .\ \{C\}\mathrm{M} \longrightarrow_{\exists} (\{C\}\mathrm{M})[x := \mathrm{N}]$ si $\mathcal{A} \vdash (\exists!\ x\ .\ C) \wedge (\forall\ x\ .\ C \Rightarrow x = \mathrm{N})$;
($A_c$) $\{C\}(\{D\}\mathrm{M}) \longrightarrow_{A_c} \{C, D\}\mathrm{M}$;
($P_c$) $\{D, P(\ldots, \{C_i\}\mathrm{M}_i, \ldots)\}\mathrm{N} \longrightarrow_{P_c} \{D, C_i, P(\ldots, \mathrm{M}_i, \ldots)\}\mathrm{N}$;
($F_c$) $f(\ldots, \{C_i\}\mathrm{M}_i, \ldots) \longrightarrow_{F_c} \{C_i\} f(\ldots, \mathrm{M}_i, \ldots)$;
($F_{\exists}$) $f(\ldots, \exists\ x_i\ .\ \mathrm{M}_i, \ldots) \longrightarrow_{F_{\exists}} \exists\ x_i\ .\ f(\ldots, \mathrm{M}_i, \ldots)$;
($P_{\exists}$) $\{D, P(\ldots, \exists\ x_i\ .\ \mathrm{M}_i, \ldots)\}\mathrm{N} \longrightarrow_{P_{\exists}} \exists\ x_i\ .\ \{D, P(\ldots, \mathrm{M}_i, \ldots)\}\mathrm{N}$;
($C_{\exists}$) $\{C\}(\exists\ x\ .\ \mathrm{M}) \longrightarrow_{C_{\exists}} \exists\ x\ .\ (\{C\}\mathrm{M})$;
(fail) la regla fail es la unión de las siguientes cinco reglas:
($\mathtt{f}_1$) $(\mathsf{fail}\ \mathrm{M}) \longrightarrow_{\mathtt{f}_1} \mathsf{fail}$;
($\mathtt{f}_2$) $(\lambda\ x\ .\ \mathsf{fail}) \longrightarrow_{\mathtt{f}_2} \mathsf{fail}$;
($\mathtt{f}_3$) $\{C\}\mathrm{M} \longrightarrow_{\mathtt{f}_3} \mathsf{fail}$ si $\{C\}$ es *inconsistente*;
($\mathtt{f}_4$) $\{C\}\mathsf{fail} \longrightarrow_{\mathtt{f}_4} \mathsf{fail}$;
($\mathtt{f}_5$) $\exists\ x\ .\ \mathsf{fail} \longrightarrow_{\mathtt{f}_5} \mathsf{fail}$.

**Ejemplo 1.3 (Los Exploradores Perdidos)** *Un avión lanza un paquete de emergencia*

*a unos exploradores perdidos. Si el avión vuela horizontalmente a 40 metros por segundo a una altitud de 100 metros sobre el nivel del suelo, a qué distancia relativa al punto donde fue lanzado alcanza el paquete el suelo?*
Sabemos que:

- $V_{x_0} = 40m/s$ es la velocidad horizontal inicial y
- $V_{y_0} = 0m/s$ es la velocidad vertical inicial;
- $t$ es una *incógnita* que representa el tiempo durante el que el paquete está en el aire.

Las coordenadas horizontal y vertical satisfacen las siguientes ecuaciones:

$$x = V_{x_0} \cdot t = 40m/s \cdot t, \qquad y = -\frac{1}{2} \cdot G \cdot t^2 = -\frac{1}{2} \cdot (9.81m/s^2) \cdot t^2 = -100m.$$

Por lo tanto, podemos construir el siguiente $\lambda_c^{\exists}$ término:

$$(\lambda\ V_{x_0}\ y\ .\ (\exists\ t\ .\ \{y = -\tfrac{1}{2} \cdot 9.81 \cdot t^2, t > 0\} V_{x_0} \cdot t))40\ -100$$
$$\longrightarrow_{\beta} \exists\ t\ .\ \{-100 = -\tfrac{1}{2} \cdot 9.81 \cdot t^2, t > 0\} 40 \cdot t \equiv \mathrm{M}$$

Asumiendo que el resolvedor de constraints resuelve el sistema en $t$, tal que:

$$\{-100 = -\tfrac{1}{2} \cdot 9.81 \cdot t^2, t > 0\} \vdash_{\pi} t = 4.52$$

entonces, aplicando la regla $\exists$ obtenemos:

$$\mathrm{M} \longrightarrow_{\exists} \{-100 = -\tfrac{1}{2} \cdot 9.81 \cdot 20.39, 4.5152 > 0\}\ 180.61$$

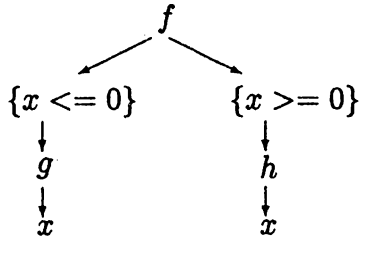
y de este término se puede concluir que el paquete alcanza el suelo a 180.61 metros del punto en que fue lanzado. □

# 2 Semántica

Aquí damos la semántica denotacional del lenguaje. La semántica para un esquema general de un lenguaje lógico-funcional de primer orden con constraints fue presentada en [10]. También en [14] fue presentada una semántica de lenguages de constraints concurrentes.

