SEMÁNTICA DENOTACIONAL DEL CÁLCULO LAMBDA

Fue difícil. Por eso, es históricamente posterior a la operacional.
Recordemos que cualquier expresión puede aplicarse a cualquier expresión. Por eso, si interpreto los términos en un conjunto C, y quiero que la aplicación del cálculo lambda se interprete como la aplicación usual en el metalenguaje, necesitaré que C sea igual o isomorfo a C -> C. En efecto, en M M la primer M actúa como función de C -> C y la segunda como elemento de C. Pero si tenemos un conjunto C que satisface eso, y sea $f \in C \to C$ cualquier función del dominio en sí mismo, podemos definir $p \in C \to C$ de la siguiente forma:

$$p\ x = f\ (x\ x)$$

Es más, ahora puedo aplicar p a sí misma: $p\ p = f\ (p\ p)$. Como se ve, partimos de una f cualquiera y le encontramos un punto fijo (p p). O sea que el conjunto C debe ser isomorfo a C -> C y debe satisfacer que toda función $f \in C \to C$ tenga punto fijo.

Es difícil.
Hasta que Scott formuló los dominios, las funciones continuas y las soluciones a ecuaciones recursivas de dominios que utilicen ->, x, + y lifting.

Entonces se definió

$$\mathrm{Dinf} \approx \mathrm{Dinf} \to \mathrm{Dinf}$$

es decir, Dinf isomorfo a Dinf -> Dinf, donde esta flecha se refiere al espacio de funciones continuas (que alguna vez hemos escrito Dinf -c-> Dinf), sabemos que todas ellas tienen punto fijo.

No vamos a estudiar Dinf, pero vamos a utilizar que existe ese isomorfismo para definir la semántica denotacional.

Sean

$$\phi \in \mathrm{Dinf} \to (\mathrm{Dinf} \to \mathrm{Dinf})$$
$$\psi \in (\mathrm{Dinf} \to \mathrm{Dinf}) \to \mathrm{Dinf}$$

los isomorfismos tales que

$$\phi\ .\ \psi = \mathrm{id}$$
$$\psi\ .\ \phi = \mathrm{id}$$

Para las variables libres utilizaremos algo parecido a los estados, salvo que como acá no hay asignación, se lo denomina ambiente y usaremos la letra griega η ($\eta \in \langle var\rangle \to \mathrm{Dinf}$). El conjunto de ambientes se denomina Env.

$$[[\_]] \in \langle exp\rangle \to \mathrm{Env} \to \mathrm{Dinf}$$

Dado que no conocemos Dinf, sólo lo manipulamos a través de los isomorfismos ϕ y ψ:

$$[[v]]\eta = \eta\ v$$

En efecto, el valor de una variable está dado por el valor que tiene asociada en el ambiente.
Para la aplicación, escribiríamos

$$[[e0\ e1]]\eta = ([[e0]]\eta)\ ([[e1]]\eta)$$

respetando que la semántica de la aplicación sea la aplicación del metalenguaje. Pero acá no dan los tipos ya que $([[e0]]\eta) \in \mathrm{Dinf}$, no es una función, tenemos que usar el isomorfismo para convertirlo en una función: $\phi([[e0]]\eta) \in \mathrm{Dinf} \to \mathrm{Dinf}$.

La corregimos y queda así:

$$[[e0\ e1]]\eta = \phi([[e0]]\eta)\ ([[e1]]\eta)$$

De la misma forma, para la abstracción escribiríamos

$$[[\lambda x.e]]\eta = \lambda d{\in}\mathrm{Dinf}.\ [[e]][\eta|v{:}d]$$

respetando que la semántica de la abstracción sea la abstracción del metalenguaje. Nuevamente no dan los tipos ya que debería ser un Dinf, y es una función de Dinf -> Dinf. El otro isomorfismo nos rescata:

$$[[\lambda x.e]]\eta = \psi(\lambda d{\in}\mathrm{Dinf}.\ [[e]][\eta|v{:}d])$$

Por ejemplo, calculemos

$$\begin{aligned}[[\lambda x.x]]\eta &= \psi(\lambda d{\in}\mathrm{Dinf}.\ [[x]][\eta|x{:}d]) && \text{(ecuación de la abstracción)}\\ &= \psi(\lambda d{\in}\mathrm{Dinf}.\ d) && \text{(ecuación de la variable)}\end{aligned}$$

que es la identidad de Dinf, convertida por ψ en un elemento de Dinf.

Si la aplicáramos a cualquier otro término:

[[(λx.x) M]]η = ϕ([[λx.x]]η) ([[M]]η)   (ecuación de la aplicación)
              = ϕ(ψ(λd∈Dinf. d)) ([[M]]η)   (cálculo que ya hicimos para λx.x)
              = (λd∈Dinf. d) ([[M]]η)   (los isomorfismos son inversas mutuas)
              = [[M]]η

Dado que Dinf sólo tiene funciones continuas, habría que demostrar que [[_]] ∈ <exp> -> Env -> Dinf comprobando que el resultado [[e]]η son siempre continuas. El libro lo

TEOREMAS

Teorema de Coincidencia: Si η w = η' w para todo w ∈ FV(e), entonces [[e]]η = [[e]]η'.

Teorema de Sustitución: Si [[d w]]η = η' w para todo w ∈ FV(e), entonces [[e/d]]η = [[e]]η'.

Teorema de Sustitución Finita: [[e/v1->e1, .., vn->en]]η = [[e]][η|v1:[[e1]]η|..|vn:[[en]]η].

Teorema de Renombre: Si v' ∉ FV(e)-{v}, entonces [[λ v'.(e/v->v')]] = [[λ v.e]].

