Metodologías de Programación II
Cálculo-λ

Dr. Alejandro Guerra-Hernández

Departamento de Inteligencia Artificial
Facultad de Física e Inteligencia Artificial
aguerra@uv.mx
http://www.uv.mx/aguerra

Maestría en Inteligencia Artificial 2009

Referente histórico

- El Cálculo-λ, publicado por Alonzo Church en 1932, es uno de los modelos matemáticos que se desarrollaron como respuesta inmediata a los problemas de Hilbert, a principios del siglo XX.
- El problema en cuestión es si existe un método mecánico general para obtener el valor de verdad de cualquier conjetura lógica, y está íntimamente relacionado con la cuestión de lo que es algorítmicamente computable.
- Lo uso también para resolver negativamente el problema del paro.

Universidad Veracruzana

Trabajos similares

- Los trabajos de Schoenfinkel y los combinadores de Curry (una forma especial de Cálculo-λ),
- Los números de Göedel, el sistema de producción de Post, las funciones recursivas de Kleene, los algoritmos de Markov;
- y el más prominente: la máquina de Turing.
- La tesis de Church-Turing: los problemas intuitivamente o efectivamente computables son exactamente aquellos que pueden computarse en un máquina de Turing (y por lo tanto, en los demás modelos también).

Universidad Veracruzana

Ideas principales del Cálculo-λ (1)

- Es el modelo más cercano a los algoritmos y su evaluación tal y como fueron formalizados en la clase anterior.
- Se le puede considerar como el paradigma de todos los lenguajes de programación tal y como los conocemos actualmente.
- Es primordialmente, una teoría de las funciones computables relacionada con propiedades fundamentales de los operadores, sus aplicaciones a operandos y con la construcción sistemática de operadores más complejos (algoritmos) a partir de componentes simples.

Ideas principales del Cálculo-λ (2)

- Su núcleo tiene que ver con poco más que la definición de variables, alcance de variables, y la substitución ordenada de variables por expresiones.
- Se trata de un lenguaje cerrado en el sentido de que su semántica puede definirse en base a equivalencia entre expresiones (o términos) del cálculo mismo.

Universidad Veracruzana

Abstracción funcional

- Una función f de n variables $v_1, \ldots, v_n$ se denotan en el Cálculo-λcomo:

$$f = \lambda v_1 \ldots v_n.e_0$$

- El símbolo f denota el nombre de la función y puede ser referenciado en cualquier parte. No es necesario especificarlo.
- La expresión del lado derecho de la ecuación define una abstracción de las variables $v_1, \ldots, v_n$ en el cuerpo de la función e_0.

Universidad Veracruzana

Variables libres, primera aproximación

- Una definición precisa de lo que ocurrencia libre de variables significa, deberemos posponerla para más adelante.
- Por ahora basta comentar que una variable u es libre en la expresión e_0 si ésta no contiene ninguna abstracción de u (la variable no está en la lista $\lambda \ldots$).

Aplicación de funciones

- Una aplicación llamada f sobre r expresiones argumento $e_1, \ldots, e_r$ tiene la forma:

$$(f\ e_1 \ldots\ e_r) = (\lambda v_1 \ldots v_n.e_0\ e_1 \ldots e_r)$$

donde r no es necesariamente igual a n. Por ejemplo:

$$(\lambda xy. +\ x\ y)\ 1\ = \lambda y. +\ 1\ y$$

- El caso especial de la aplicación de una abstracción a las variables abstraídas, nos da como resultado el cuerpo de la abstracción:

$$(\lambda v_1 \ldots v_n.e_0\ v_1 \ldots v_n) = e_0$$

Funciones Curryficadas

- Para mantener el aparato formal simple y conciso, el Cálculo-λ considera, sin perdida de generalidad, unicamente abstracciones de una variable (funciones de un argumento).
- Esto se debe al descubrimiento de Schoenfinkel y Curry que permite representar abstracciones *n*-arias, como *n*-folds anidados de abstracciones unarias ¿recuerdan el extraño nombre de funciones Curryficadas?