Esta semántica está intuitivamente motivada por el siguiente ejemplo. Consideremos el término

$$f(\{x <= 0\}g(x), \{x >= 0\}h(x)) \longrightarrow\!\!\!\!\rightarrow_{A_c} \{x <= 0, x >= 0\}f(g(x), h(x))$$



La información contenida en los constraints es global y afecta al término entero. Esta es la razón por la cual la función semántica tiene, además del término en cuestión, no sólo un entorno con valores para las variables sino también un constraint global como argumento. Usaremos reticulados completos (lattices) como dominios, donde $\bot$ y $\top$ son el menor y mayor elemento, respectivamente, de cada dominio. **Env** es el dominio de los entornos, **V** el de los valores, **F** el de las funciones y **C** es el de valores *compatibles* compuesto por tres puntos ordenados de la siguiente manera: $\bot_{\mathbf{C}} \sqsubset \mathbf{Comp} \sqsubset \top_{\mathbf{C}}$.

$\mathcal{CLE}$ : Constrained Lambda Expression $\rightarrow$ (**Env** $\times$ Constraint) $\rightarrow$ $\mathbf{V} \oplus \mathbf{F}$
$\mathcal{CON}$: Constraint $\rightarrow$ (**Env** $\times$ Constraint) $\rightarrow$ **C**
$\mathcal{AC}$: Atomic Constraint $\rightarrow$ (**Env** $\times$ Constraint) $\rightarrow$ **C**

$$\begin{array}{lcl} \mathbf{F} & \cong & \mathbf{V} \oplus \mathbf{F} \rightarrow \mathbf{V} \oplus \mathbf{F} \\ \mathbf{Env} & = & \mathcal{V} \rightarrow \mathbf{V} \oplus \mathbf{F} \\ \mathbf{V} & = & \mathcal{U}_Q \cup \{\top_{\mathbf{V}}, \bot_{\mathbf{V}}\} \text{ ordenado por } \bot_{\mathbf{V}} \sqsubset v \sqsubset \top_{\mathbf{V}} \text{ para todo } v \text{ en } \mathcal{U}_Q, \\ & & \text{donde } \mathcal{U}_Q \text{ es el dominio de constraints subyacente.} \end{array}$$

Para determinar la semántica de un término M, debemos "recolectar" la información de los constraints contenida en M. Esto es hecho por la función *collect*, que permite la manipulación de los efectos colaterales que un constraint tiene sobre los términos que acompaña.

Debido a la existencia de variables ligadas, debemos duplicar el conjunto de variables $\mathcal{V}$. Más aún, dado que una variable puede estar ligada más de una vez en un término (e.g. $\lambda\, x\, .\, ((\lambda\, x\, .\, x)x)$), toda variable duplicada tendrá un índice que será actualizado usando una función *shift*.

**Definición 2.1 (La Función Shift)** La función **shift** es inductivamente definida como sigue: **shift** : $(\mathcal{V} \times (\lambda_c\text{-term} \cup \text{Constraint})) \rightarrow \lambda_c\text{-term}$

- $\mathbf{shift}(x, \mathsf{fail}) = \mathsf{fail}$;
- $\mathbf{shift}(x, k) = k$;
- $\mathbf{shift}(x, y) = \begin{cases} x^1 & \text{if } x \equiv y, \\ x^{n+1} & \text{if } \exists\ n : y \equiv x^n, \\ y & \text{en otro caso;} \end{cases}$
- $\mathbf{shift}(x, \text{m-ary-func}(\lambda_c\text{-term}_1, \ldots, \lambda_c\text{-term}_m)) =$ $\text{m-ary-func}(\mathbf{shift}(x, \lambda_c\text{-term}_1), \ldots, \mathbf{shift}(x, \lambda_c\text{-term}_m))$;
- $\mathbf{shift}(x, \lambda\ \text{var}\ .\ \lambda_c\text{-term}) = \begin{cases} \lambda\ x\ .\ \lambda_c\text{-term} & \text{si } x \equiv \text{var}, \\ \lambda\ \text{var}\ .\ \mathbf{shift}(x, \lambda_c\text{-term}) & \text{en otro caso;} \end{cases}$
- $\mathbf{shift}(x, (\lambda_c\text{-term}_1\ \lambda_c\text{-term}_2)) = (\mathbf{shift}(x, \lambda_c\text{-term}_1)\ \mathbf{shift}(x, \lambda_c\text{-term}_2))$;
- $\mathbf{shift}(x, \{C\}\lambda_c\text{-term})) = \mathbf{shift}(x, \{C\})\mathbf{shift}(x, \lambda_c\text{-term})$;
- $\mathbf{shift}(x, \{\text{ac}_1, \ldots, \text{ac}_k\}) = \{\mathbf{shift}(x, \text{ac}_1), \ldots, \mathbf{shift}(x, \text{ac}_k)\}$;
- $\mathbf{shift}(x, \text{n-ary-pred}(\lambda_c\text{-term}_1, \ldots, \lambda_c\text{-term}_n)) =$ $\text{n-ary-pred}(\mathbf{shift}(x, \lambda_c\text{-term}_1), \ldots, \mathbf{shift}(x, \lambda_c\text{-term}_n))$;
- $\mathbf{shift}(x, \lambda_c\text{-term}_1 = \lambda_c\text{-term}_2) = \mathbf{shift}(x, \lambda_c\text{-term}_1) = \mathbf{shift}(x, \lambda_c\text{-term}_2)$. □

**Definición 2.2 (La Función Collect Function)** La función **collect** esta definida sobre términos y constraints retornando un constraint. **collect** : $\lambda_c$-term $\rightarrow$ Constraint

