Combinando Lógica con Patrones

Dr. Ariel Arbiser, Lic. Gabriela Steren.

DEPARTAMENTO DE COMPUTACION

Patrones y Cálculo Lambda

Correspondencia de patrones.- En lenguajes de programación funcional como Haskell, se usan patrones para armar programas del siguiente modo:

- `head (x:xs) = x`
- `tail (x:xs) = xs`
- `length [] = 0` `length (x:xs) = 1 + length xs`

Al aplicar una de estas funciones a un argumento, por ejemplo `head (1:2:[])`, la estructura del argumento `1:2:[]` se compara con la del patrón `x:xs` y, si coinciden, la función se evalúa reemplazando cada parte del patrón por la parte correspondiente del argumento.

Cálculo Lambda.- El Cálculo Lambda es un *sistema de reescritura* que permite definir funciones y sus aplicaciones. Este sistema provee una base formal para los lenguajes funcionales.
Los términos del Cálculo Lambda tienen la siguiente forma:

x	(variables)
MN	(aplicaciones, con M y N términos)
$\lambda x.M$	(abstracciones, con x una variable y M un término).

Las abstracción $\lambda x.M$ es una función con parámetro x y cuerpo M. La siguiente regla de reducción define el comportamiento de estas funciones:

$$(\lambda x.M)N \to M\{x \leftarrow N\} \text{ (sustitución de } x \text{ por } N \text{ en } M.)$$

Por ejemplo: $(\lambda x.x)y \to y$ $(\lambda x.z)y \to z$ $(\lambda x.xx)y \to yy$ $(\lambda x.xx)(\lambda x.xx) \to (\lambda x.xx)(\lambda x.xx) \to \dots$

Este cálculo es *confluente*: aun si hay distintas posibilidades de reducción (evaluación) - por ejemplo en $(\lambda x.x)((\lambda y.z)v)$ -, los distintos caminos siempre vuelven a encontrarse.

Cálculo Lambda con Patrones.- Existen distintas variantes del Cálculo Lambda que introducen la posibilidad de usar patrones para definir funciones. Una de las más simples es λP.
λP es similar al Cálculo Lambda original, pero con abstracciones de la forma $\lambda P.M$, donde P es un término llamado *patrón*.
La regla de reducción en λP es la siguiente:

$$(\lambda P.M)P^\sigma \to M^\sigma$$

siendo σ una sustitución cualquiera de variables por términos.
Por ejemplo: $(\lambda(\lambda x.x).z)(\lambda y.y) \to z$ pero $(\lambda(\lambda x.x).z)y$ no reduce porque y no es de la forma $\lambda x.x$.

λP definido de esta manera no es confluente, pero sí lo es si se restringen los patrones permitidos. Entre las restricciones que se usan para lograr la confluencia están *RPC* (patrones "rígidos") y RPC^+(una variante de *RPC* que se verifica sintácticamente).

Lógica Combinatoria

La Lógica Combinatoria fue desarrollada en la década de 1920 con el objetivo de eliminar las variables ligadas, presentes en la lógica clásica. Actualmente se utiliza como modelo teórico de la computación y como base para desarrollar lenguajes funcionales. A pesar de su simplicidad, captura muchos aspectos esenciales de la computación, como la posibilidad de expresar **todas las funciones computables**.
Hay distintas variantes, pero en todas ellas los términos pueden ser: variables, constantes especiales llamadas *combinadores*, y aplicaciones de un término a otro. Los combinadores representan funciones de alto orden. Los más comúnmente utilizados son S y K, con las siguientes reglas de reducción asociadas:

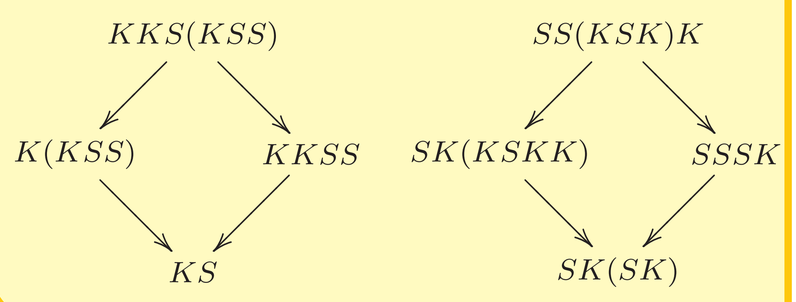
$$\begin{array}{lcl} KMN & \to & M \\ SMNO & \to & MO(NO) \end{array}$$

Por ejemplo: $SKKx \to Kx(Kx) \to x$.