$$f = \lambda v_1 \dots v_n . e_0 \equiv \lambda v_1 . \lambda v_2 \dots . \lambda v_n . e_0$$

Sintáxis y Funciones Curryficadas

- Usando la notación currificada, la aplicación de f a r operandos toma la forma de un r-fold de aplicaciones anidadas:

$$(f\ e_1 \ldots e_r) \equiv (\ldots((f\ e_1)\ e_2)\ldots e_r)$$

- Lo anterior reduce la construcción de expresiones (o términos) del Cálculo-λ a la siguiente regla sintáctica:

$$e =_s v \mid c \mid (e_0\ e_1) \mid \lambda v.e_0$$

- Las expresiones que se forman a partir de la aplicación sistemática de estas reglas se conocen como fórmulas bien formadas (fbf) del Cálculo-λ.
- Cualquier otra expresión no es válida en el Cálculo-λ.

Universidad Veracruzana

Aplicaciones revisitadas

- Una aplicación $(e_0\ e_1)$ representa el resultado de aplicar e_0 a e_1.
- Se dice que e_0 es la expresión en la posición del operador; y que e_1 es la expresión en la posición del operando de la aplicación.
- Es común que e_0 y e_1 se identifiquen como la función y el argumento de la aplicación, lo cual no es totalmente correcto, puesto que la sintaxis del Cálculo-λ permite que cualquier expresión válida esté en cualquiera de las dos posiciones y sólo las abstracciones y las operaciones primitivas $+, -, *, \ldots$ son funciones verdaderas.

Universidad Veracruzana

Aplicaciones de abstracciones

- Estamos interesados particularmente en aplicaciones de la forma $(\lambda v.e_0\ e_1)$ que tienen una abstracción en la posición del operador y una expresión válida en la posición del operando.
- De hecho, basta considerar únicamente el Cálculo-λ puro, cuyo conjunto de constantes está vacío. Sin operadores primitivos, las abstracciones son las únicas funciones en juego.

Reglas de transformación

- La belleza del Cálculo-λ puro reside en que sólo necesitamos preocuparnos por una sola regla de transformación:

$$\lambda v.e_0\ e_1 \rightarrow_\beta\ e_0[v \leftarrow e_1]$$

- La regla se conoce como β-reducción o β-contracción y se dice que simplifica o reduce el β-redex $(\lambda v.e_0\ e_1)$ a su reductum o contractum $e_0[v \leftarrow e_1]$.

Conflictos entre nombres de variables

- Desafortunadamente esta regla no es tan simple como parece a primera vista. Existen problemas con respecto a las variables ligadas en las abstracciones y la existencia de variables libres con los mismos nombres, en cuyo caso, las substituciones no pueden llevarse a cabo sin una acción correctiva. Por ejemplo en:

$$(\lambda u.\lambda v.u\ w) \to \lambda v.w \quad \text{y} \quad (\lambda u.\lambda v.u\ v) \to \lambda v.v$$

Universidad Veracruzana

Variable libre

- Con V denotando el conjunto de variables, definimos el conjunto de variables libres FV de una expresión e recorriendo las tres formas sintácticas del Cálculo-λ puro y especificando lo que son las variables libres por casos:

$$FV(e) = \begin{cases} \{v\} & \text{Si } e =_s v \in V \\ FV(e_0) \cup FV(e_1) & \text{Si } e =_s (e_0\ e_1) \\ FV(e_0) \setminus \{v\} & \text{Si } e =_s \lambda v.e_0 \end{cases}$$

Variable acotada

- Es posible ofrecer una definición complementaria del conjunto de variables acotadas en la expresión e:

$$BV(e) = \begin{cases} \emptyset & \text{Si } e =_s v \in V \\ BV(e_0) \cup BV(e_1) & \text{Si } e =_s (e_0\ e_1) \\ BV(e_0) \cup \{v\} & \text{Si } e =_s \lambda v.e_0 \end{cases}$$

- Una variable v está libre en una expresión e, si y sólo si $v \in FV(e)$; y que una variable v está acotada en una expresión e, si y sólo si $v \in BV(e)$.