- **collect**(fail) $= \{\mathbf{F}\}$;
- **collect**(constant) = **collect**(var) $= \emptyset$;
- **collect**(m-ary-func($\lambda_c$-term$_1, \ldots, \lambda_c$-term$_m$)) $= \bigcup_{i=1}^{m}$ **collect**($\lambda_c$-term$_i$);
- **collect**($\lambda$ var . $\lambda_c$-term) = **shift**(var, **collect**($\lambda_c$-term));
- **collect**($\lambda_c$-term$_1$ $\lambda_c$-term$_2$) = **collect**($\lambda_c$-term$_1$) $\cup$ **collect**($\lambda_c$-term$_2$);
- **collect**($\{C\}\lambda_c$-term) = **collect**($\{C\}$) $\cup$ **collect**($\lambda_c$-term);
- **collect**($\{c_1, \ldots, c_n\}$) $= \bigcup_{i=1}^{n}$ **collect**($c_i$);
- **collect**($\{\lambda_c$-term$_1 = \lambda_c$-term$_2\}$) = $\{\lambda_c$-term$_1 = \lambda_c$-term$_2\} \cup$ **collect**($\lambda_c$-term$_1$) $\cup$ **collect**($\lambda_c$-term$_2$);
- **collect**($\{$n-ary-pred($\lambda_c$-term$_1, \ldots, \lambda_c$-term$_n$)$\}$) = $\{$n-ary-pred($\lambda_c$-term$_1, \ldots, \lambda_c$-term$_n$)$\} \cup \bigcup_{i=1}^{n}$ **collect**($\lambda_c$-term$_i$). □

**Ejemplo 2.3**

**collect**$((\lambda\ x\ .\ \{x \times y = 0\}y)y) =$
**collect**$(\lambda\ x\ .\ \{x \times y = 0\}y) \cup$ **collect**$(y) =$
**shift**$(x,$ **collect**$(\{x \times y = 0\}y)) =$
**shift**$(x,$ **collect**$(\{x \times y = 0\} \cup$ **collect**$(y))) =$
**shift**$(x, \{x \times y = 0\}) =$
$\{$**shift**$(x, x) \times$ **shift**$(x, y) =$ **shift**$(x, 0)\} = \{x^1 \times y = 0\}$ □

Las denotaciones son las siguientes:

$\mathcal{CLE}[\![\text{fail}]\!] = \lambda\ e \in \mathbf{Env}, c \in \text{Constraint}\ .\ \top_{\mathbf{V \oplus F}}$

$\mathcal{CLE}[\![\text{constant}]\!] = \lambda\ e \in \mathbf{Env}, c \in \text{Constraint}\ .\ \mathbf{in}_1(\mathcal{I}_Q(\text{constant}))$
donde $\mathcal{I}_Q$ es la función de interpretación de constraints subyacente.

$\mathcal{CLE}[\![\text{var}]\!] = \lambda\ e \in \mathbf{Env}, c \in \text{Constraint}\ .\ \mathbf{unique}\ \{v : e\{\text{var} := v\} \models \exists\ \vec{y}\ .\ c_{\text{var}}\}$
donde $c_{\text{var}}$ es la unión de los constraints atómicos dependientes de la variable var, $\vec{y}$ son las variables libres de $c_{\text{var}}$ diferentes de var, $e\{\text{var} := v\} \models \exists\ \vec{y}\ .\ c_{\text{var}}$ abrevia "existen $v_i \in \mathbf{V \oplus F}$ tales que $e\{\text{var} := v\}\{y_i := v_i\} \models c_{\text{var}}$," y $e\{\text{var} := v\}$ es como sigue:

- $e\{\text{var} := v\}(y) = e(y)$ if var $\not\equiv y$;
- $e\{\text{var} := v\}(\text{var}) = \begin{cases} v & \text{si } e(\text{var}) = \bot_{\mathbf{V}} \text{ or } e(\text{var}) = v \\ e(\text{var}) & \text{en otro caso.} \end{cases}$

A continuación introducimos las denotaciones para términos que tienen que *compartir* sus valores con otros subtérminos.

$\mathcal{CLE}[\![$m-ary-func($\lambda_c$-term$_1, \ldots, \lambda_c$-term$_m$)$]\!] = \lambda\ e \in \mathbf{Env}, c \in \text{Constraint}\ .$
**let** *cons* $= c \cup$ **collect**(m-ary-func($\lambda_c$-term$_1, \ldots, \lambda_c$-term$_m$))
**in** $\mathcal{FUNC}[\![$m-ary-func$]\!]$ ($\mathcal{CLE}[\![\lambda_c$-term$_1]\!]$ *e cons*, $\ldots, \mathcal{CLE}[\![\lambda_c$-term$_m]\!]$ *e cons*)

$\mathcal{CLE}[\![(\lambda_c \text{ var } . \lambda_c\text{-term})]\!] = \lambda\, e \in \mathbf{Env}, c \in \text{Constraint}\ .$
$\mathbf{in}_2(\lambda\, v \in \mathbf{V} \oplus \mathbf{F}\ .\ \mathcal{CLE}[\![\lambda_c\text{-term}]\!]\, e[\text{var} := v]\ \ c[\text{var} := x_{new}])$
donde $x_{new}$ es una variable *nueva*.[1]

$\mathcal{CLE}[\![(\lambda_c\text{-term}_1\ \ \lambda_c\text{-term}_2)]\!] = \lambda\, e \in \mathbf{Env}, c \in \text{Constraint}\ .$
$\mathbf{let}\ \ cons_1 = \mathbf{collect}(\lambda_c\text{-term}_1)$
$\quad\ cons_2 = \mathbf{collect}(\lambda_c\text{-term}_2)$
$\mathbf{in}\ [\mathbf{id}_{\mathbf{F}}, \lambda\, x \in \mathbf{V}\ .\ \perp_{\mathbf{F}}](\mathcal{CLE}[\![\lambda_c\text{-term}_1]\!]\, e\ c \cup cons_2)(\mathcal{CLE}[\![\lambda_c\text{-term}_2]\!]\, e\ c \cup cons_1)$

