Tipos de datos en Cálculo-λ puro

José de Jesús Lavalle Martínez

Benemérita Universidad Autónoma de Puebla
Facultad de Ciencias de la Computación
Fundamentos de Lenguajes de Programación CCOS 255

Primavera 2020

Contenido

Tesis de Church I

- Abstracción, aplicación y reducción-β (vía sustitución) forman la esencia del cálculo-λ.

Tesis de Church I

- A pesar de que el cálculo-λ parece muy simple a simple vista, es suficientemente expresivo para codificar todos los algoritmos. Esto está expresado por la tesis de Church (1930).

  Todo lo que sea intuitivamente computable es definible en el cálculo-λ.

Tesis de Church I

- Si cualquier función computable, o todo algoritmo, se puede representar en el cálculo-λ, entonces es posible ver a un programa como un término-λ.

Tesis de Church I

- Los lenguajes de programación que están basados sobre el cálculo-λ se llaman **lenguajes de programación funcionales**, el término *funcional* se debe a que un programa en estos lenguajes es una función y un cálculo se hace evaluando una función. Algunos lenguajes funcionales son LISP, ML, Haskell y Miranda.

Tesis de Church II

- No es posible demostrar formalmente la tesis de Church porque se basa en una *idea intuitiva* de lo que es computable.

Tesis de Church II

- Es posible refutarla dando una función que es intuitivamente computable y probando que no es definible en el cálculo-λ. Pero esto no ha ocurrido hasta ahora y no se espera que ocurra.

Tesis de Church II

- Por otro lado, tampoco existe una definición precisa de que es un lenguaje de programación.

Tesis de Church II

- Pero al menos nos gustaría poder expresar tipos de datos elementales tales como números naturales, booleanos y pares ordenados.

¿Qué significa definible en el cálculo-λ? I

Empecemos considerando a los número naturales.

- Una representación de los números naturales como términos-λ debe tener la propiedad de que la representación de n es distinta a la representación de m si $n \neq m$.

¿Qué significa definible en el cálculo-λ? I

Empecemos considerando a los número naturales.

- Es deseable representar un número natural mediante una forma normal, esto es, un término que no se puede reducir más.

¿Qué significa definible en el cálculo-λ? I

Empecemos considerando a los número naturales.

- La representación de un número natural debe ser un término cerrado, para que ninguna variable libre pueda ser capturada al ponerla en un contexto más grande. Recuerde que una caraterística de las funciones es que al evaluarlas en un argumento siempre darán el mismo resultado independientemente del contexto al que pertenezcan.

¿Qué significa definible en el cálculo-λ? II

- De lo anterior una representación de los números naturales en el cálculo-λ es una secuencia infinita de formas normales cerradas distintas.

¿Qué significa definible en el cálculo-λ? II

- La representación de los números naturales también nos debe servir para hacer cálculos con ellos.

¿Qué significa definible en el cálculo-λ? II

- Para lograr esto, la secuencia infinita de formas normales cerradas debe ser sistemática en algún sentido.

Definición de Definible

Definición 1

Asuma que sabemos como representar a los número naturales en el cálculo-λ. Escribamos $[n]$ para referirnos al término-λ cerrado que representa al número natural n. Decimos que una función $f : nat \to nat$ es **definible** en el cálculo-λ si existe un término-λ $[f]$ tal que:

$$[f][n] =_\beta [f(n)]$$

para todo número natural n.

Es decir, la codificación de la función f aplicada a la codificación del número natural n es equivalente-β a la codificación de $f(n)$

Equivalencia-β

Observación 2

- La igualdad usada ($=_\beta$) se llama equivalencia-β, también llamada convertibilidad-β.

Equivalencia-β

Observación 2

- Equivalencia-β es la cerradura reflexiva, simétrica y transitiva de reducción-β.

Equivalencia-β

Observación 2

- Los términos M y N son convertibles-β si existe una secuencia finita de reducciones-β, ya sea de izquierda a derecha o de derecha a izquierda entre M y N.

Observación 2

- Intuitivamente, M y N representan el mismo objeto (como en $3^2, 9$ y $16 - 7$, los tres representan al mismo objeto).

