Universidad Tecnológica Nacional

Facultad Regional Resistencia

Paradigmas de Programación

Programación Funcional

Historia

- David Hilbert, siguiendo con su programa con el cual propone desafíos a los matemáticos (23 Problemas de Hilbert del 1900), formula 3 preguntas en 1928, la tercera de las cuales se conoce como:

  “Hilbert's Entscheidungsproblem”.

- El Entscheidungsproblem: es el reto en lógica simbólica de encontrar un algoritmo general que decida si una fórmula del cálculo de primer orden es lógicamente válida.
- El teorema de completitud de Gödel, postula que una formula lógica es lógicamente válida si y solo sí, para cada interpretación, la misma es verdadera.

David Hilbert (1862-1943)

http://www.answers.com/topic/hilbert-s-problems

http://www.answers.com/topic/entscheidungsproblem-1

Origen

- Sus orígenes provienen del Cálculo Lambda (λ-cálculo), una teoría matemática elaborada por **"Alonzo Church"** como apoyo a sus estudios sobre computabilidad en la década de 1930.
- Church usó el cálculo lambda en 1936 para resolver el Entscheidungsproblem
- Church probó que no había algoritmo que pudiese ser considerado como una "solución" al Entscheidungsproblem.
- Independientemente el mismo año, Turing prueba lo mismo con su "Máquina de Turing".

Alonzo Church (1903-1995)

UTN
FRRe

Cálculo Lambda I

- El cálculo lambda es un sistema formal diseñado para investigar:
  - *la definición de función,*
  - *la aplicación de una función,*
  - *la recursividad.*

- Por ejemplo:
  - **`f(x)=t`**, donde t contiene a **`x`**
  - luego **`f(u)`** = **`t[x:=u]`**,
    que resulta de sustituir **`u`** en cada aparición de **`x`** en **`t`**.
  - Si **`f(x)=x*x`**, entonces **`f(3)=3*3=9`**.

Cálculo Lambda II

- La principal característica del cálculo lambda es su simplicidad, ya que permite efectuar solo dos operaciones:

  - **Abstracción funcional:** Definir funciones de un solo argumento y con un cuerpo especifico, denotado por la siguiente terminología:

    $$\lambda \mathbf{x.B}$$

  - **Aplicación de función:** Aplicar alguna de las funciones definidas sobre un argumento real (A).

    $$(\lambda \mathbf{x.B})\ \mathbf{A}$$

Ejemplos:

- $(\lambda\ x.\ x + 2)\ 3 \quad \rightarrow \quad 5$
- $(\lambda\ f.\ f\ 3)\ (\lambda\ x.\ x + 2) \quad \rightarrow \quad (\lambda\ x.\ x + 2)\ 3 \quad \rightarrow \quad 3 + 2 \quad \rightarrow \quad 5$

Cálculo Lambda
Sintaxis

- Consideramos un conjunto finito de variables {a, b, c, ..., x, y, z}.
- El conjunto de todas las λ-expresiones por la siguiente gramática libre de contexto en BNF.

```
<expr> ::= <var>
         | (λ <var>.<expr>)       Abstracción Funcional
         | (<expr> <expr>)        Aplicación
```

Las dos primeras reglas generan funciones y la tercera, describe la aplicación de una función a un argumento.

Ejemplos

$\lambda x.x$

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

$\lambda f.f\ (\lambda x.x)$

http://www.answers.com/topic/lambda-calculus

UTN
FRRe

Convenciones sintácticas

Convenciones sintácticas para hacer más sencillas las λ-expresiones

1. La aplicación va a ser asociativa por la izquierda:

   $(((MN)P)Q) \rightarrow MNPQ$

2. La abstracción es asociativa por la derecha:

   $(\lambda x.(\lambda y.M)) \rightarrow \lambda x.\lambda y.M$

3. La aplicación es prioritaria sobre la abstracción:

   $(\lambda x.(MN)) \rightarrow \lambda x.MN$

4. Se puede suprimir símbolos **λ** en abstracciones consecutivas:

   $\lambda x.\ \lambda y.\ \ldots\ \lambda z.M \rightarrow \lambda xy\ \ldots\ z.M$

Ámbito de variables

- El ámbito de un identificador es la porción de un programa donde el identificador es accesible.