$\mathcal{CLE}[\![(\text{constraint } \lambda_c\text{-term})]\!] = \lambda\, e \in \mathbf{Env}, c \in \text{Constraint}\ .$
$\mathbf{let}\ \ cons_1 = \mathbf{collect}(\text{constraint})$
$\quad\ cons_2 = \mathbf{collect}(\lambda_c\text{-term})$
$\mathbf{in}\ \ \mathbf{if}\ \mathcal{CON}[\![c \cup cons_1 \cup cons_2]\!]\, e \neq \emptyset$
$\quad\ \mathbf{then}\ \mathcal{CLE}[\![\lambda_c\text{-term}]\!]\, e\ c \cup cons_1$

$\mathcal{CLE}[\![(\exists \text{ var } . \lambda_c\text{-term})]\!]\, e\ c = \mathcal{CLE}[\![\lambda_c\text{-term}[\text{var} := x_{new}]]\!]\, e\ c$

Ahora definimos la semántica de un constraint teniendo en cuenta que términos $\lambda_c$ pueden aparecer *dentro* de un constraint. Por ejemplo, un constraint atómico que testea la igualdad de dos $\lambda$ abstracciones por medio de la igualdad $=_{\mathbf{V} \oplus \mathbf{F}}$. (En muchos de estos casos el resultado de tal comparación será $\perp_{\mathbf{C}}$.)

$$\begin{aligned}\mathcal{CON}[\![\{\text{constraint}\}]\!] = \mathcal{CON}[\![\{\ & \text{ac}_1(\lambda_c\text{-term}_1^1, \ldots, \lambda_c\text{-term}_{n_1}^1),\\ & \text{ac}_2(\lambda_c\text{-term}_1^2, \ldots, \lambda_c\text{-term}_{n_2}^2),\\ & \ldots,\\ & \text{ac}_k(\lambda_c\text{-term}_1^k, \ldots, \lambda_c\text{-term}_{n_k}^k)\ \}]\!] =\end{aligned}$$

$\lambda\, e \in \mathbf{Env}, c \in \text{Constraint}\ .$
$\mathbf{let}\ cons = c \cup \mathbf{collect}(\{\text{constraint}\})$
$\mathbf{in}\ \ \mathbf{and}\ (\ \mathcal{AC}[\![\text{ac}_1(\lambda_c\text{-term}_1^1, \lambda_c\text{-term}_2^1, \ldots, \lambda_c\text{-term}_{n_1}^1)]\!]\, e\ cons$
$\qquad\qquad \vdots$
$\qquad\qquad \mathcal{AC}[\![\text{ac}_k(\lambda_c\text{-term}_1^k, \lambda_c\text{-term}_2^k, \ldots, \lambda_c\text{-term}_{n_1}^k)]\!]\, e\ cons)$

$\mathcal{AC}[\![\text{n-ary-pred}(\lambda_c\text{-term}_1, \lambda_c\text{-term}_2, \ldots, \lambda_c\text{-term}_n)]\!] = \lambda\, e \in \mathbf{Env}, c \in \text{Constraint}\ .$
$\mathbf{let}\ cons = c \cup \mathbf{collect}(\{\text{n-ary-pred}(\lambda_c\text{-term}_1, \lambda_c\text{-term}_2, \ldots, \lambda_c\text{-term}_n)\})$
$\mathbf{in}\ \mathcal{PRED}[\![\text{n-ary-pred}]\!]\, (\mathcal{CLE}[\![\lambda_c\text{-term}_1]\!]\, e\ cons,\ \ldots,\ \mathcal{CLE}[\![\lambda_c\text{-term}_n]\!]\, e\ cons)$

$\mathcal{AC}[\![\lambda_c\text{-term}_1 = \lambda_c\text{-term}_2]\!] = \lambda\, e \in \mathbf{Env}, c \in \text{Constraint}\ .$
$\mathbf{let}\ cons = c \cup \mathbf{collect}(\{\lambda_c\text{-term}_1 = \lambda_c\text{-term}_2\})$
$\mathbf{in}\ (\mathcal{CLE}[\![\lambda_c\text{-term}_1]\!]\, e\ cons =_{\mathbf{V} \oplus \mathbf{F}} \mathcal{CLE}[\![\lambda_c\text{-term}_2]\!]\, e\ cons)$

$\mathcal{PRED}[\![\text{n-ary-pred}]\!] = \mathcal{I}_{\mathbf{V}}(\text{n-ary-pred}) \in (\mathbf{V} \oplus \mathbf{F})^n \rightarrow \mathbf{C}$
donde $\mathcal{I}_{\mathbf{V}}$ es la función $\mathcal{I}_Q$ de interpretación de constraints subyacente aumentada para tratar mínimo y máximo ($\perp_{\mathbf{V} \oplus \mathbf{F}}$ y $\top_{\mathbf{V} \oplus \mathbf{F}}$) en la forma obvia.

[1] Nótese que $x_{new}$ *nueva* implica $e(x_{new}) = \perp_{\mathbf{V} \oplus \mathbf{F}}$.