Alcance del abstractor

- En una abstracción $\lambda v.e$, llamamos al cuerpo e el alcance del abstractor λv, lo que significa que todas las ocurrencias libres de v en e están acotadas por λv. Por ejemplo, en la expresión:

  $$(\lambda u.(\lambda v.(\lambda z.(z\ (v\ u))\ v)\ u)\ w)$$

  la variable w está libre, puesto que no existe abstractor para ella. La variable u está acotada en la abstracción $\lambda u.(\ldots)$, pero libre en su cuerpo, que es el alcance del abstractor λu.

Universidad Veracruzana

Abstracciones cerradas y abiertas

- Una abstracción cuyo conjunto de variables libres está vacío, se dice cerrado o que es un combinador;
- En otro caso se dice que la abstracción es abierta.
- De gran relevancia para la implementación de lenguajes de programación basados en el Cálculo-λ, son los llamados super combinadores, que son abstracciones cerradas cuyos cuerpos pueden contener recursivamente sólo expresiones cerradas (o super combinadores). Por ej.

$$I = \lambda x.x$$

Substituciones

- Como se lleva a cabo la substitución en la β-reducción

$$(\lambda v.e_b\ e_a) \to_\beta e_b[v \leftarrow e_a]$$

$$e_b[v \leftarrow e_a] \quad = \quad \begin{cases} e_a & \text{Si } e_b =_s v \in V \\ u & \text{Si } e_b =_s u \in V \wedge v \neq_s u \\ (e_0[v \leftarrow e_a]\ e_1[v \leftarrow e_a]) & \text{Si } e_b =_s (e_0\ e_1) \\ \lambda v.e_0 & \text{Si } e_b =_s \lambda v.e_0 \\ \lambda u.e_0[v \leftarrow e_a] & \text{Si } e_b =_s \lambda u.e_0 \quad \wedge \\ & u \notin FV(e_a) \vee v \notin FV(e_b) \\ \lambda w.e_0[u \leftarrow w][v \leftarrow e_a] & \text{Si } e_b =_s \lambda u.e_0 \quad \wedge \\ & u \in FV(e_a) \wedge v \in FV(e_b) \quad \wedge \\ & w \in V \wedge w \notin FV(e_a) \cup FV(e_b) \end{cases}$$

Renombrado de variables: α-conversión

- La transformación que renombra:

$$\lambda u.e_0 \rightarrow_\alpha \lambda w.e_0[u \leftarrow w]$$

se conoce como α-conversión. Su realización se basa en la aplicación de una función de α-conversión a la abstracción cuyas variables ligadas necesitamos cambiar:

$$(\lambda v.\lambda w.(v\ w)\ \lambda u.e_0)$$

- Esta transformación procede con la aplicación de las reglas definidas anteriormente en dos pasos: primero mapea a $\lambda w.(\lambda u.e_0\ w)$ y después a $\lambda w.e_0[u \leftarrow w]$.