- La abstracción **λx.E** introduce a la variable **x** cuyo ámbito es la expresión **E** vinculando a todas las variables **x** que ocurran en **E**.
- En este caso, decimos que **x** está vinculada en la abstracción **λx.E**. La abstracción es similar a los argumentos formales de una función en el cuerpo de la función.

Variables libres y ligadas

- **Variables Ligadas**
  - Una variable **x** se dice ligada (o asociada) en una expresión **E** si aparece en el ámbito de una abstracción de variable instanciable **x**.

$$\begin{aligned} \mathrm{Bound}[x] &= \{\} \\ \mathrm{Bound}[\lambda x.E] &= \mathrm{Bound}[E] \cup \{x\} \\ \mathrm{Bound}[E_1\ E_2] &= \mathrm{Bound}[E_1] \cup \mathrm{Bound}[E_2] \end{aligned}$$

  - Ejemplo:
    - **(λy.z (λx.x y))**
      - La z es libre y las x están ligadas.
      - Las dos **y** son ligadas.

Variables Ligadas

$\textbf{Bound}[\lambda y.x\ (\lambda x.x\ y)] =$

$\textbf{Bound}[x\ (\lambda x.x\ y)] \cup \{y\} =$

$\textbf{Bound}[x] \cup \textbf{Bound}[(\lambda x.x\ y)] \cup \{y\} =$

$\{\ \} \cup \textbf{Bound}[(\lambda x.x\ y)] \cup \{y\} =$

$\{\ \} \cup \textbf{Bound}[x\ y] \cup \{x\} \cup \{y\} =$

$\{\ \} \cup \textbf{Bound}[x] \cup \textbf{Bound}[y] \cup \{x\} \cup \{y\} =$

$\{\ \} \cup \{\ \} \cup \{\ \} \cup \{x\} \cup \{y\} =$

$\{x, y\}$

$\textbf{Bound}[\lambda xy.x] =$

$\textbf{Bound}[\lambda x.(\lambda y.x)] =$

$\textbf{Bound}[\lambda y.x] \cup \{x\} =$

$\{y\} \cup \{x\} =$

$\{y, x\}$

Variables libres y ligadas

- **Variables Libres**
  - Una variable se dice libre en **E** si tiene ocurrencias que no están ligadas en **E**. El conjunto de las variables libres de una expresión **E** se pueden definir recursivamente como sigue:

$$\begin{aligned}
\mathrm{Free}[x] &= \{x\} \\
\mathrm{Free}[\lambda x.E] &= \mathrm{Free}[E] - \{x\} \\
\mathrm{Free}[E_1\ E_2] &= \mathrm{Free}[E_1] \cup \mathrm{Free}[E_2]
\end{aligned}$$

  - Ejemplos:
    - **$\mathrm{Free}[\lambda x.x(\lambda y.xyz)] = \{z\}$**
    - **$\mathrm{Free}[\lambda xy.x] = \varnothing$**

UTN
FRRe

Variables Libres

$Free[\lambda x.x\ (\lambda y.xyz)] =$

$Free[x\ (\lambda y.xyz)] - \{x\} =$

$Free[x] \cup Free[(\lambda y.xyz)] - \{x\} =$

$\{x\} \cup Free[xyz] - \{y\} - \{x\} =$

$\{x\} \cup Free[x] \cup Free[y] \cup Free[z] - \{y\} - \{x\} =$

$\{x\} \cup \{x\} \cup \{y\} \cup \{z\} - \{y\} - \{x\} =$

$\{x, y, z\} - \{y\} - \{x\} =$

$\{x, z\} - \{x\} =$

$\{z\}$

$Free[\lambda xy.x] =$

$Free[\lambda x(\lambda y.x)] =$

$Free[\lambda y.x] - \{x\} = \varnothing$

Relación de equivalencia

- El conjunto de todas las expresiones lambda se denomina Λ.
- Sobre este conjunto se define una relación de equivalencia basada en la idea que dos expresiones pueden denotar la misma función.
- Esta relación de equivalencia se define mediante reglas de cálculo que hacen cumplir las propiedades:
  - Reflexiva: $M \equiv M$
  - Simétrica: $M \equiv N \Rightarrow N \equiv M$
  - Transitiva: $M \equiv N$ y $N \equiv P \Rightarrow M \equiv P$

Equivalencia de Expresiones

- *Definición:* En λ-cálculo dos λ-expresiones *M* y *N* que sólo difieren en sus variables ligadas son **equivalentes**.