$\mathcal{FUNC}[\![\text{m-ary-func}]\!] = \mathcal{I}_{\mathbf{V}}(\text{m-ary-func}) \in (\mathbf{V} \oplus \mathbf{F})^m \to \mathbf{V} \oplus \mathbf{F}$

**Teorema 2.4** Las reglas de reducción preservan la semántica □

**Demostración:** Basta ver que si $M \to M'$, entonces $\mathcal{CLE}[\![M]\!] = \mathcal{CLE}[\![M']\!]$. □

**Teorema 2.5** La función semántica $\mathcal{CLE}$ es monótona. □

**Demostración:** Es suficiente con verificar que si $M \sqsubseteq M'$, entonces $\mathcal{CLE}[\![M]\!] \sqsubseteq \mathcal{CLE}[\![M']\!]$. □

# 3 La Propiedad Church-Rosser (CR)

La demostración de la propiedad CR está organizada de la siguiente manera. Primero demostramos que la relación $\Phi = (\beta \cup \exists)$ satisface la propiedad CR. Para esta prueba demostramos que la regla $\exists$ satisface la propiedad CR y que conmuta con la regla $\beta$. Dado que la regla $\beta$ es CR, según el lema de Hindley-Rosen (ver [1, página 64]) la relación $\Phi$ es también CR.

La demostración de la propiedad CR débil de la relación $\Upsilon = (A_c \cup \text{fail} \cup P_c \cup F_c \cup F_\exists \cup P_\exists \cup C_\exists)$ fue demostrada en [2, 3]. Básicamente todos los casos de solapamiento fueron analizados. También fue demostrado que la relación $\Upsilon$ termina y, siguiendo el lema de Newman [13], la relación $\Upsilon$ es CR.

Finalmente demostramos que la clausura reflexiva y transitiva de las relaciones $\Phi$ y $\Upsilon$ conmutan. Nuevamente según el lema de Hindley-Rosen, la relación $\Gamma$ definida como $\Gamma = (\Phi \cup \Upsilon)$ es CR. Esto es, el cálculo completo satisface la propiedad CR.

**Lema 3.1 (La Regla $\exists$ es CR)** Si $M \twoheadrightarrow_\exists M_1$ y $M \twoheadrightarrow_\exists M_2$, entonces existe $M_3$ tal que $M_1 \twoheadrightarrow_\exists M_3$ y $M_2 \twoheadrightarrow_\exists M_3$.

**Demostración:** Existe sólo un caso de solapamiento de la forma

$$M \equiv \exists\, x\, .\, \{C\}(\ldots \exists\, y\, .\, \{D\}M' \ldots)$$

donde $\mathcal{A} \vdash (\exists!\, x\, .\, C) \wedge (\forall\, x\, .\, C \Rightarrow x = N_1)$,
y $\mathcal{A} \vdash (\exists!\, y\, .\, D) \wedge (\forall\, y\, .\, D \Rightarrow y = N_2)$.
Sin pérdida de generalidad sea $M \equiv \exists\, x\, .\, \{C\}(\exists\, y\, .\, \{D\}M')$. Entonces:

$$\begin{array}{lll} M & \longrightarrow_\exists & (\{C\}(\exists\, y\, .\, \{D\}M'))[x := N_1] \\ & \equiv & \{C\}[x := N_1](\exists\, y\, .\, (\{D\}[x := N_1]M'[x := N_1])) \end{array}$$

$$\begin{array}{lll} & \longrightarrow_{\exists} & \{C\}[x := \mathrm{N}_1](\{D\}[x := \mathrm{N}_1]\mathrm{M}'[x := \mathrm{N}_1])[y := \mathrm{N}_2] \\ & \equiv & \{C\}[x := \mathrm{N}_1](\{D\}[x := \mathrm{N}_1][y := \mathrm{N}_2]\mathrm{M}'[x := \mathrm{N}_1][y := \mathrm{N}_2]) \equiv_{\mathrm{def}} \mathrm{M}_1 \end{array}$$

$$\begin{array}{lll} \mathrm{M} & \longrightarrow_{\exists} & \exists\, x\, .\, (\{C\}((\{D\}\mathrm{M}')[y := \mathrm{N}_2])) \\ & \equiv & \exists\, x\, .\, (\{C\}(\{D\}[y := \mathrm{N}_2]\mathrm{M}'[y := \mathrm{N}_2])) \\ & \longrightarrow_{\exists} & \{C\}(\{D\}[y := \mathrm{N}_2]\mathrm{M}'[y := \mathrm{N}_2])[x := \mathrm{N}_1] \\ & \equiv & \{C\}[x := \mathrm{N}_1](\{D\}[y := \mathrm{N}_2][x := \mathrm{N}_1]\mathrm{M}'[y := \mathrm{N}_2][x := \mathrm{N}_1]) \equiv_{\mathrm{def}} \mathrm{M}_2 \end{array}$$

La cuestión es bajo qué condiciones vale $\mathrm{M}_1 \equiv \mathrm{M}_2$. Aplicando el lema de sustitución (ver [1, 2.1.16]) en $\mathrm{M}_1$ tenemos:

$$\begin{array}{lll} \mathrm{M}_1 & \equiv & \{C\}[x := \mathrm{N}_1](\{D\}[x := \mathrm{N}_1][y := \mathrm{N}_2]\mathrm{M}'[x := \mathrm{N}_1][y := \mathrm{N}_2]) \\ & \equiv & \{C\}[x := \mathrm{N}_1](\{D\}[y := \mathrm{N}_2][x := \mathrm{N}_1]\mathrm{M}'[y := \mathrm{N}_2][x := \mathrm{N}_1]) \equiv \mathrm{M}_2 \end{array}$$

El primer paso de la equivalencia es correcto debido a la convención de variables, dado que $y \notin \mathrm{FV}(\mathrm{N}_1)$. Por ello la sustitución $[x := \mathrm{N}_1[y := \mathrm{N}_2]]$ es equivalente a $[x := \mathrm{N}_1]$. Luego la propiedad se cumple y la regla $\exists$ satisface la propiedad diamante. Según el lema [1, 3.2.2] la regla $\exists$ es CR. □

**Lema 3.2 (Las Reglas $\exists$ y $\beta$ Conmutan)** Si $\mathrm{M} \longrightarrow_{\exists} \mathrm{M}_1$ y $\mathrm{M} \longrightarrow_{\beta} \mathrm{M}_2$, entonces existe $\mathrm{M}_3$ tal que $\mathrm{M}_1 \longrightarrow_{\beta} \mathrm{M}_3$ y $\mathrm{M}_2 \longrightarrow_{\exists} \mathrm{M}_3$.

