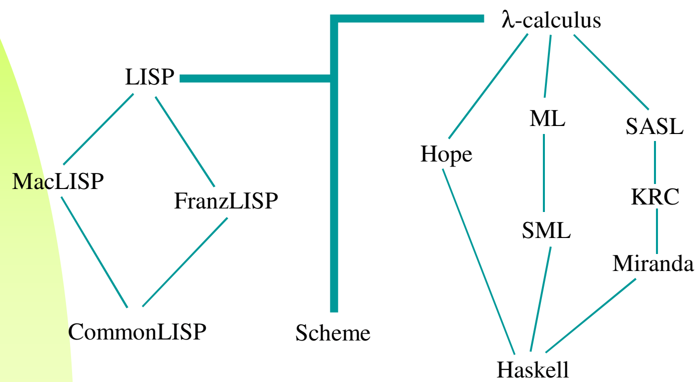
Lenguajes de programación

Introducción al Cálculo Lambda

El Cálculo Lambda (λ-calculus)

- Formalismo de cálculo propuesto por Alonso Church (1930).
- Describe el comportamiento de las funciones
  - Sintaxis $\rightarrow$ expresiones lambda (λ-expresiones)
  - Cálculos $\rightarrow$ reglas de cómputo
- A las reglas de cómputo también se les llama reglas de reescritura; sirven para transformar las λ-expresiones.
- Considerado como el primer lenguaje de programación funcional.

Evolución de la Programación Funcional



Algunas características del λ-calculus

- ¡¡¡Las funciones no tienen nombre!!!
- ¡¡¡Las expresiones no tienen tipos!!!
- Todas las funciones tienen un solo argumento.
- Las funciones se pueden aplicar a sí mismas.

Sintaxis del λ-calculus puro

- Se asume la existencia de un conjunto de identificadores (o variables).
  - $x \in$ Identificadores
- En BNF, $e \in \lambda$- expresión ssi

  $e ::= x \mid \boxed{e_1\ e_2} \mid \boxed{\lambda x.e} \mid (e)$

  Aplicaciones *Abstracciones*

Ejemplos

$e ::= x \mid e_1\ e_2 \mid \lambda x.e \mid (e)$

- x
- $x\ y$
- $(\lambda x.\lambda y.f\ x\ y)\ (\lambda x.x)\ w$
- $(\lambda x.(x\ x))\ (\lambda x.(x\ x))$
- $\lambda f.(\lambda x.f(x\ x))(\lambda x.f(x\ x))$
- …
- …

Extensión del λ-calculus puro para manejar constantes

- Se asume la existencia de un conjunto de identificadores (o variables) y de otro conjunto de constantes.
  - $x \in$ Identificadores
  - $c \in$ Constantes
- En BNF, $e \in \lambda$- expresión ssi

$$e ::= x \mid c \mid \underbrace{\boxed{e_1\ e_2}}_{\textit{Aplicaciones}} \mid \underbrace{\boxed{\lambda x.e}}_{\textit{Abstracciones}} \mid (e)$$

Ejemplos

$e ::= x \mid c \mid e_1\ e_2 \mid \lambda x.e \mid (e)$

- x
- 4
- + x 2
- (λx.λy.+ x y) (λz.z) 4
- (λf.λl. f l) car m
- λf.λx.COND (= x 1) 1 (* x (f (- x 1)))
- λf.(λx.f (x x))(λx.f (x x)) Fac
- …
- …

Interpretación en términos de una función

- **Abstracción**:
  - ***λx.x*** : es una función de un único argumento ***x***; la función regresa **x**
- **Aplicación**:
  - ***(λx. x) 3*** : ***3*** es el argumento de la función ***(λx. x)***.
- ¿Cómo se interpreta la abstracción ***(λx.λy.+ x y)***?
  - La función de un argumento (***x***) que regresa otra función de un argumento (***y***)
- ¿Cómo se interpreta la aplicación ***(λx.λy.+ x y) 3 4***?
  - La suma de *3* y *4*

Aplicación y asociatividad

- ¿Cómo se interpreta la expresión *(λx.λy. + x y) (λz.z) 3* ?
  - Opción 1: Los argumentos de *(λx.λy. + x y)* son *(λz.z)* y *3*
  - Opción 2: El argumento de *(λz.z)* es *3* y el resultado de esa aplicación se convierte en el argumento de *(λx.λy. + x y)*
- Asociatividad a la izquierda VS Asociatividad a la derecha.
- En este caso el resultado no es igual:
  - En el primer caso, la interpretación sería la λ-expresión *(((λx.λy. + x y) (λz.z)) 3)* equivalente a *+ (λz.z) 3*
  - En el segundo caso, la interpretación sería la λ-expresión *((λx.λy. + x y) ((λz.z) 3))* equivalente a *λy.+ 3 y*
- Por default: asociatividad a la izquierda