Ejemplo :

M = x (λy.y)   es equivalente a

N = x (λz.z)

M ≡ N

Semántica Operacional

- La evaluación de una expresión se compone de pasos de reducción donde cada uno de los pasos se obtiene por reescritura:

$$\mathbf{E} \rightarrow \mathbf{E}'$$

- Se parte de un estado inicial (expresión inicial) y mediante un proceso de reescritura se obtiene un estado final (expresión final)

- Cada reducción de **E**, reemplaza cierta subexpresión de acuerdo con ciertas reglas; tales subexpresiones se llaman ***redex*** (**red**ucible **ex**pression).

- Se considera finalizado el cómputo cuando ya no aparecen más redexes.

λ-reducciones

- Las reglas de reescritura que se utilizan para reescribir un redex son:
  - δ-REDUCCIÓN
  - α-REDUCCIÓN o α-CONVERSIÓN
  - β-REDUCCIÓN
  - η-REDUCCIÓN

δ-reducción

- Se llaman δ-reducción a la regla que transforma constantes. Se describe con $\rightarrow_\delta$

- Ejemplo

$$* \; (+\,1\,2) \; (-\,4\,1) \rightarrow_\delta$$
$$* \; 3 \; (-\,4\,1) \rightarrow_\delta$$
$$* \; 3 \; 3 \rightarrow_\delta$$
$$9$$

α-conversión

- Definiremos la relación de **α-reducción** (o **α-conversión**) como sigue:

$$\lambda x.M \quad \rightarrow_\alpha \quad \lambda y.\,[x{:=}y]M$$

$$\text{si } y \notin \mathrm{Free}(M).$$

- La **α-reducción** es formalizar que *si renombramos variables ligadas* de **λ-expresiones,** *éstas no cambian* (mientras no utilicemos variables libres para la sustitución).

UTN
FRRe

β-reducción

- La **β-reducción** es el proceso de sustitución del argumento N, sobre el cuerpo de la abstracción, reemplazando todas las ocurrencias de la variable instanciable por el argumento.

$$(\lambda x.M)\ N \quad \to_\beta \quad [x:=N]\ M$$

  - La λ-expresion **(λx.M) N** es un **β-redex**, es decir, se puede reducir mediante una β-reducción.
  - Otras notaciones: **[N/x] M – M[x:=N] – “x:=N”.e**

UTN
FRRe

β-reducción Ejemplos

- **$(\lambda x.\ *\ x\ x)\ 2 \quad \rightarrow_\beta \quad (*\ 2\ 2) \quad \rightarrow_\delta \quad 4$**
- **$(\lambda x.\ *\ x\ x)\ 2 \quad \rightarrow_{\beta\delta} \quad 4$**
- **$(\lambda x.x\ y)\ (\lambda z.z) \quad \rightarrow_\beta \quad (\lambda z.z)\ y \quad \rightarrow_\beta \quad y$**
- **$((\lambda xy.\ *\ x\ y)\ 7)\ 8 \quad \rightarrow_\beta \quad (\lambda y.\ *\ 7\ y)\ 8 \quad \rightarrow_\beta \quad *\ 7\ 8 \quad \rightarrow_\delta \quad 56$**
- **$(\lambda f.f\ 3)\ (\lambda x.+\ x\ 1) \quad \rightarrow_\beta \quad (\lambda x.+\ x\ 1)\ 3 \quad \rightarrow_\beta \quad +\ 3\ 1 \quad \rightarrow_\delta \quad 4$**

UTN
FRRe

β-reducción
Ejercicios

- **Reducir las siguientes expresiones:**

- **$(\lambda v.((\lambda z.z)\ x))$**

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

η-reducción

- La η-reducción (también llamada extensionalidad) expresa la idea de que dos funciones son lo mismo si dan el mismo resultado para todos sus argumentos.
-

$$\lambda \mathbf{x.M\ x} \ \rightarrow_\eta \ \mathbf{M}$$

- La λ-expresión λ**x.M x** es un η-redex, es decir, se puede reducir mediante una η-reducción. También se dice que M se expande o se extiende de λx.M x.