Universidad Veracruzana

Ejemplo de β-reducciones

$$(\lambda u.(\lambda v.(\lambda z.(z\ u)\ v)\ u)\ z)$$
$$\downarrow$$
$$(\lambda v.(\lambda z.(z\ u)\ v)\ u)[\ u \leftarrow z\ ]$$
$$\downarrow$$
$$(\lambda v.(\lambda z.(z\ u)\ v)[\ u \leftarrow z\ ]\ u[\ u \leftarrow z\ ])$$
$$\downarrow$$
$$(\lambda v.(\lambda z.(z\ u)\ v)[\ u \leftarrow z\ ]\ z)$$
$$\downarrow$$
$$(\lambda v.(\lambda z.(z\ u)[\ u \leftarrow z\ ]\ v[\ u \leftarrow z\ ]\ )\ z)$$
$$\downarrow$$
$$(\lambda v.(\lambda w.(z\ u)[\ z \leftarrow w\ ][\ u \leftarrow z\ ]\ v)\ z)$$
$$\downarrow$$
$$(\lambda v.(\lambda w.(w\ u)[\ u \leftarrow z\ ]\ v)\ z)$$
$$\downarrow$$
$$(\lambda v.(\lambda w.(w\ z)\ v)\ z)$$
$$\downarrow$$
$$(\lambda w.(w\ z)\ v)[\ v \leftarrow z\ ]$$
$$\downarrow$$
$$(\lambda w.(w\ z)\ w)$$
$$\downarrow$$
$$(w\ z)[\ w \leftarrow z\ ]$$
$$\downarrow$$
$$(z\ z)$$

Universidad Veracruzana

Problemas

- La historia es diferente para λ-expresiones que reducen abstracciones, como:

$$(\lambda u.(\lambda v.\lambda z.((z\ u)\ v)\ z)\ v))$$

- Encontramos dos conflictos entre nombres, el primero al substituir el argumento externo v bajo λv; y el segundo al substituir el argumento interno z bajo λz. Si renombramos v por x y luego z por y, obtenemos la abstracción $\lambda y.((y\ v)\ z)$ como resultado.
- Si invertimos el orden en el que usamos x e y, obtendríamos la abstracción $\lambda x.((x\ v)\ z)$.

Universidad Veracruzana

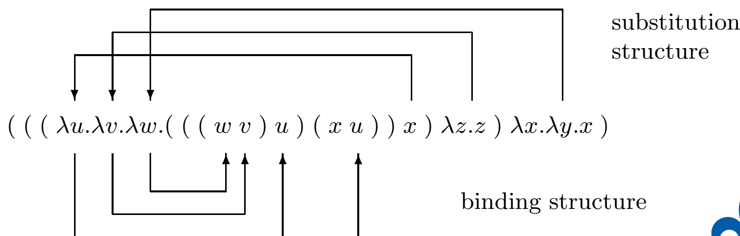
Estructura de ligado

- La elección de los nombres de las variables ligadas no es importante. Su única función es relacionar a los abstractores con posiciones sintácticas en el cuerpo de la abstracción. substituidos. A esta relación se le conoce como estructura de ligado. Por ejemplo:

$$\lambda u.\lambda v.\lambda w.(((w\ v)\ u)(x\ u)) \equiv \lambda x_1.\lambda x_2.\lambda x_3.(((x_3\ x_2)\ x_1)(x\ x_1))$$

Estructura de substitución

- La posición de un abstractor en una secuencia de abstractores también identifica, en el caso de aplicaciones anidadas, el nivel de anidamiento en que el argumento será tomado. A esto se le llama estructura de substitución.



Otro ejemplo

$$(((\lambda u.\lambda v.\lambda w.(((w\ v)\ u)\ (x\ u))\ x)\ \lambda z.z)\ \lambda x.\lambda y.x)$$

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

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

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

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

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

$$(x\ x)$$

Un nuevo renombrado de variables

- A todas las variables se les dará el mismo nombre, por ejemplo z, y la estructura de ligado se definirá por medio de los llamados operadores de desligue ó llaves protectoras que se colocan frente a la ocurrencia de las variables en el cuerpo de la abstracción.
- Si una ocurrencia de la variable z está precedida por n de estos operadores, denotados como:

$$\underbrace{//\ldots/}_{n}\; z$$

entonces la ocurrencia está protegida contra las n imbricaciones más internas del abstractor λz que liga a la variable.

Ejemplo

- Usando está notación, la abstracción del ejemplo puede representarse como sigue:

$$\lambda u.\lambda v.\lambda w.(((w\ v)\ u)(x\ u)) =_s \lambda z.\lambda z.\lambda z.(((z\ /z)\ //z)\ (x\ //\ z))$$

- Todas las variables ligadas se llaman ahora z y, por ejemplo, las ocurrencias de z que remplazan a la variable original u, están precedidas por dos llaves de protección $//$, de forma que los dos abstractores λz más internos, no le afecten. La variable x no se modifica porque aparece libre.

