CÁLCULO LAMBDA

Habíamos definido el lenguaje de las expresiones (términos lambda), FV y la sustitución.

```
<exp> ::=               {término lambda, o expresión}
        <var>           {variable}
      | <exp> <exp>     {aplicación, el primero es el operador y el segundo el operando}
      | λ<var>.<exp>    {abstracción o expresión lambda}
```

La aplicación asocia a izquierda. En λv.e, la primer ocurrencia de v es ligadora y su alcance es e.

```
FV(v) = {v}
FV(e0 e1) = FV(e0) ∪ FV(e1)
FV(λv.e) = FV(e) - {v}


Δ = <var> -> <exp>
_/_ ∈ <exp> x Δ -> <exp>

v/d = d v
(e0 e1)/d = (e0/d) (e1/d)
(λv.e)/d = λv'.(e/[d|v:v'])
```

Prop

(a) si para todo w ∈ FV(e) d w = d' w entonces (e/d) = (e/d')
(b) sea i la sustitución identidad, entonces e/i = e
(c) FV(e/d) = ∪{FV(d w) | w ∈ FV(e)}

Escribimos e/v->e' en vez de e/[i|v:e'] y e/v1->e1,..,vn->en en vez de e/[i|v1:e1|..|vn:en].

RENOMBRE

La operación de cambiar una ocurrencia de la expresión lambda λv.e por λv'.(e/v->v') donde v' ∉ FV(e) - {v} se llama renombre o cambio de variable ligada. Si e' se obtiene a partir de e por 0 o más renombres de ocurrencias de subfrases, se dice que e' se obtiene a partir de e por renombre. También se dice que e α-convierte a e'. Se puede ver que ésta es una relación de equivalencia. Se dice que e y e' son α-equivalentes y se escribe e ≡ e'.

Se supone que renombre debe preservar la semántica (aunque hubo algunas excepciones en los primeros lenguajes funcionales).

REDUCCIÓN

Una expresión de la forma (λv.e) e' se llama redex (plural redices, a veces mal dicho rédexes). Es la aplicación de una función (λv.e) a su argumento (e'). Debe calcularse reemplazando las ocurrencias libres de v en e por e', es decir (e/v->e').

Cuando en una expresión e0 se reemplaza una ocurrencia de un redex (λv.e) e' por (e/v->e') y luego de 0 o más renombres se obtiene e1, se dice que e0 β-contrae a e1. En realidad no neceseriamente e1 va a ser más pequeña que e0. Por ejemplo e0 = (λx.xxxx) (λy.λz.y z) β-contrae a (λy.λz.y z) (λy.λz.y z) (λy.λz.y z) (λy.λz.y z) que es más grande.

Se define como en la semántica de transiciones del lenguaje imperativo

```
Γ = conjunto de configuraciones
Γ = <exp>
Γ_t = {expresiones sin redex}
Γ_n = {expresiones con redex}
-> ⊆ Γ_n x Γ
```

regla β-reducción o reducción β:

```
----------------------
(λv.e) e' -> (e/v->e')
```

regla renombre:

```
e0 -> e1     e1 ≡ e1'
----------------------
      e0 -> e1'
```

regla clausura contextual

```
 e0 -> e1
----------
e0' -> e1'
```

donde e1' se obtiene a partir de e0' reemplazando una ocurrencia de e0 por e1.

Como antes se define la clausura reflexiva y transitiva -*->, sólo cambia la última regla que incluye la posibilidad de renombre:

```
 e -> e'
---------
e -*-> e'

e -*-> e'    e' -*-> e''
------------------------
       e -*-> e''

 e ≡ e'
---------
e -*-> e'
```

Cuando e -*-> e' se dice que e reduce a e'.

Una configuración terminal es una expresión que no tiene redex. A una tal expresión se la llama forma normal. Si e reduce a una forma normal e', e' es una forma normal de e. Existen varias otras nociones de redex, contracción, reducción, forma normal, etc. Cuando es necesario distinguir se dice β-redex, β-contracción, β-reducción, β-forma normal.