Ejemplo equivalencia-β

Ejemplo 3

$$(\lambda xy.x)(\lambda x.x)(\lambda xy.x) =_\beta (\lambda xy.y)(\lambda xy.x)(\lambda x.x)$$

Solución:

$$\begin{aligned}(\lambda xy.x)(\lambda x.x)(\lambda xy.x) &\to_\beta (\lambda y.(\lambda x.x))(\lambda xy.x)\\ &\to_\beta (\lambda x.x)\\ &\leftarrow_\beta (\lambda y.y)(\lambda x.x)\\ &\leftarrow_\beta (\lambda xy.y)(\lambda xy.x)(\lambda x.x)\end{aligned}$$

Booleanos

¿Qué términos-λ representarán a *true* y *false*?

- Muy difícil responder directamente, ya que en los lenguajes de programación que conocemos son constantes dadas.

Booleanos

¿Qué términos-λ representarán a *true* y *false*?

- Muy difícil responder directamente, ya que en los lenguajes de programación que conocemos son constantes dadas.
- Es mejor responder especificando formalmente el comportamiento que deben tener *true* y *false* en algún contexto que nos sea familiar.

Booleanos

¿Qué términos-λ representarán a *true* y *false*?

- Muy difícil responder directamente, ya que en los lenguajes de programación que conocemos son constantes dadas.
- Es mejor responder especificando formalmente el comportamiento que deben tener *true* y *false* en algún contexto que nos sea familiar.
- En este caso usaremos el operador condicional (if-then-else) como contexto para especificar el comportamiento que deben exhibir y así poder definirlos.

ite, true y false

En cierto sentido tenemos el siguiente par de "ecuaciones" que queremos resolver:

$$\mathit{ite}\ \ \mathit{true}\ P\ Q \to_\beta^* P$$

$$\mathit{ite}\ \ \mathit{false}\ P\ Q \to_\beta^* Q$$

ite, true y false

En cierto sentido tenemos el siguiente par de "ecuaciones" que queremos resolver:

$$\textit{ite}\ \ \textit{true}\ P\ Q \to_\beta^* P$$
$$\textit{ite}\ \ \textit{false}\ P\ Q \to_\beta^* Q$$

Así,

1. *true* procesará dos parámetros y siempre regresará el primero, borrando o ignorando el segundo.

ite, true y false

En cierto sentido tenemos el siguiente par de "ecuaciones" que queremos resolver:

$$\mathit{ite}\ \ \mathit{true}\ P\ Q \to_\beta^* P$$
$$\mathit{ite}\ \ \mathit{false}\ P\ Q \to_\beta^* Q$$

Así,

1. *true* procesará dos parámetros y siempre regresará el primero, borrando o ignorando el segundo.
2. *false* procesará dos parámetros y siempre regresará el segundo, borrando o ignorando el primero.

ite, true y false

En cierto sentido tenemos el siguiente par de "ecuaciones" que queremos resolver:

$$\textit{ite}\ \ \textit{true}\ P\ Q \to_\beta^* P$$
$$\textit{ite}\ \ \textit{false}\ P\ Q \to_\beta^* Q$$

Así,

1. *true* procesará dos parámetros y siempre regresará el primero, borrando o ignorando el segundo.
2. *false* procesará dos parámetros y siempre regresará el segundo, borrando o ignorando el primero.
3. Un término-λ que procesa dos parámetros se puede escribir como $(\lambda x.(\lambda y.))$ o abreviando $(\lambda xy.)$, el primer parámetro que procesará será x y el segundo será y, por lo tanto.

ite, true y false

En cierto sentido tenemos el siguiente par de "ecuaciones" que queremos resolver:

$$\textit{ite}\ \ \textit{true}\ P\ Q \rightarrow_\beta^* P$$
$$\textit{ite}\ \ \textit{false}\ P\ Q \rightarrow_\beta^* Q$$

Así,

1. *true* procesará dos parámetros y siempre regresará el primero, borrando o ignorando el segundo.
2. *false* procesará dos parámetros y siempre regresará el segundo, borrando o ignorando el primero.
3. Un término-λ que procesa dos parámetros se puede escribir como $(\lambda x.(\lambda y.))$ o abreviando $(\lambda xy.)$, el primer parámetro que procesará será x y el segundo será y, por lo tanto.
   - $\textit{true} = (\lambda xy.x)$,
   - $\textit{false} = (\lambda xy.y)$,
   - De paso, $\textit{ite} = (\lambda bxy.bxy), b \in \{\textit{true}, \textit{false}\}$.

Negación

¿Cómo definimos la negación?

Negación

¿Cómo definimos la negación?

