CÁLCULO LAMBDA

Motivación: notación para no necesitar nombrar funciones. Por ejemplo, x+y puede ser:

f(x) = x+y
g(y) = x+y
h(x,y) = x+y

La notación lambda permite expresar sin dar nombre:

λx.x+y
λy.x+y
λ(x,y).x+y

e incluso

λx.λy.x+y
λy.λx.x+y

En algunos contextos, los matemáticos usan notaciones similares:

En integrales, ∫x+y.dx enfatiza que x varía e y está fija. En ∑_{i∈{0..10}} i+j, i varía y j está fija.

El cálculo lambda, en principio es una notación para no tener que nombrar funciones. Pero además, el cálculo lambda puro, sólo contiene variables, aplicaciones y la notación lambda (llamada abstracción).

```
<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.

Por ejemplo, λx.(λy.xyx)x es lo mismo que λx.((λy.((xy)x))x)

Se hacen exactamente las mismas definiciones de ocurrencia ligada, ocurrencia libre y variable libre. El conjunto de variables libres se define también por inducción en la estructura de los términos:

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

También se define sustitución:

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'])

donde v' ∉ ∪{FV(d w) | w ∈ FV(e) - {v}}

La sustitución en el cálculo lambda es fundamental: permite definir la semántica operacional.

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 (α)

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).

CONTRACCIÓN (β)

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 y se escribe e0 -> e1. En realidad no necesariamente 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.

Una expresión sin redices se llama forma normal.

Ejemplos:

(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 expresiones de cada línea son formas normales.

Los últimos dos ejemplos comienzan con la misma expresión, se diferencian, pero cuando llegan a la forma normal dan el mismo resultado.

Utilizando la notación de la semántica de transiciones para el lenguaje imperativo tendríamos

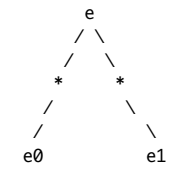
Γ = conjunto de configuraciones
Γ = <exp>
Γ_t = {expresiones sin redex / en forma normal}
Γ_n = {expresiones con redex}
-> ⊆ Γ_n x Γ

y como se vió en alguno de los ejemplos de arriba -> es no determinístico. Sin embargo, si -*-> es la clausura reflexiva (y por renombre) y transitiva de ->.

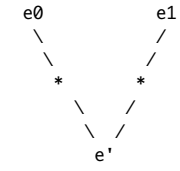
Teorema de Church-Rosser: Si e -*-> e0 y e -*-> e1 entonces existe e' tal que e0 -*-> e' y e1 -*-> e'.

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.

Pero no toda expresión tiene forma normal:

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

Sea Δ' = (λx.xxy).
Δ' Δ' = (λx.xxy) Δ' -> Δ' Δ' y -> Δ' Δ' y y -> ...

Y hay casos en las que la forma normal sólo se encuentra si se "ejecuta bien":

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

EVALUACION

Hemos presentado la relación ->, que permite realizar cómputos, ejecuciones de un término lambda hasta llevarlo a su forma normal en caso de que la tenga. Pero ésa no es exactamente la idea de ejecución que implementan los lenguajes aplicativos. La idea de ejecución (llamada evaluación) que se implementa habitualmente tiene las siguientes diferencias con la relación ->:

1) sólo se evalúan expresiones cerradas (es decir, sin variables libres)
2) es determinística
3) no busca formas normales sino formas canónicas

Existen varias definiciones de evaluación. Estudiaremos las dos más importantes: la evaluación (en orden) normal y la evaluación eager o estricta. La primera es la que ha dado lugar a los lenguajes de programación funcionales lazy (Haskell) y la segunda, a los estrictos (ML).

Como la evaluación busca una forma canónica de un término, las formas canónicas juegan el rol de ser "valores" de expresiones.

La noción de forma canónica depende de la definición de evaluación. Se define una noción de forma canónica para la evaluación normal, y otra para la evaluación eager. En el caso del cálculo lambda coinciden: son las abstracciones (en otro lenguajes, veremos, dejan de coincidir).

Entonces tenemos: formas canónicas = abstracciones.

Como a partir de ahora nos concentraremos en expresiones cerradas, la siguiente propiedad ayuda para comparar las nociones de forma canónica y forma normal:

Prop: Una aplicación cerrada no puede ser forma normal.
Dem: Sea e una aplicación cerrada. Sea e = e0 e1 .. en donde e0 no es una aplicación. Como e es una aplicación, n ≥ 1. Si e0 fuera una variable, e no sería cerrada. Por lo tanto, es una abstracción y e contiene el redex e0 e1. Por lo tanto no es forma normal.

Entonces, podemos concluir que una expresión cerrada que es forma normal es también forma canónica. El recíproco no vale, por ejemplo λx.(λy.y) x es forma canónica cerrada pero no es forma normal. Peor aún, λx.Δ Δ es forma canónica cerrada y no tiene forma normal.

¿Por qué conformarse con una forma canónica en vez de continuar ejecutando hasta obtener una forma normal? Podemos entenderlo de la siguiente manera: A) es razonable circunscribir la atención a expresiones cerradas (punto 1, apenas empezamos a hablar de evaluación) porque son las que parecen tener sentido, B) una vez que se alcanzó una abstracción λv.M, continuar evaluando implicaría evaluar M que puede no ser una expresión cerrada, puede contener a la variable v.