Problemas

- El problema con las llaves de protección es que necesitan ser modificadas dinámicamente, conforme se aplican las β-reducciones.
- Cada vez que un abstractor desaparece de la abstracción original, una llave de protección debe desaparecer.
- Pero también, si una variable libre entra en el alcance de un abstractor, y esta variable tiene el mismo nombre que la variable abstraída, entonces en concordancia, debemos introducir una llave de protección más.

Universidad Veracruzana

Ejemplo

$$\lambda z.\lambda z.((\underline{(\lambda z.\lambda z.\lambda z.(((z\ /z)\ //z)\ (////z\ //z))\ /z)}\ \lambda z.z)\ \lambda z.\lambda z./z)$$

$$\lambda z.\lambda z.(\underline{(\lambda z.\lambda z.(((z\ /z)\ ///z)\ (///z\ ///z))\ \lambda z.z)}\ \lambda z.\lambda z./z)$$

$$\lambda z.\lambda z.\underline{(\lambda z.(((z\ \lambda z.z)\ //z)\ (//z\ //z))\ \lambda z.\lambda z./z)}$$

$$\lambda z.\lambda z.((\underline{(\lambda z.\lambda z./z\ \lambda z.z)\ /z)}\ (/z\ /z))$$

$$\lambda z.\lambda z.(\underline{(\lambda z.\lambda z.z\ /z)}\ (/z\ /z))$$

$$\lambda z.\lambda z.\underline{(\lambda z.z\ (/z\ /z))}$$

$$\lambda z.\lambda z.(/z\ /z)$$

Estado de ligadura

- Sea

$$\underbrace{/\ldots//}_{i}\ v =_s /(i) \mid i \in \{0, 1, \ldots\}$$

- la ocurrencia de una variable v con i llaves de protección y sea que $j \in N_o$ enumere, de la ocurrencia de la variable hacía afuera y comenzando en $j = 0$, los abstractores λv. .
- Esta ocurrencia es, con respecto al abstractor j-ésimo:

$$\rightarrow \textit{libre} \text{ Si } i > j, \quad \rightarrow \textit{acotada} \text{ Si } i = j, \quad \rightarrow \textit{sombreada} \text{ Si } i < j$$

- El índice de protección i de la ocurrencia de una variable, denota lo que se conoce como índice de ligado ó distancia de ligado (con respecto al abstractor).

Universidad Veracruzana

β-reducción con índices de protección

- El cuerpo de la abstracción e_b:
  - Decrementar el número i de llaves de protección, si aún quedan disponibles,
  - Se substituye por e_a, si la variable está acotada,
  - No se cambia nada, si la variable está sombreada.
- El operando e_a:
  - Incrementar el número i de llaves de protección en un valor k si la variable es libre o acotada con respecto al λv más interno que rodea la aplicación $(\lambda v.e_b\ e_a)$; y si la variable penetra el alcance de k abstractores anidados λv cuando es substituida en e_b,
  - Permanece intacta si la variable está sombreada.

Universidad Veracruzana

α-conversión con índices de protección

- Se puede transformar una expresión con variables acotadas de distinto nombre, en una que tenga solo variables, digamos z, aplicando la función de α-conversión $\lambda v.\lambda z.(v\ z)$ a todas las abstracciones de una variable.
- En el caso del ejemplo, la función se inserta en la expresión original:

$$\lambda v.\lambda z.(v\ z)\lambda u.\ldots$$

Ejemplo

$$(\lambda v.\lambda z.(v\ z)\ \lambda u.(\lambda v.\lambda z.(v\ z)\ \lambda v.\underline{(\lambda v.\lambda z.(v\ z)\ \lambda w.(((w\ v)\ u)\ (x\ u)))}))$$

