EVALUACION EN ORDEN NORMAL

Hemos definido la reducción, 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 de reducción:

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.

Evaluación en orden normal: El valor de una expresión cerrada e es la primer forma canónica que se encuentre en la reducción en orden normal a partir de e.

Ejemplo:

```
(λx.x (λy.x y y) x) (λz.λw.z) -> (λz.λw.z) (λy.(λz.λw.z) y y) (λz.λw.z)
                              -> (λw.λy.(λz.λw.z) y y) (λz.λw.z)
                              -> λy.(λz.λw.z) y y
                              -> λy.(λw.y) y
                              -> λy.y
```

es la reducción en orden normal de (λx.x (λy.x y y) x) (λz.λw.z). La primer forma canónica es λy.(λz.λw.z) y y. Diremos que (λx.x (λy.x y y) x) (λz.λw.z) evalúa (en orden normal) a λy.(λz.λw.z) y y. Lo escribimos (λx.x (λy.x y y) x) (λz.λw.z) =N=> λy.(λz.λw.z) y y.

No parece razonable, ¿verdad? ¿Por qué conformarse con ese término que no es una forma normal? En realidad, podemos aceptarlo razonando 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. De hecho, si revisamos el ejemplo podemos ver que si no nos conformáramos con λy.(λz.λw.z) y y, el resto de la reducción deja λy intacto y continúa reduciendo una expresión que tiene la variable y libre.

Para una expresión cerrada e, decimos que e evalúa a z (e =N=> z) cuando z es la primer forma canónica en la reducción en orden normal a partir de e. Si no hay una tal forma canónica, decimos que e diverge (e↑).

Es fácil de observar que si e -> e', FV(e') ⊆ FV(e). Una contracción no puede agregar variables libres. Eso garantiza que una reducción que comienza con una expresión cerrada, contiene sólo expresiones cerradas.

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.

Corolario: Una forma normal cerrada es una forma canónica.
Dem: Si es cerrada no puede ser una variable. Por la Prop anterior, tampoco puede ser una aplicación. Entonces debe ser una abstracción (o sea, forma canónica).

Este corolario es interesante porque implica que si comienzo a realizar una reducción en orden normal a partir de una expresión cerrada, si la misma llega a una forma normal, entonces necesariamente tiene que contener una forma canónica (ya que la forma normal sería cerrada y por lo tanto canónica).

Conclusión: sea e expresión cerrada, según sea la reducción en orden normal a partir de e, tenemos 3 posibilidades:

1) si la reducción en orden normal a partir de e es finita, entonces alcanza una forma normal e'. Esta forma normal es cerrada y por lo tanto es forma canónica. Por lo tanto existe una primer forma canónica z (que puede o no ser e') en la reducción en orden normal a partir de e. Esa forma canónica satisface e =N=> z.

El ejemplo que dimos arriba trata este caso, (λx.x (λy.x y y) x) (λz.λw.z) =N=> λy.(λz.λw.z) y y.

2) si la reducción en orden normal a partir de e es infinita (no contiene ninguna forma normal), y aún así contiene una forma canónica. Entonces, existe una primer forma canónica z en la reducción en orden normal a partir de e. Esa forma canónica satisface e =N=> z.

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

Como λy.Δ Δ es una forma canónica, (λx.λy.xx) Δ =N=> λy.Δ Δ.

3) si la reducción en orden normal a partir de e es infinita (no contiene ninguna forma normal) y tampoco contiene formas canónicas. En este caso e diverge (e↑).

Δ Δ -> Δ Δ -> Δ Δ -> ...

Acá no hay forma canónica alguna, (Δ Δ)↑.

La evaluación puede definirse también utilizando semántica natural o big-step. En este tipo de semántica, uno no describe un paso de ejecución, sino directamente una relación entre los términos y sus valores (que también son términos, son formas canónicas). Llamaremos =IN=> a esta relación para enfatizar que es la definición inductiva (I) de la evaluación normal (N).

Una regla para formas canónicas:

```
---------------
λv.e =IN=> λv.e
```

Una regla para la aplicación:

```
e =IN=> λv.e"    (e"/v->e') =IN=> z
-----------------------------------
           e e' =IN=> z
```

Cuando afirmamos que e =IN=> z se cumple, estamos diciendo que existe un árbol de derivación que demuestra e =IN=> z. Estos árboles usualmente son difíciles de graficar, por el tamaño de las expresiones intervinientes. Se suele usar indentación para expresar la estructura. También simplificamos escribiendo directamente => para la relación. Por ejemplo, para el ejemplo visto en el caso 2, tendríamos:

```
(λx.λy.xx) Δ            (es una aplicación, por lo tanto usamos la 2da regla, a continuación siguen los 2 hijos)
  λx.λy.xx => λx.λy.xx  (primer hijo, es una abstracción, por lo tanto usamos la 1er regla, no hay hijos)
  λy.Δ Δ => λy.Δ Δ      (segundo hijo, también abstracción, no hay hijos)
=> λy.Δ Δ               (terminó la prueba)
```

Otro ejemplo es el visto en el caso 1, arriba.

```
(λx.x (λy.x y y) x) (λz.λw.z)                    (aplicación, siguen 2 hijos: a y b)
  λx.x (λy.x y y) x => λx.x (λy.x y y) x         (a: abstracción, no hay hijos)
  (λz.λw.z) (λy.(λz.λw.z) y y) (λz.λw.z)         (b: aplicación, siguen 2 hijos: ba y bb)
    (λz.λw.z) (λy.(λz.λw.z) y y)                 (ba: aplicación, siguen 2 hijos: baa y bab)
      λz.λw.z => λz.λw.z                         (baa: abstracción, no hay hijos)
      λw.λy.(λz.λw.z) y y => λw.λy.(λz.λw.z) y y (bab: abstracción, no hay hijos)
    => λw.λy.(λz.λw.z) y y                       (terminó ba)
    λy.(λz.λw.z) y y => λy.(λz.λw.z) y y         (bb: abstracción, no hay hijos)
  => λy.(λz.λw.z) y y                            (terminó b)
=> λy.(λz.λw.z) y y                              (terminó la prueba)
```

La notación con indentación sola es difícil de leer, se mejora si se escriben corchetes "[" a la

izquierda encerrando los subárboles. En el caso de arriba habría un corchete que abarque las 10 líneas, otro que abarque sólo la línea 2, otro de la 3 a la 9, otro de la 4 a la 7, otro para la 8 sola, otro para la 5 sola y otro para la 6 sola. Los corchetes más grandes van más a la izquierda. Es difícil de hacer en un archivo de texto como éste.

En cierta forma las dos definiciones dan dos algoritmos. El primer algoritmo es "realizar sucesivas contracciones en la reducción de orden normal hasta producir la primer forma canónica". El algoritmo que se infiere de la definición inductiva es el siguiente:

```
evalN :: Exp -> Exp
evalN (Lam v e) = (Lam v e)
evalN (App e e') = case evalN e of
                     Lam v e" -> evalN (e"/v->e')
```

que, por supuesto, requiere definir la sustitución para que sea un programa ejecutable. Los árboles mostrados más arriba pueden interpretarse como que la indentación corresponde a las llamadas recursivas.