Tenemos que resolver las "ecuaciones":

$$\mathit{not}\ \mathit{true} \rightarrow_\beta^* \mathit{false}$$

$$\mathit{not}\ \mathit{false} \rightarrow_\beta^* \mathit{true}$$

Negación

¿Cómo definimos la negación?

Tenemos que resolver las "ecuaciones":

$$not\ true \rightarrow_\beta^* false$$
$$not\ false \rightarrow_\beta^* true$$

Recordando que *true* procesa dos parámetros regresando el primero e ignorando el segundo, que *false* procesa dos parámetros regresando el segundo e ignorando el primero, tenemos:

$$true\ false\ true \rightarrow_\beta^* false$$
$$false\ false\ true \rightarrow_\beta^* true$$

Negación

¿Cómo definimos la negación?

Tenemos que resolver las "ecuaciones":

$$\textit{not true} \rightarrow_\beta^* \textit{false}$$
$$\textit{not false} \rightarrow_\beta^* \textit{true}$$

Recordando que *true* procesa dos parámetros regresando el primero e ignorando el segundo, que *false* procesa dos parámetros regresando el segundo e ignorando el primero, tenemos:

$$\textit{true false true} \rightarrow_\beta^* \textit{false}$$
$$\textit{false false true} \rightarrow_\beta^* \textit{true}$$

Por lo tanto,

$$\textit{not} = (\lambda b.b\ \textit{false true}), b \in \{\textit{true}, \textit{false}\}.$$

Conjunción

¿Cómo definimos la conjunción?

Tenemos que resolver las "ecuaciones":

$$\begin{aligned} \mathit{and}\ \mathit{true}\ \mathit{true} &\to_\beta^* \mathit{true} \\ \mathit{and}\ \mathit{true}\ \mathit{false} &\to_\beta^* \mathit{false} \\ \mathit{and}\ \mathit{false}\ \mathit{true} &\to_\beta^* \mathit{false} \\ \mathit{and}\ \mathit{false}\ \mathit{false} &\to_\beta^* \mathit{false} \end{aligned}$$

De las dos primeras "ecuaciones',' si el primer parámetro es *true* el resultado depende del segundo parámetro; de las dos últimas, si el primer parámetro es *false*, el resultado es *false* sin importar el segundo parámetro. Por lo tanto,

$$\mathit{and} = (\lambda xy.x\ y\ \mathit{false}).$$

Construcción de pares I

¿Cómo definimos pares?

Nuevamente a través del comportamiento que deben exhibir en algún contexto. El contexto en este caso serán dos propiedad de un par π, las propiedades son que se puede recuperar tanto el primero como el segundo elemento de un par, mediante las proyecciones π_1 y π_2. Concretamente se deben resolver las "ecuaciones" siguientes:

$$\pi_1(\pi PQ) \to_\beta^* P$$
$$\pi_2(\pi PQ) \to_\beta^* Q$$

Fijémonos en la primera "ecuación" y recordemos que un término-λ que procesa dos argumentos y regresa el primero ignorando el segundo es $(\lambda lr.l)$.

Construcción de pares II

$$
\begin{aligned}
P &\leftarrow_\beta (\lambda r.P)Q \\
&\leftarrow_\beta (\lambda lr.l)PQ \\
&\leftarrow_\beta (\lambda z.zPQ)(\lambda lr.l) \\
&\leftarrow_\beta (\lambda u.u(\lambda lr.l))(\lambda z.zPQ)
\end{aligned}
$$

De aquí obtenemos que $\pi_1 = (\lambda u.u(\lambda lr.l))$.

Construcción de pares III

¿Cómo obtenemos $(\lambda z.zPQ)$?

$$\begin{aligned}(\lambda z.zPQ) &\leftarrow_\beta (\lambda rz.zPr)Q \\ &\leftarrow_\beta (\lambda lrz.zlr)PQ\end{aligned}$$

Así, siguiendo un razonamiento similar para π_2 tenemos que:

$$\begin{aligned}\pi &= (\lambda lrz.zlr), \\ \pi_1 &= (\lambda u.u(\lambda lr.l)), \\ \pi_2 &= (\lambda u.u(\lambda lr.r)).\end{aligned}$$

Numerales de Church I

Para definir los numerales de Church usaremos la composición de funciones. Si F y P son términos-λ, la aplicación de F a F aplicado a P la escribimos como $F^2(P)$ en lugar de $F(FP)$ y la iteración de tres aplicaciones de F a P como $F^3(P)$ en lugar de $F(F(FP))$, etcetera. En general tenemos:

Numerales de Church I

Para definir los numerales de Church usaremos la composición de funciones. Si F y P son términos-λ, la aplicación de F a F aplicado a P la escribimos como $F^2(P)$ en lugar de $F(FP)$ y la iteración de tres aplicaciones de F a P como $F^3(P)$ en lugar de $F(F(FP))$, etcetera. En general tenemos:

$$F^0(P) = P,$$
$$F^{n+1}(P) = F(F^n(P)).$$

Numerales de Church II

Definición 4

Para todo número natural n el numeral de Church c_n se define como sigue:

$$c_n = (\lambda s.(\lambda z.s^n(z))) = \lambda s.\lambda z.s^n(z) = \lambda sz.s^n(z)$$

Numerales de Church II

Definición 4

Para todo número natural n el numeral de Church c_n se define como sigue:

$$c_n = (\lambda s.(\lambda z.s^n(z))) = \lambda s.\lambda z.s^n(z) = \lambda sz.s^n(z)$$

Los primeros cinco numerales de Church son los siguientes:

$$\begin{aligned} c_0 &= \lambda s.\lambda z.z, \\ c_1 &= \lambda s.\lambda z.sz, \\ c_2 &= \lambda s.\lambda z.s(sz), \\ c_3 &= \lambda s.\lambda z.s(s(sz)), \\ c_4 &= \lambda s.\lambda z.s(s(s(sz))). \end{aligned}$$

Suc I

¿Cómo definimos el sucesor de un numeral de Church?

Especificando su comportamiento de la siguiente manera:

$$\mathit{Suc}\ c_n \rightarrow_\beta^* c_{n+1}$$

Suc I

¿Cómo definimos el sucesor de un numeral de Church?

Especificando su comportamiento de la siguiente manera:

$$\mathit{Suc}\ c_n \to_\beta^* c_{n+1}$$

Observe que:

$$\begin{aligned} c_n s z &= (\lambda s' z' . s'^n(z'))sz \\ &\to_\beta (\lambda z' . s^n(z'))z \\ &\to_\beta s^n(z). \end{aligned}$$

Tan importante que la vamos a numerar:

$$c_n s z = s^n(z). \tag{1}$$

Suc I

¿Cómo definimos el sucesor de un numeral de Church?

Especificando su comportamiento de la siguiente manera:

$$\mathit{Suc}\ c_n \to_\beta^* c_{n+1}$$

Observe que:

$$\begin{aligned} c_n s z &= (\lambda s' z' . s'^n(z'))sz \\ &\to_\beta (\lambda z' . s^n(z'))z \\ &\to_\beta s^n(z). \end{aligned}$$

Tan importante que la vamos a numerar:

$$c_n s z = s^n(z). \tag{1}$$

De manera similar tenemos:

$$\begin{aligned} s(c_n s z) &\to_\beta^* s(s^n(z)) \\ &= s^{n+1}(z). \end{aligned}$$

Suc II

$$\begin{aligned} s(c_n s z) &\to_\beta^* s(s^n(z)) \\ &= s^{n+1}(z). \end{aligned}$$

Esto significa que hay un término-λ que tiene a c_n como subtérmino y que se reduce a c_{n+1}.

Suc II

$$s(c_n sz) \rightarrow_\beta^* s(s^n(z)) = s^{n+1}(z).$$

Esto significa que hay un término-λ que tiene a c_n como subtérmino y que se reduce a c_{n+1}. Este término-λ es $\lambda sz.s(c_n sz)$ y su reducción-β es como sigue:

$$\lambda sz.s(c_n sz) \rightarrow_\beta^* \lambda sz.s^{n+1}(z) = c_{n+1}.$$

Suc II

$$s(c_n sz) \rightarrow_\beta^* s(s^n(z))$$
$$= s^{n+1}(z).$$

Esto significa que hay un término-λ que tiene a c_n como subtérmino y que se reduce a c_{n+1}. Este término-λ es $\lambda sz.s(c_n sz)$ y su reducción-β es como sigue:

$$\lambda sz.s(c_n sz) \rightarrow_\beta^* \lambda sz.s^{n+1}(z)$$
$$= c_{n+1}.$$

Ahora debemos definir un término-λ que una vez aplicado a c_n se reduzca al término $\lambda sz.s(c_n sz)$, así obtenemos finalmente que:

Suc II

$$s(c_n sz) \rightarrow_\beta^* s(s^n(z)) = s^{n+1}(z).$$

Esto significa que hay un término-λ que tiene a c_n como subtérmino y que se reduce a c_{n+1}. Este término-λ es $\lambda sz.s(c_n sz)$ y su reducción-β es como sigue:

$$\lambda sz.s(c_n sz) \rightarrow_\beta^* \lambda sz.s^{n+1}(z) = c_{n+1}.$$

Ahora debemos definir un término-λ que una vez aplicado a c_n se reduzca al término $\lambda sz.s(c_n sz)$, así obtenemos finalmente que:

$$\mathit{Suc} = \lambda x.\lambda sz.s(xsz) = \lambda xsz.s(xsz).$$

Plus I

$$\mathit{Plus}\ c_m c_n \to_\beta^* c_{m+n}.$$

$$\begin{aligned} c_{m+n} &= \lambda sz.s^{m+n}(z) \\ &= \lambda sz.s^m(s^n(z)) \\ &\overset{(1)}{=} \lambda sz.s^m(c_n sz) \\ &\overset{(1)}{=} \lambda sz.c_m s(c_n sz) \end{aligned}$$

Nuevamente, lo único que nos hace falta es un término-λ que aplicado a c_m y c_n se evalue a c_{m+n}, éste es:

$$\mathit{Plus} = \lambda xy.\lambda sz.xs(ysz)$$

Mul y Exp

$$\mathit{Mul}\ c_m c_n \rightarrow_\beta^* c_{m \times n}$$
$$\mathit{Exp}\ c_m c_n \rightarrow_\beta^* c_{m^n}$$

$$\mathit{Mul} = \lambda xy.\lambda s.x(ys)$$
$$\mathit{Exp} = \lambda xy.yx$$

Listas

$$\mathit{head}\ (\mathit{cons}\ HT) \rightarrow_\beta^* H$$
$$\mathit{tail}\ (\mathit{cons}\ HT) \rightarrow_\beta^* T$$

$$\begin{aligned}
\mathit{nil} &= \lambda xy.y \\
\mathit{cons} &= \lambda ht.\lambda z.zht \\
\mathit{head} &= \lambda l.l(\lambda ht.h) \\
\mathit{tail} &= \lambda l.l(\lambda ht.t)
\end{aligned}$$

Ejercicios I

1. Implemente `ronaster` de tal manera que regrese una lista con los términos-λ que ha reducido numerados, empezando con el que originalmente se llamo y finalizando con la forma normal beta que calcula. Como ejemplo tenemos la siguiente invocación con el caso de prueba 13 para las estrategias de reducción.

```
- ronaster(a(a(l(x 5, l(x 1, a(a(v(x 5), v(x 1)), v(x 1)))), v(x 2)), a(l(x 3, v(x 3)), v(x 4))));
> val it =
[(0, a(a(l(x 5, l(x 1, a(a(v(x 5), v(x 1)), v(x 1)))), v(x 2)), a(l(x 3, v(x 3)), v(x 4)))),
(1, a(l(x 1, a(a(v(x 2), v(x 1)), v(x 1))), a(l(x 3, v(x 3)), v(x 4)))),
(2, a(a(v(x 2), a(l(x 3, v(x 3)), v(x 4))), a(l(x 3, v(x 3)), v(x 4)))),
(3, a(a(v(x 2), v(x 4)), a(l(x 3, v(x 3)), v(x 4)))),
(4, a(a(v(x 2), v(x 4)), v(x 4)))] : (int * t) list
```

   Note lo siguiente:

   - `> val ronaster = fn :  t -> (int * t) list`
   - Al término con el que invocamos a la función `ronaster` le corresponde el número 0 en la secuencia de reducciones.
   - Use la técnica del parámetro acumulante para formar la secuencia de reducciones.
   - En ML un par se escribe poniendo entre paréntesis cada elemento del par separado por coma, por ejemplo.

Ejercicios II

```
- (3, l(x 1, v(x 1)));
> val it = (3, l(x 1, v(x 1))) : int * t
```

2. Defina un término-λ para la disyunción siguiendo la metodología de la sección Booleanos.
3. Implemente todos los tipos de datos vistos en esta presentación y sus respectivas operaciones.
4. Pruebe cada operación con un caso de prueba de su elección.