$$\downarrow$$

$$(\lambda v.\lambda z.(v\ z)\ \lambda u.(\lambda v.\lambda z.(v\ z)\ \lambda v.\lambda z.\underline{(\lambda w.(((w\ v)\ u)\ (x\ u))\ z)}))$$

$$\downarrow$$

$$(\lambda v.\lambda z.(v\ z)\ \lambda u.\underline{(\lambda v.\lambda z.(v\ z)\ \lambda v.\lambda z.(((z\ v)\ u)\ (x\ u)))})$$

$$\downarrow$$

$$(\lambda v.\lambda z.(v\ z)\ \lambda u.\lambda z.\underline{(\lambda v.\lambda z.(((z\ v)\ u)\ (x\ u))\ z)})$$

$$\downarrow$$

$$\underline{(\lambda v.\lambda z.(v\ z)\ \lambda u.\lambda z.\lambda z.(((z\ /z)\ u)\ (x\ u)))}$$

$$\downarrow$$

$$\lambda z.\underline{(\lambda u.\lambda z.\lambda z.(((z\ /z)\ u)\ (x\ u))\ z)}$$

$$\downarrow$$

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

Secuencias de reducción

- Dadas dos λ-expresiones e y e', se dice que e es β-reducible a e', denotado por $e \longmapsto_\beta e'$, si y sólo si e puede ser transformado en e' por una secuencia finita (posiblemente vacía) de β-reducciones y α-conversiones.
- Esto produce una secuencia $e_0, \ldots, e_n$ de λ-expresiones con $e_0 =_s e$ y $e_n =_s e'$ tal que para todos los índices $i \in \{0, \ldots, n-1\}$ tenemos que $e_i \rightarrow_\beta e_{i+1}$ ó $e_i \rightarrow_\alpha e_{i+1}$.

Equivalencia sintáctica

- Dos λ-expresiones son semánticamente equivalentes, denotado por $e = e'$, si y sólo si e puede ser transformada en e' por una secuencia finita (posiblemente vacía) de β-reducciones, β-reducciones reversas y α-conversiones.
- Esto es, para todos los índices $i \in \{0, \ldots, n-1\}$, debemos tener $e_i \rightarrow_\beta e_{i+1}$ ó $e_{i+1} \rightarrow_\beta e_i$ ó $e_i \rightarrow_\alpha e_{i+1}$.

Forma Normal

- El objetivo de reducir una λ-expresión e es transformarla en alguna expresión e^{NF} que no contiene más redices, ó a la que no se le puedan aplicar más reglas de β-reducción.
- Esta expresión se conoce como la forma normal ó el valor de la expresión e. Si para llegar de e a e^{NF} necesitamos una secuencia finita de β-reducciones, entonces e^{NF} es también la forma normal de las λ-expresiones intermedias.

Universidad Veracruzana

Indeterminismo

- Una λ-expresión compleja que contiene diversos redices, generalmente lleva a la elección entre diferentes secuencias alternativas de β-reducciones. Tal que debemos alcanzar una forma normal
  - para eventualmente todas las posibles secuencias, es decir, tras muchas β-reducciones finitas ejecutadas en cualquier orden; si tenemos suerte;
  - para ninguna de las secuencias posibles, porque no existe una forma normal. Por ejemplo, ninguna de las secuencias termina en un número finito de pasos;
  - para algunas de las secuencias posibles, pero no para todas. Esto es, algunas secuencias terminan de manera finita, pero otras no.