Reglas de conversión en λ-calculus con constantes

- A las reglas en general se les llama **conversiones**.
- Diferentes tipos de conversión
  - δ-conversión
    - Evaluación de funciones predefinidas
  - β-conversión
    - aplicación de una función a sus argumentos
  - α-conversión
    - renombrar variables
  - η-conversión
    - eliminar variables

¿Conversión o Reducción?

- Cuando las reglas sólo se aplican en un sentido se les llama **reducciones**.
- δ-reducción
  - Evaluación de funciones predefinidas
- β-reducción
  - aplicación de una función a sus argumentos
  - *Formalmente*: $(\lambda x.e_1)e_2 \Leftrightarrow [e_2/x]e_1$
  - *Informalmente*: reemplazar las ocurrencias de *x* en la λ-expresión e_1 por la λ-expresión e_2
- Evaluación de una constante igual a la constante misma.

Redex y reducción

- Un *redex* es una λ-expresión que puede ser reducida aplicando alguna regla (*δ-redex, β-redex, etc.*)

- $\boxed{(\lambda x.x)\ 3}$ tiene un *redex*

- $\boxed{(\lambda a.\lambda b.+\ a\ b)\ \boxed{((\lambda x.x)\ 3)}}$ *5* tiene dos *redex*

Redex y reducción

¿Cuántos *redex* tiene la siguiente λ-expresión?

=> La λ-expresión tiene 3 *redex*

δ-redex

$$\boxed{(\lambda x.\boxed{(\lambda f.f)\ +}\ \boxed{(*\ 3\ 4)}\ x)\ 7}$$

β-redex

Ejemplos de δ-reducción

- δ-reducción
  - Evaluación de funciones predefinidas
- Suponiendo que *Ctes = {=, +, 0, 1, If, True, False, Cons, ...}*:
  - $(= 0\ 0) \Rightarrow_\delta True$
  - $(= 0\ 1) \Rightarrow_\delta False$
  - $(+ 0\ 0) \Rightarrow_\delta 0$
  - $(+ 0\ 1) \Rightarrow_\delta 1$
  - $(+ 27\ 32) \Rightarrow_\delta 59$
  - $(If\ True\ e_1\ e_2) \Rightarrow_\delta e_1$
  - $(If\ False\ e_1\ e_2) \Rightarrow_\delta e_2$
  - $(Car\ (Cons\ e_1\ e_2)) \Rightarrow_\delta e_1$
  - $(Cdr\ (Cons\ e_1\ e_2)) \Rightarrow_\delta e_2$
  - …

Ejemplos de β-reducción

- Aplicación de una función a sus argumentos
  - Dado $(\lambda x.e_1)e_2$, reemplazar las ocurrencias de x en la λ-expresión e_1 por la λ-expresión e_2
- $(\lambda x.x)\ 3 \Rightarrow_\beta 3$
- $(\lambda a.\lambda b.+\ a\ b)\ (\lambda x.x) \Rightarrow_\beta \lambda b.+\ (\lambda x.x)\ b$
- $(\lambda a.\lambda b.+\ a\ b)\ ((\lambda x.x)\ 3)\ 5$

$$= [(\lambda a.\lambda b.+\ a\ b)\ ((\lambda x.x)\ 3)]\ 5$$
$$\Rightarrow_\beta (\lambda b.+\ ((\lambda x.x)\ 3)\ b)\ 5$$
$$\Rightarrow_\beta +\ ((\lambda x.x)\ 3)\ 5$$
$$\Rightarrow_\beta +\ 3\ 5 \Rightarrow_\delta 8$$

PREGUNTA

- ¿Existe una única forma para reducir una λ-expresión?