Ejemplos:

$\lambda xy.+\ y\ x \quad \rightarrow_\eta \quad \lambda y.+\ y \quad \rightarrow_\eta \quad +$

$\lambda x.(\lambda y.y)\ x \quad \rightarrow_\eta \quad \lambda y.y$

Sustitución

- La sustitución de x por N en M (denotada por [x:=N]M) es el resultado de cambiar en el λ-termino M, todas las apariciones de la variable libre x por el λ-termino N.

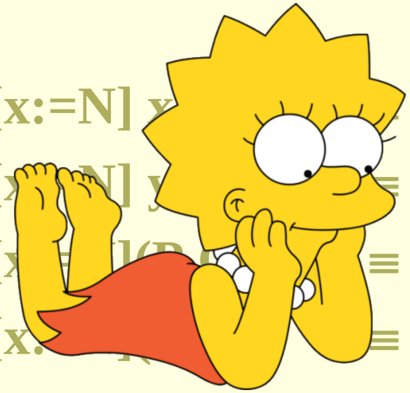
[x:=N] x	**≡**	**N**	
[x:=N] y	**≡**	**y**	**si x ≠ y**
[x:=N](P Q)	**≡**	**[x:=N]P [x:=N]Q**	
[x:=N](λx.P)	**≡**	**λx.P**	*(porque x está ligada en P)*
[x:=N](λy.P)	**≡**	**λy.([x:=N]P)**	**si x ≠ y**

UTN
FRRe

Sustitución

- La sustitución de x por N en M … [x:=N]M) es el resultad… …nino M, todas las aparicio… …r el λ-termino N.

Qué ocurriría si en P hay una “x” que necesite sustituirse y en “N” una “y” libre?



$[x:=N]\,x \equiv N$

$[x:=N]\,y \equiv y$ si $x \neq y$

$[x:=N](P\,Q) \equiv [x:=N]P\ [x:=N]Q$

$[x:=N](\lambda x.P) \equiv \lambda x.P$ *(porque x está ligada en P)*

$[x:=N](\lambda y.P) \equiv \lambda y.([x:=N]P)$ si $x \neq y$

$$\lambda y.([x:=N]P)$$

Captura de Variables

- Al sustituir **y** en el cuerpo de la abstracción, la ocurrencia libre de **y** reemplazará a **x**, transformándose en ligada:

$$(\lambda x.\ (\lambda y.\ P))\ \ N$$

$$(\lambda x.\ (\lambda y.\ (\ x\ \ y\ )))\ \ \ y\ =\ \ \lambda\, y\, .\, (\ y\ \ y\ )$$

- El conflicto ocurre cuando:
  - y ocurre libre en N; y
  - x ocurre libre en P

Las **y** libres de N se ligarán en **λy.P**

UTN
FRRe

Sustitución Segura

- Una sustitución segura es aquella en la que no se produce ninguna captura de variables.

- *Formalmente*:

  Para la sustitución:

  $$[x:=N]P$$

  Se ha de cumplir la condición suficiente:

  $$\textbf{Bound (P)} \cap \textbf{Free (N)} = \varnothing$$

- Si esto ocurre será necesario hacer una α-conversión:

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

$$\Downarrow$$

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

UTN
FRRe

Sustitución: redefinición

- Podríamos redefinir la sustitución usando la noción de α-equivalencia y evitar de este modo la captura de variables.

$$
\begin{array}{llll}
 & [x:=N]\,x & \equiv N & \\
 & [x:=N]\,y & \equiv y & \text{si } x \neq y \\
 & [x:=N](P\,Q) & \equiv [x:=N]P\,[x:=N]Q & \\
 & [x:=N](\lambda x.P) & \equiv \lambda x.P & \\
\text{si } y \notin Free(N) & [x:=N](\lambda y.P) & \equiv \lambda y.([x:=N]P) & \text{con } x \neq y \\
\text{si } y \in Free(N) & [x:=N](\lambda y.P) & \equiv \lambda z.([x:=N]([y:=z]P)) & \\
 & & \quad \text{con } z \notin Free(N) \cup Free(P); & \\
 & & \quad x \neq y &
\end{array}
$$

Redex

- Un redex es un termino de la forma:

$$(\lambda x.M)\ \ N$$

La abstracción funcional representa la **función a aplicar** y el término N el **argumento efectivo**.

Un redex representa la idea de un cómputo que está por realizarse.