**Demostración:** Existen tres casos de solapamiento:

- $\mathrm{M} \equiv \exists\, y\, .\, \{C\}((\lambda\, x\, .\, \mathrm{N}_1)\mathrm{N}_2)$,
- $\mathrm{M} \equiv (\lambda\, x\, .\, (\exists\, y\, .\, (\{C\}\mathrm{N}_1)))\mathrm{N}_2\ y$
- $\mathrm{M} \equiv (\lambda\, x\, .\, \mathrm{N}_1)(\exists\, y\, .\, (\{C\}\mathrm{N}_2))$,

donde $\mathcal{A} \vdash (\exists!\, y\, .\, C) \wedge (\forall\, y\, .\, C \Rightarrow y = \mathrm{N}_3)$.

*Caso 1:*
$$\begin{array}{lll} \mathrm{M} & \longrightarrow_{\exists} & (\{C\}((\lambda\, x\, .\, \mathrm{N}_1)\mathrm{N}_2))[y := \mathrm{N}_3] \\ & \equiv & \{C\}[y := \mathrm{N}_3]((\lambda\, x\, .\, \mathrm{N}_1[y := \mathrm{N}_3])\mathrm{N}_2[y := \mathrm{N}_3]) \\ & \longrightarrow_{\beta} & \{C\}[y := \mathrm{N}_3]\mathrm{N}_1[y := \mathrm{N}_3][x := \mathrm{N}_2[y := \mathrm{N}_3]] \\ & \equiv & \{C\}[y := \mathrm{N}_3]\mathrm{N}_1[x := \mathrm{N}_2][y := \mathrm{N}_3] \equiv \mathrm{M}_3 \\ \mathrm{M} & \longrightarrow_{\beta} & \exists\, y\, .\, \{C\}\mathrm{N}_1[x := \mathrm{N}_2] \\ & \longrightarrow_{\exists} & \{C\}[y := \mathrm{N}_3]\mathrm{N}_1[x := \mathrm{N}_2][y := \mathrm{N}_3] \equiv \mathrm{M}_3 \end{array}$$

*Caso 2:*
$$\begin{array}{lll} \mathrm{M} & \longrightarrow_{\exists} & (\lambda\, x\, .\, (\{C\}\mathrm{N}_1)[y := \mathrm{N}_3])\mathrm{N}_2 \\ & \longrightarrow_{\beta} & (\{C\}\mathrm{N}_1)[y := \mathrm{N}_3][x := \mathrm{N}_2] \equiv \mathrm{M}_3 \\ \mathrm{M} & \longrightarrow_{\beta} & (\exists\, y\, .\, (\{C\}\mathrm{N}_1))[x := \mathrm{N}_2] \\ & \equiv & \exists\, y\, .\, ((\{C\}\mathrm{N}_1)[x := \mathrm{N}_2]) \\ & \longrightarrow_{\exists} & (\{C\}\mathrm{N}_1)[x := \mathrm{N}_2][y := \mathrm{N}_3] \\ & \equiv & (\{C\}\mathrm{N}_1)[y := \mathrm{N}_3][x := \mathrm{N}_2] \equiv \mathrm{M}_3 \end{array}$$

*Caso 3:* $\mathrm{M} \longrightarrow_{\exists} (\lambda\, x\, .\, \mathrm{N}_1)(\mathrm{N}_2[y := \mathrm{N}_3])$
$\longrightarrow_{\beta} \mathrm{N}_1[x := \mathrm{N}_2[y := \mathrm{N}_3]] \equiv \mathrm{M}_3$
$\mathrm{M} \longrightarrow_{\beta} \mathrm{N}_1[x := \exists\, y\, .\, (\{C\}\mathrm{N}_2)]$
$\longrightarrow_{\exists} \mathrm{N}_1[x := \mathrm{N}_2[y := \mathrm{N}_3]] \equiv \mathrm{M}_3$

En todos los casos las reducciones conmutan, luego la propiedad se cumple. □

A continuación demostramos que no existe una derivación infinita

$$\mathrm{M}_1 \longrightarrow_{\Upsilon} \mathrm{M}_2 \longrightarrow_{\Upsilon} \ldots,$$

es decir $\Upsilon$ termina. Siguiendo [4, página 270], definimos un orden de reducción $\mathcal{W}$ entre términos. La terminación de la regla $\Upsilon$ es asegurada porque cada par en la relación $\Upsilon$ está en el orden $\mathcal{W}$ (es decir, $(\mathrm{M}, \mathrm{N}) \in \Upsilon \Rightarrow (\mathrm{M}, \mathrm{N}) \in \mathcal{W}$).