Las siguientes reducciones terminan en formas normales:

(λx.y) (λz.z) -> y

(λx.x) (λz.z) -> (λz.z)

(λx.xx) (λz.z) -> (λz.z) (λz.z) -> (λz.z)

(λx.(λy.yx)z) (zw) -> (λx.zx) (zw) -> z(zw)

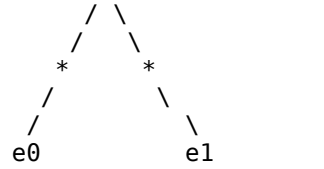
(λx.(λy.yx)z) (zw) -> (λy.y(zw))z -> z(zw)

Las últimas dos comienzan con la misma expresión, se diferencian, pero cuando llegan a la forma normal dan el mismo resultado. Es consecuencia del

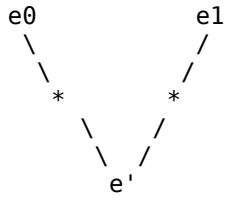
Teorema de Church-Rosser: Si e -*-> e0 y e -*-> e1 entonces existe e' tal que e0 -*-> e' y e1 -*-> e'.

También se la llama confluencia o propiedad del diamante:

Si



entonces



Corolario: Salvo renombre, toda expresión tiene a lo sumo una forma normal.
Dem: Sea e, supongamos que e0 y e1 son formas normales de e. Tenemos e-*-> e0 y e-*-> e1. Por teorema, existe e' tal que e0 -*-> e' y e1 -*-> e'. Pero como e0 y e1 eran formas normales no tenían redices, por lo tanto e' sólo puede ser renombre de e0 y de e1, por lo tanto e0 y e1 son renombres entre sí.

A pesar de este resultado, no todo término tiene forma normal:

Sea Δ = (λx.xx).
Δ Δ = (λx.xx) (λx.xx) -> (λx.xx) (λx.xx) = Δ Δ
-> ...

(λx.xxy) (λx.xxy) -> (λx.xxy) (λx.xxy) y
-> (λx.xxy) (λx.xxy) y y
-> ...

Sea Y1 = (λx.f(xx)), Y2 = Y1 Y1
Y2 = (λx.f(xx)) (λx.f(xx)) -> f ((λx.f(xx)) (λx.f(xx))) = f Y2
-> f (f ((λx.f(xx)) (λx.f(xx)))) = f (f Y2)
-> ...

Ninguna de estas tiene forma normal.

Por otro lado, hay situaciones mixtas: que tienen forma normal pero puede no encontrarse:

(λx.λy.y) (Δ Δ) -> λy.y
(λx.λy.y) (Δ Δ) -> (λx.λy.y) (Δ Δ) -> ...

¿Dada una expresión, hay alguna manera de elegir una secuencia de reducciones que va a terminar siempre que sea posible?

Secuencia de reducciónes en orden normal (o reducción en orden normal): es una reducción que en cada paso contrae el redex (externo) que se encuentre más a la izquierda.

La reducción en orden normal en cada paso contrae el redex que se encuentre más a la izquierda. A veces un redex puede encontrarse dentro de otro redex. Puede observarse, que de esos dos, el externo comienza más a la izquierda que el interno. Si asumimos que eso significa que el externo está más a la izquierda que el interno, no hace falta pensar en redex externos e internos. La definición de arriba usa la palabra externo entre paréntesis para remarcar que no es imprescindible usarla, basta con referirse al que se encuentra más a la izquierda si por ello entendemos comenzar más a la izquierda.

Teorema de Estandarización: Si hay alguna secuencia de reducciones a partir de e que termina, entonces la secuencia de reducciones en orden normal a partir de e termina.

AÑO 2011: Material para el práctico

EXPRESIVIDAD DEL CÁLCULO LAMBDA

Veamos algo sobre la expresividad del cálculo lambda. De la misma manera que aceptamos que los booleanos pueden representarse con el 0 y el 1, la disyunción con el máximo y la conjunción con el mínimo (o producto), podemos comprobar que se pueden representar con términos lambda si las propiedades habituales de los booleanos se respetan.

Por ejemplo:

T = λx.λy.x

F = λx.λy.y

son posibles definiciones para verdadero y falso respectivamente. Podemos definir el if then else:

if b then t else e = b t e

En efecto, si reemplazamos b por el término T definido más arriba, resulta que if T then t else e -> t; y si lo reemplazamos por F, resulta que if F then t else e -> e.

Podemos definir entonces

if_then_else_ = λb.λt.λe.b t e

Ahora podemos definir

not b = if b then F else T
b or c = if b then T else c
b and c = if b then c else F

O sea,

not = λb.b F T
or = λb.λc.b T c
and = λb.λc.b c F

Hay otras variantes posibles, por ejemplo,

not = λb.λx.λy.b y x