Forma Normal

- Dada una λ-expresión nos interesa su forma mas reducida, que sería la "salida" de la función (no contiene ningún redex).
- *Definición:* Una λ-expresión está en **forma normal** si no contiene ningún redex.
- Si una λ-expresión M se reduce a una λ-expresión N en forma normal, es decir,

$$M \to^* N$$

decimos que N es una forma normal de M.

Forma Normal II

- **Observación:**

  No toda λ-expresión admite forma normal.

  Ejemplo: $\boldsymbol{\Omega = (\lambda x.x\ x)(\lambda y.y\ y)}$

$$\boldsymbol{\Omega = (\lambda x.x\ x)(\lambda y.y\ y)}$$

$$\boldsymbol{[x:=(\lambda y.y\ y)]\ (x\ x)}$$

$$\boldsymbol{(\lambda y.y\ y)(\lambda y.y\ y) = \Omega}$$

Observamos que $\Omega \rightarrow^* \Omega$; es decir, nunca se llega a una expresión sin β-redex (no β-reducible), luego no admite forma normal.

UTN
FRRe

Reducción de λ-expresiones

- La reducción de un término a una forma normal puede realizarse siguiendo una variedad de estrategias posibles en la aplicación de las reglas.

Hagamos la reducción:

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

Ordenes de Reducción

- El orden de reducción determina la elección del redex a reducir; para identificar cuál será el redex elegido se usará lo siguiente:
  - **Redex más a la izquierda:** es aquel cuya λ aparece textualmente a la izquierda de cualquier otro redex de la expresión.
  - **Redex externo:** es aquel que no está contenido en otro redex.
  - **Redex interno:** es aquel que no contiene otro redex.

- Ejemplos:

$$\underbrace{(\lambda x.x)\ a}_{\text{interno}}\ (\overbrace{(\lambda x.x)\ b}^{\text{interno}})$$

$$\underbrace{(\lambda y.(\overbrace{(\lambda z.y)\ x}^{\text{interno}})\ y)\ a}_{\text{externo}}$$

Ordenes de Evaluación II

- Se distinguen dos ordenes de evaluación más importantes:
  - **Orden Impaciente:** se reduce el redex más interno de más a la izquierda.
  - **Orden Perezoso:** se reduce el redex más externo de más a la izquierda.

Propiedades

- *Propiedades de “confluencia”*
- *Propiedades de “terminación”*

Propiedad de Confluencia

- **Teorema (de Churh-Rosser):**

  Para toda λ-expresión M, si $M \rightarrow^* P$ y $M \rightarrow^* Q$, existe una λ-expresión E tal que $P\rightarrow^*E$ y $Q\rightarrow^*E$ (es la "propiedad del diamante").

$$\begin{tikzcd}[column sep=small] & M \arrow[dl] \arrow[dr] & \\ P \arrow[dr, dashed] & & Q \arrow[dl, dashed] \\ & E & \end{tikzcd}$$

- **Consecuencia (corolario):**

  Si M admite forma normal N, ésta es **única** salvo α-reducción (renombramiento de variables).

Propiedad de Terminación

- **Observación:**

  No toda secuencia de λ-reducciones termina

  Como en el caso visto: $\mathbf{\Omega = (\lambda x.x\ x)(\lambda y.y\ y)}$

- **Teorema:**

  Si una secuencia $M \to^* \ldots$ termina, entonces lo hace en una forma normal. Es decir, entonces M admite forma normal.

UTN
FRRe

Propiedad de Terminación

- **Observación:**

  Si M admite forma normal, esto **no** significa que cualquier secuencia que empiece en M, termine.

  Ejemplo:

  $((\lambda y.x)\ \Omega)$ admite forma normal x.

  Impaciente: empezando "por dentro" ($\Omega \to \Omega \to$ ...)

  $$(\lambda y.x)\ \Omega \to (\lambda y.x)\ \Omega \to \ldots$$ (no termina)

  Perezoso: empezando por fuera

  $$(\lambda y.x)\ \Omega \to x$$

- **Teorema (de "estandarización"):**

  Si E admite forma normal, y reducimos eligiendo los β-redex "de izquierda a derecha, y de fuera hacia dentro", (forma "perezosa") entonces la reducción termina (en la forma normal de E).

UTN FRRe