**Definición 3.3 (Peso)** $\mathcal{W}$ es la función sobre términos $\lambda_c^{\exists}$ definida por:

- $\mathcal{W}(\mathsf{fail}) = \mathcal{W}(k) = \mathcal{W}(x) = 3$;
- $\mathcal{W}(\lambda\, x\, .\, \mathrm{M}) = \mathcal{W}(\exists\, x\, .\, \mathrm{M}) = 1 + \mathcal{W}(\mathrm{M})$;
- $\mathcal{W}(\{C\}\mathrm{M}) = 1 + \mathcal{W}(\{C\}) \cdot \mathcal{W}(\mathrm{M})$;
- $\mathcal{W}((\mathrm{M}\ \mathrm{N})) = \mathcal{W}(\mathrm{M}) + \mathcal{W}(\mathrm{N})$;
- $\mathcal{W}(f(\mathrm{M}_1, \ldots, \mathrm{M}_n)) = 2^{\Sigma\ \mathcal{W}(M_i)}$;
- $\mathcal{W}(\{c_1, \ldots, c_n\}) = \Sigma\ \mathcal{W}(c_i)$;
- $\mathcal{W}(P(\mathrm{M}_1, \ldots, \mathrm{M}_m)) = 2 \cdot \Sigma\ \mathcal{W}(\mathrm{M}_i)$; □

Claramente la función $\mathcal{W}$ es entera y positiva. Esto es, para cada $\mathrm{M} \in \Lambda_c^{\exists}$, $\mathcal{W}(\mathrm{M}) > 0$. Esta función induce un orden de reducción bien fundado.

**Lema 3.4 ($\Upsilon$ Termina)** La regla $\Upsilon$ termina.

**Demostración:** Aplicando la función $\mathcal{W}$ definida en 3.3 obtenemos que el *redex* de cada regla de la relación $\Upsilon$ es $\mathcal{W}$-mayor que su *contractum*. Por lo tanto, según [4, página 270] la regla $\Upsilon$ termina. □

**Teorema 3.5 (La Relación $\Upsilon$ es CR)** Si $\mathrm{M} \twoheadrightarrow_{\Upsilon} \mathrm{M}_1$ y $\mathrm{M} \twoheadrightarrow_{\Upsilon} \mathrm{M}_2$, entonces existe $\mathrm{M}_3$ tal que $\mathrm{M}_1 \twoheadrightarrow_{\Upsilon} \mathrm{M}_3$ y $\mathrm{M}_2 \twoheadrightarrow_{\Upsilon} \mathrm{M}_3$. □

**Demostración:** Según [3] la relación $\Upsilon$ satisface la propiedad CR débil. Según 3.4 la relación $\Upsilon$ termina. Siguiendo el lema de Newman (ver [13]), la relación $\Upsilon$ satisface la propiedad CR. □

**Teorema 3.6 (El Cálculo $\lambda_c^{\exists}$ es CR)** Si $\mathrm{M} \twoheadrightarrow_{\Gamma} \mathrm{M}_1$ y $\mathrm{M} \twoheadrightarrow_{\Gamma} \mathrm{M}_2$, entonces existe $\mathrm{M}_3$ tal que $\mathrm{M}_1 \twoheadrightarrow_{\Gamma} \mathrm{M}_3$ y $\mathrm{M}_2 \twoheadrightarrow_{\Gamma} \mathrm{M}_3$. □

**Demostración:** Hemos visto que ambas relaciones $\Phi$ y $\Upsilon$ satisfacen la propiedad CR. Analizando los casos de solapamineto se obtiene que ambas relaciones conmutan en un solo paso. Por el lema de Hindley-Rosen, todo el cálculo satisface la propiedad CR. □

# 4 Conclusiones y Perspectivas

Se ha implementado una máquina abstracta para el sistema donde los constraints están restrictos a ecuaciones lineales e inecuaciones. Para ello hemos desarrollado una versión del cálculo con sustituciones explícitas. Hemos especificado el cálculo $\lambda_c^{\exists}$ para los sistemas RAP y TIP (ver [5]) y esperamos poder probar la confluencia local usando estos demostradores de teoremas. Como resolvedor de constraints usamos un algoritmo simplex modificado como es descripto en [7]. El cálculo presentado es una extensión elegante del cálculo $\lambda$ tradicional que permite no sólo manipular información incompleta sino también agregar algunas características imperativas a la funcionalidad del cálculo $\lambda$.

En el futuro removeremos la restricción de unicidad de las soluciones de los constraints. Esto es, vamos a permitir soluciones tales como: $C \vdash_\pi x = \mathrm{N} \vee x = \mathrm{M}$. Este hecho sólo afecta la regla $\exists$, que será extendida para manipular más de un valor.

# Referencias


[1] H. P. Barendregt. *The Lambda Calculus. Its Syntax and Semantics.* North Holland, 2nd edition, 1984.

[2] J. N. Crossley, L. Mandel, and M. Wirsing. Una Extensión de Constraints al Cálculo Lambda. In *Proceedings of the Segundo Congreso de Programación Declarativa, ProDe.93*, Blanes, Girona Spain, September 29-30, October 1 1993. (In Spanish).

[3] J. N. Crossley, L. Mandel, and M. Wirsing. Untyped Constrained Lambda Calculus. Institut für Informatik, Number 9318, Ludwig-Maximilians-Universität München, Leopoldstraße 11b, 80802 München, Germany, October 1993. 48 pages.

[4] N. Dershowitz and J.-P. Jouannaud. Rewriting Systems. In J. V. Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 6, pages 244–320. Elsevier Science Publishers, 1984.

[5] U. Fraus. Inductive Theorem Proving for Algebraic Specifications – TIP System User's Manual. Technical Report MIP 9401, Universität Passau, 1994.

[6] Y. Guo and H. Pull. FALCON: Functional and Logic Language with Constraints (Language Definition) Technical Report IC/FPG/Phoenix/15/3, Imperial College, March 1991.

[7] P. V. Hentenryck and T. Graf. Standard Forms for Rational Linear Arithmetic in Constraint Logic Programming. Technical Report IR-LP-2217, European Computer-Industry Research Center, 1989.

[8] J. Jaffar, S. Michaylov, P. Stuckey, and R. Yap. The CLP(ℝ) Language and System. *ACM Transactions on Programming Languages and Systems*, pages 339–395, 1992.