Estrategias de Evaluación (1)

- Consideremos un ejemplo de la primer clase de expresión: $(\lambda u.(\lambda w.(\lambda w.u)\ u)\ w))$ donde:
  - procediendo de afuera hacía adentro:

$$(\lambda u.(\lambda w.(\lambda w.u)\ u)\ w)) \rightarrow_\beta (\lambda w.(\lambda(w.//w\ /w)w) \rightarrow_\beta (\lambda w./w\ w) \rightarrow_\beta w$$

  - procediendo de adentro hacía afuera:

$$(\lambda u.(\lambda w.(\lambda w.u)\ u)\ w)) \rightarrow_\beta (\lambda u.(\lambda w.u\ u)\ w) \rightarrow_\beta (\lambda u.u\ w) \rightarrow_\beta w$$

  - y aún si comenzásemos por el radice de en enmedio, llegaríamos a la forma normal w.

Universidad Veracruzana

Estrategias de Evaluación (2)

- Ahora, una expresión simple que no tiene forma normal es la auto-aplicación:

$$(\lambda u.(u\ u)\ \lambda u.(u\ u)) \rightarrow_\beta (\lambda u.(u\ u)\ \lambda u.(u\ u)) \rightarrow_\beta \dots$$

que incesantemente se reproduce a si misma.

Estrategias de Evaluación (3)

- Observen la siguiente aplicación:

$$((\lambda w.\lambda v.u\ \lambda w.w) \quad (\lambda u.(u\ u)\ \lambda u.(u\ u)))$$

  la abstracción $\lambda w.\lambda v.u$ es una función selector que reproduce su primer argumento, es decir $\lambda w.w$, pero elimina el segundo, que en este caso particular es la misma aplicación.

- De forma que una reducción de afuera hacía adentro lleva a w, pero una de afuera hacía adentro no termina.

Universidad Veracruzana

Reducción en orden normal

- Esta estrategia debe ser definida por una función de transformación τ_N que mapea λ-expresiones a λ-expresiones como sigue:

$$\tau_N(e) = \begin{cases} v & \text{Si } e =_s v \in V \\ \lambda v.\tau_N(e_b) & \text{Si } e =_s \lambda v.e_b \\ \tau'_N(e) & \text{Si } e =_s (\lambda v.e_b\ e_2) \\ \tau'_N(\tau_N(e_1)e_2) & \text{Si } e =_s (e_1\ e_2) \text{ y } e_1 \neq_s \lambda v.e_b \end{cases}$$

- donde:

$$\tau'_N(e) = \begin{cases} \tau_N(e_b[v \leftarrow e_2]) & \text{Si } e =_s (\lambda v.e_b\ e_2) \\ (e_1\ \tau_N(e_2)) & \text{Si } e =_s (e_1\ e_2) \text{ y } e_1 \neq_s \lambda v.e_b \end{cases}$$

Evaluador abstracto

- La función τ_N define un evaluador abstracto para expresiones del Cálculo-λ puro, similar al evaluador `EVAL` del capítulo anterior.
- A diferencia de `EVAL`, τ_N solo se propaga recursivamente a través de las expresiones operador, pero no toca los operandos hasta que el operador es procesado y no es una abstracción (el último caso en τ'_N. Si es una abstracción, entonces el operando es substituido por ocurrencias libres de la variable ligada (el primer caso de τ'_N).

Llamadas por nombre y por valor

- La reducción en orden normal también se conoce como de afuera a adentro y de izquierda a derecha.
- Esta estrategia se conoce también como llamada por nombre y es usada por lenguajes de programación como Algol y Simula.
- Es posible definir una estrategias como la usada por `EVAL`, conocida como primero operandos o llamada por valor.

Universidad Veracruzana

Teorema de Church-Rosser

- Sean e_0, e_1, e_2, e_3 λ-expresiones. Entonces $e_0 \longmapsto_\beta e_1$ y $e_0 \longmapsto_\beta e_2$ implican que existe una expresión e_3 tal que $e_1 \longmapsto_\beta e_3$ y $e_2 \longmapsto_\beta e_3$.
- La prueba está más allá del contenido de este curso, pero observen que si e_3 es una forma normal, entonces esta forma es única.

Bibliografía

W. Kluge.
Abstract Computing Machines: A Lambda Calculus Perspective.
Springer-Verlag, Berlin Heidelberg New York, 2005.