$$
\begin{tikzcd}
 & (\lambda x.(\lambda f.f) + (*\ 3\ 4)\ x)\ 7 \arrow[dl, "\Rightarrow_\beta"'] \arrow[d, "\Rightarrow_\beta"] \arrow[dr, "\Rightarrow_\delta"] & \\
(\lambda f.f) + (*\ 3\ 4)\ 7 \arrow[d, "\Rightarrow_\beta"'] \arrow[dr, "\Rightarrow_\delta"] & (\lambda x. + (*\ 3\ 4)\ x)\ 7 \arrow[dl, "\Rightarrow_\beta"'] \arrow[dr, "\Rightarrow_\delta"] & (\lambda x.(\lambda f.f) + 12\ x)\ 7 \arrow[dl, "\Rightarrow_\beta"'] \arrow[d, "\Rightarrow_\beta"] \\
+ (*\ 3\ 4)\ 7 \arrow[dr, "\Rightarrow_\delta"'] & (\lambda f.f) + 12\ 7 \arrow[d, "\Rightarrow_\beta"] & (\lambda x. + 12\ x)\ 7 \arrow[dl, "\Rightarrow_\beta"] \\
 & + 12\ 7 \arrow[d, "\Rightarrow_\delta"] & \\
 & 19 &
\end{tikzcd}
$$

Forma normal de una λ-expresión

- Se dice que una λ-expresión está en forma normal cuando ya no tiene "*redexes*"
  - *λx.λy.+ x y* está en forma normal
  - *λf.λb.f (λc.c)* está en forma normal
  - *(λx.λy.+ x y) 3* no está en forma normal
  - *λf.λb.(λc.c) f* no está en forma normal
- Una λ-expresión que no está en forma normal se puede llevar a ésta por medio de una reducción.
  - La forma normal de *(λx.λy.+ x y) 3* es *λy.+ 3 y*
  - La forma normal de *λf.λb.(λc.c) f* es *λf.λb.f*
  - La forma normal de *(λx.(λf.f) + (* 3 4) x) 7 es 19*

Importancia de la forma normal

- El concepto de forma normal es equivalente a la terminación de la evaluación.
- ¡¡¡¡ En términos de programación, es equivalente a la terminación de un programa !!!!

PROBLEMAS

¿Todas las λ-expresiones se pueden llevar a forma normal?

$$(\lambda z.z\,z)(\lambda z.z\,z)$$

$$\Big\downarrow \Rightarrow_\beta$$

$$(\lambda z.z\,z)(\lambda z.z\,z)$$

$$\Big\downarrow$$

$$\ldots$$

NO

¡¡¡ En términos de programación es equivalente a los programas que "ciclan" !!!

PROBLEMAS

Ordenes diferentes de reducción pueden conducir a resultados diferentes.

$$\boxed{(\lambda x.\lambda y.y)(\boxed{(\lambda z.z\ z)\ (\lambda z.z\ z)})}$$

$$\begin{tikzcd}
 & (\lambda x.\lambda y.y)((\lambda z.z\ z)\ (\lambda z.z\ z)) \arrow[dl, "\Rightarrow_\beta"'] \arrow[dr, "\Rightarrow_\beta"] & \\
(\lambda y.y) & & (\lambda x.\lambda y.y)((\lambda z.z\ z)(\lambda z.z\ z)) \arrow[d] \\
 & & \ldots
\end{tikzcd}$$

¿Cómo obtener la forma normal cuándo existe?

- El proceso de reducción se lleva a cabo siguiendo algún orden predeterminado:
  - El redex más pequeño.
  - El redex que está más a la izquierda.
  - El redex que está más a la derecha.
  - Primero δ-redex, luego β-redex
  - El más bonito
  - …

Tipos de redex

- **Interno**: no contiene otro(s) “redexes”
- **Externo**: no está contenido dentro de otro(s) “redexes”
- **Extrema derecha**: textualmente a la derecha de los demás.
- **Extrema izquierda**: textualmente a la izquierda de los demás.