Existe una correspondencia entre el Cálculo Lambda y la Lógica Combinatoria, pudiendo traducir los términos de un sistema al otro.
Por ejemplo:

$$\begin{array}{lcl} \lambda x.x & \rightsquigarrow & SKK \\ \lambda x.y & \rightsquigarrow & Ky \\ \lambda x.\lambda y.y & \rightsquigarrow & K(SKK) \\ \lambda x.\lambda y.yx & \rightsquigarrow & S(K(S(SKK)))K \\ \lambda x.xx & \rightsquigarrow & S(SKK)(SKK) \end{array}$$

Este cálculo es *confluente*. Ejemplos:



Resultados principales

El sistema CL_P.- Definimos una extensión de la Lógica Combinatoria SK, introduciendo combinadores decorados con patrones. Cada patrón debe contrastarse contra un argumento para que el término pueda reducirse.
Los nuevos combinadores son la forma: K_{P}, S_{P}, $\Pi^1_{\mathrm{PP'}}$, $\Pi^2_{\mathrm{PP'}}$, donde P y P' son patrones, y la aplicación PP' también es un patrón. A los combinadores K_{x} y S_{x} los llamamos K y S respectivamente.
Hay distintas formas de definir el conjunto de patrones. Si se permite usar cualquier término como patrón, el sistema no es confluente (tal como sucede en λP). El siguiente es un conjunto de patrones "seguros" que permite traducir (entre otros) todos los términos de λP que satisfacen RPC^+, manteniendo la confluencia.

$$P, P', P'' ::= \quad x \mid K_{\mathrm{P}} \mid S_{\mathrm{P}} \mid K_{\mathrm{P}}P' \mid S_{\mathrm{P}}P' \mid S_{\mathrm{P}}P'P'' \mid \Pi^1_{\mathrm{PP'}} \mid \Pi^2_{\mathrm{PP'}}$$

donde x es una variable, PP' (donde aparece) es un patrón, y ningún patrón tiene variables repetidas. Dos patrones se consideran iguales si difieren a lo sumo en los nombres de sus variables.

Las reglas de reducción de CL_P son las siguientes:

$$\begin{array}{ll} K_{\mathrm{P}}MP^\sigma & \to M \\ S_{\mathrm{P}}M_1M_2P^\sigma & \to M_1P^\sigma(M_2P^\sigma) \\ \Pi^1_{\mathrm{PQ}}(P^\sigma Q^\sigma) & \to P^\sigma \\ \Pi^2_{\mathrm{PQ}}(P^\sigma Q^\sigma) & \to Q^\sigma \end{array}$$

siendo P, Q y PQ patrones, M, M_1 y M_2 términos cualesquiera y σ una sustitución de variables por términos.

Algunos ejemplos:

- $S_{\mathrm{Kx}}\Pi^1_{\mathrm{Ky}}\Pi^2_{\mathrm{KS}}(KS) \to \Pi^1_{\mathrm{Ky}}(KS)(\Pi^2_{\mathrm{KS}}(KS)) \to K(\Pi^2_{\mathrm{KS}}(KS)) \to KS.$
- $S_{\mathrm{Kx}}\Pi^1_{\mathrm{Ky}}\Pi^2_{\mathrm{KS}}(KS) \to \Pi^1_{\mathrm{Ky}}(KS)(\Pi^2_{\mathrm{KS}}(KS)) \to \Pi^1_{\mathrm{Ky}}(KS)S \to KS.$
- $\Pi^2_{\mathrm{S}\Pi^1_{\mathrm{Kx}}}(S\Pi^1_{\mathrm{Ky}})(KS_{\mathrm{K}}) \to \Pi^1_{\mathrm{Ky}}(KS_{\mathrm{K}}) \to K.$
- $\Pi^2_{\mathrm{K_K S}}(KS)$ no reduce, ya que KS no coincide con $K_{\mathrm{K}}S$.

Extendimos las traducciones entre el Cálculo Lambda y la Lógica Combinatoria SK, para poder traducir términos de λP a CL_P y viceversa. Por ejemplo:

$$\begin{array}{lcl} \lambda(\lambda x.x).y & \rightsquigarrow & K_{\mathrm{SKK}}\, y \\ \lambda(\lambda y.x).x & \rightsquigarrow & S_{\mathrm{Kx}}(K(SKK))\Pi^2_{\mathrm{Kx}} \end{array}$$

Se puede asimismo introducir constructores en el lenguaje para poder representar estructuras de datos más cómodamente. Los constructores también pueden ser utilizados como patrones, permitiendo definir funciones como las siguientes:

$$\texttt{head} = S_{(:)\,\mathrm{x\,y}}(K\Pi^2_{(:)\,\mathrm{z}})\Pi^1_{(:)\,\mathrm{v\,w}} \qquad \texttt{tail} = \Pi^2_{(:)\,\mathrm{x\,y}}$$

Desarrollamos además extensiones y variantes que se adaptan a diversas necesidades (como por ejemplo admitir otras nociones de correspondencia o distintas familias de patrones), así como un sistema de tipos para CL_P, para el cual probamos la terminación de todos los términos tipables.

Grupo de Investigación

Este trabajo fue realizado por miembros del Grupo de Reescritura, Cálculo Lambda y Sistemas de tipos.