[9] H. C. Lock, A. Mück, and T. Streicher. A tiny functional logic constraint language and its continuation semantics. In *ESOP '94, Edinburgh*, LNCS 788, pages 439–453, April 1994.

[10] F. J. López-Fraguas. A General Scheme for Constraint Functional Logic Programming. In H. Kirchner and G. Levi, editors, *Proceedings of the Algebraic and Logic Programming Conference, ALP-1992*, pages 213–227, October 28–31 1992.

[11] L. Mandel. The Semantics of the Untyped Constrained Lambda Calculus. Institut für Informatik, Number 9319, Ludwig-Maximilians-Universität München, Leopoldstraße 11b, 80802 München, Germany, October 1993.

[12] L. Mandel. *Constrained Lambda Calculus.* PhD thesis, Ludwig-Maximilians-Universität München, Leopoldstraße 11b, 80802 München, Germany, December 1995. 213 pages.

[13] M. H. A. Newman. On Theories with a Combinatorial Definition of "Equivalence". *Annals of Mathematics*, 43(2):223–243, 1942.

[14] P. Panangaden, V. Saraswat, and M. Rinard. Semantic Foundations of Concurrent Constraint Programming. In *Proceedings of the Eighteenth Annual ACM Symposium on Principles of Programming Languages*, 1991.

[15] I. Sutherland. SKETCHPAD: A Man-Machine Graphical Communication System. In *IFIPS Proceedings of the Sprint Joint Computer Conference*, Jan. 1963. See also his Ph.D thesis from M.I.T.

# 4 Conclusiones y Perspectivas

Se ha implementado una máquina abstracta para el sistema donde los constraints están restrictos a ecuaciones lineales e inecuaciones. Para ello hemos desarrollado una versión del cálculo con sustituciones explícitas. Hemos especificado el cálculo $\lambda_c^{\exists}$ para los sistemas RAP y TIP (ver [5]) y esperamos poder probar la confluencia local usando estos demostradores de teoremas. Como resolvedor de constraints usamos un algoritmo simplex modificado como es descripto en [7]. El cálculo presentado es una extensión elegante del cálculo $\lambda$ tradicional que permite no sólo manipular información incompleta sino también agregar algunas características imperativas a la funcionalidad del cálculo $\lambda$.

En el futuro removeremos la restricción de unicidad de las soluciones de los constraints. Esto es, vamos a permitir soluciones tales como: $C \vdash_\pi x = \mathrm{N} \vee x = \mathrm{M}$. Este hecho sólo afecta la regla $\exists$, que será extendida para manipular más de un valor.

# Referencias


[1] H. P. Barendregt. *The Lambda Calculus. Its Syntax and Semantics.* North Holland, 2nd edition, 1984.

[2] J. N. Crossley, L. Mandel, and M. Wirsing. Una Extensión de Constraints al Cálculo Lambda. In *Proceedings of the Segundo Congreso de Programación Declarativa, ProDe.93*, Blanes, Girona Spain, September 29-30, October 1 1993. (In Spanish).

[3] J. N. Crossley, L. Mandel, and M. Wirsing. Untyped Constrained Lambda Calculus. Institut für Informatik, Number 9318, Ludwig-Maximilians-Universität München, Leopoldstraße 11b, 80802 München, Germany, October 1993. 48 pages.

[4] N. Dershowitz and J.-P. Jouannaud. Rewriting Systems. In J. V. Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, chapter 6, pages 244–320. Elsevier Science Publishers, 1984.

[5] U. Fraus. Inductive Theorem Proving for Algebraic Specifications – TIP System User's Manual. Technical Report MIP 9401, Universität Passau, 1994.

[6] Y. Guo and H. Pull. FALCON: Functional and Logic Language with Constraints (Language Definition) Technical Report IC/FPG/Phoenix/15/3, Imperial College, March 1991.

[7] P. V. Hentenryck and T. Graf. Standard Forms for Rational Linear Arithmetic in Constraint Logic Programming. Technical Report IR-LP-2217, European Computer-Industry Research Center, 1989.

[8] J. Jaffar, S. Michaylov, P. Stuckey, and R. Yap. The CLP(ℝ) Language and System. *ACM Transactions on Programming Languages and Systems*, pages 339–395, 1992.

[9] H. C. Lock, A. Mück, and T. Streicher. A tiny functional logic constraint language and its continuation semantics. In *ESOP '94, Edinburgh*, LNCS 788, pages 439–453, April 1994.

[10] F. J. López-Fraguas. A General Scheme for Constraint Functional Logic Programming. In H. Kirchner and G. Levi, editors, *Proceedings of the Algebraic and Logic Programming Conference, ALP-1992*, pages 213–227, October 28–31 1992.

[11] L. Mandel. The Semantics of the Untyped Constrained Lambda Calculus. Institut für Informatik, Number 9319, Ludwig-Maximilians-Universität München, Leopoldstraße 11b, 80802 München, Germany, October 1993.

[12] L. Mandel. *Constrained Lambda Calculus.* PhD thesis, Ludwig-Maximilians-Universität München, Leopoldstraße 11b, 80802 München, Germany, December 1995. 213 pages.

[13] M. H. A. Newman. On Theories with a Combinatorial Definition of "Equivalence". *Annals of Mathematics*, 43(2):223–243, 1942.

[14] P. Panangaden, V. Saraswat, and M. Rinard. Semantic Foundations of Concurrent Constraint Programming. In *Proceedings of the Eighteenth Annual ACM Symposium on Principles of Programming Languages*, 1991.

[15] I. Sutherland. SKETCHPAD: A Man-Machine Graphical Communication System. In *IFIPS Proceedings of the Sprint Joint Computer Conference*, Jan. 1963. See also his Ph.D thesis from M.I.T.