$$(\lambda x.\boxed{(\lambda f.f)\ +}\ \boxed{(*\ 3\ 4)}\ x)\ 7$$

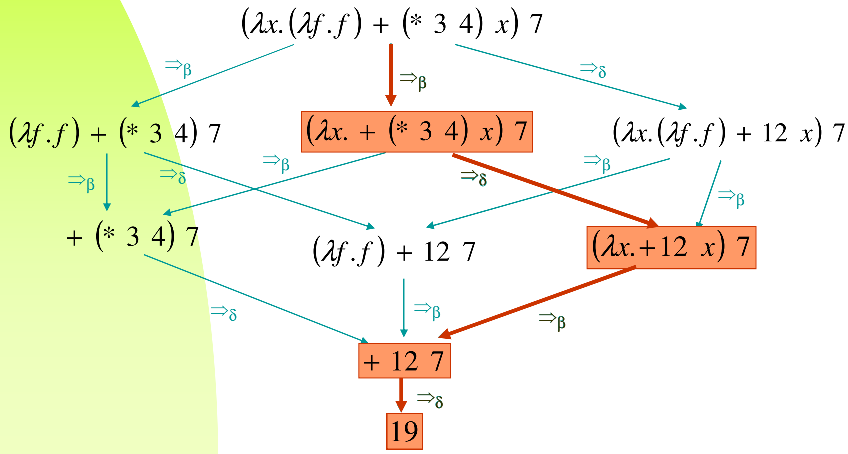
- (λf.f) + es el redex interno de extrema izquierda
- (* 3 4) es el redex interno de extrema derecha
- (λx.(λf.f) + (* 3 4) x) 7 es el redex externo de extrema izquierda y derecha al mismo tiempo.

Ordenes de reducción

- **Aplicativo**
  - Consiste en seleccionar el redex interno de extrema izquierda
- **Normal**
  - Consiste en seleccionar el redex externo de extrema izquierda

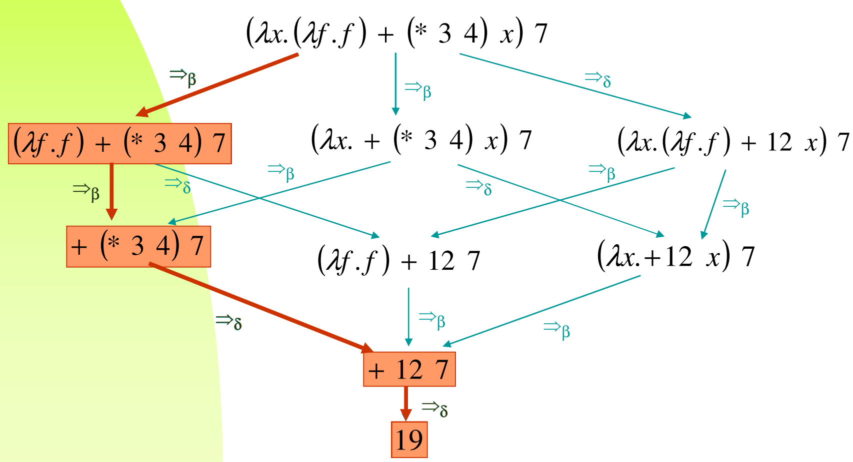
Orden de reducción aplicativo

- Seleccionar el *redex* **interno** de extrema izquierda.



Orden de reducción normal

- Seleccionar el *redex* **externo** de extrema izquierda.



¿Aplicativo o Normal?

$(\lambda x.\lambda y.y)((\lambda z.z\ z)\ (\lambda z.z\ z))$

$\downarrow \Rightarrow_\beta$

$(\lambda x.\lambda y.y)((\lambda z.z\ z)(\lambda z.z\ z))$

$\downarrow \Rightarrow_\beta$

$(\lambda x.\lambda y.y)((\lambda z.z\ z)(\lambda z.z\ z))$

$\downarrow$

…

VS

$(\lambda x.\lambda y.y)((\lambda z.z\ z)\ (\lambda z.z\ z))$

$\downarrow \Rightarrow_\beta$

$\lambda y.y$

¿Aplicativo o Normal?

$(\lambda x.*\ x\ x)(+\ 2\ 3)$

$\downarrow \Rightarrow_\delta$

$(\lambda x.*\ x\ x)5$

$\downarrow \Rightarrow_\beta$

$*\ 5\ 5$

$\downarrow \Rightarrow_\delta$

25

VS

$(\lambda x.*\ x\ x)(+\ 2\ 3)$

$\downarrow \Rightarrow_\beta$

$*\ (+\ 2\ 3)\ (+\ 2\ 3)$

$\downarrow \Rightarrow_\delta$

$*\ 5\ (+\ 2\ 3)$

$\downarrow \Rightarrow_\delta$

$*\ 5\ 5$

$\downarrow \Rightarrow_\delta$

25

Conclusiones

- **El orden de reducción APLICATIVO:**
  - Equivale a una sustitución tipo parámetros por valor.
  - Es por lo general más eficiente.
  - Es el que usa la mayoría de los lenguajes funcionales.
- **El orden de reducción NORMAL:**
  - Equivale a una sustitución tipo parámetro por nombre.
  - Asegura obtener la forma normal, si existe.
  - Es utilizado por los lenguajes de evaluación perezosa.