Todos estos teoremas se demuestran muy sencillamente por inducción en la estructura de la expresión. Se utilizan los isomorfismos, pero no se utiliza en absoluto que sean isomorfismos, es decir, no se utiliza que φ . ψ = id y ψ . φ = id en la demostración de estos teoremas.

En cambio, sí se las utiliza para los teoremas que siguen. Para demostrar la correctitud de la regla β , se usa φ . ψ = id. Mientras que para demostrar la correctitud de la regla η , se usa ψ . φ = id.

Prop (correctitud de la regla β): [[(λ v.e) e']] = [[e/v->e']].
Dem: dada en clase y en el libro.

Prop (correctitud de la regla η): Si v ∉ FV(e), entonces [[λ v.e v]] = [[e]].
Dem: dada en clase y en el libro.

Sin conocer la construcción de Dinf, no podemos demostrar que Δ Δ (donde Δ = (λ x.xx)) tiene como semántica ⊥. Asumiremos que [[Δ Δ]]η = ⊥.

¿A qué evaluación corresponde esta semántica? ¿A la normal o a la eager?

A ninguna. Para verlo, calculemos [[λ y.Δ Δ]]η :

[[λ y.Δ Δ]]η = ψ (λ d∈Dinf.[[Δ Δ]][η |y:d])
             = ψ (λ d∈Dinf.⊥)

La función λ d∈Dinf.⊥, es la que para cada elemento de Dinf devuelve ⊥. Recordemos que por definición de dominio de funciones, esta función es el ⊥' de Dinf -> Dinf. O sea que nos queda

[[λ y.Δ Δ]]η = ψ (⊥')

Como ψ es un isomorfismo, tiene que mapear bottom en bottom, queda

[[λ y.Δ Δ]]η = ⊥

Conclusión, la semántica denotacional de λ y.Δ Δ es bottom. Eso demuestra que no coincide ni con la evaluación normal ni con la eager ya que ambas consideran a λ y.Δ Δ como un valor, su evaluación no diverge.

SEMÁNTICA DENOTACIONAL NORMAL

Hay que distinguir entre ⊥ y λ d∈D.⊥. El primero corresponde a una expresión sin forma canónica, mientras que el segundo corresponderá a una (expresión que tenga) forma canónica como λ y.Δ Δ.

Definimos V para interpretar a las expresiones que tienen forma canónica. Intuitivamente, V = valores, de la misma forma que a las formas canónicas las llamamos también valores en su momento. En cambio D, es el conjunto de resultados, que incluyen valores y otras cosas, en este caso, sólo se agrega bottom:

Definimos

D = V_⊥ donde V ≈ D -> D

con

φ ∈ V -> (D -> D)
ψ ∈ (D -> D) -> V

Ahora, λ d∈D.⊥ es el bottom de D -> D, y por lo tanto ψ (λ d∈D.⊥) es el bottom de V, pero no es el bottom de D.

Observar que:

φ _⊥⊥ ∈ D -> (D -> D)
Lbottom . ψ ∈ (D -> D) -> D

Gracias a estas funciones podemos reescribir las tres ecuaciones que dimos al comienzo, reemplazando φ por φ _⊥⊥ y ψ por Lbottom . ψ :

Env = <var> -> D
[[_]] ∈ <exp> -> Env -> D

[[v]]η = η v
[[e0 e1]]η = φ _⊥⊥([[e0]]η) ([[e1]]η)
[[λ x.e]]η = (Lbottom . ψ)(λ d∈D. [[e]][η |v:d])

Los teoremas que vimos antes siguen valiendo porque no dependen de las identidades. También vale la regla β , ya que la igualdad

φ _⊥⊥ . (Lbottom . ψ) = id

sigue valiendo. Pero no vale la regla η , ya que la semántica de λ y.Δ Δ y es ψ (λ d∈D.⊥) mientras que la de Δ Δ es ⊥.

Es interesante observar que si bien la semántica denotacional no expresa un orden de evaluación ya que no es operacional, establece indirectamente el orden natural en que debe evaluarse. Por ejemplo, en el caso de la aplicación, si [[e0]]η = ⊥, entonces φ _⊥⊥([[e0]]η) = ⊥_{D->D} por definición de f_⊥⊥. Ahora ⊥_{D->D} es la función que siempre devuelve ⊥, entonces φ _⊥⊥([[e0]]η) ([[e1]]η) = ⊥, o sea, [[e0 e1]]η = ⊥. Acabamos de comprobar que si [[e0]]η = ⊥ entonces [[e0 e1]]η = ⊥. Eso indica claramente que e0 debe evaluarse para evaluarse e0 e1. ¿Por qué otra razón la no terminación de e0 podría implicar siempre la no terminación de e0 e1?

En cambio, por más que [[e1]]η = ⊥, no necesariamente [[e0 e1]]η = ⊥. Eso indica que e1 no necesariamente debe evaluarse para evaluarse e0 e1.

Para ver un ejemplo en que [[e1]]η = ⊥ y [[e0 e1]]η ≠ ⊥ lo da e0 = λ y.λ x.x, e1 = Δ Δ. Como mencionamos más arriba, la regla β vale, por lo tanto [[e0 e1]]η = [[λ x.x]]η ≠ ⊥.

SEMÁNTICA DENOTACIONAL EAGER

La semántica denotacional eager, no debería satisfacer la regla β , ya que la evaluación eager no lo hace. En efecto,
si la evaluación eager satisfaciera dicha regla, la evaluar (λ y.λ x.x) (Δ Δ) debería dar lo mismo que evaluar λ x.x, pero ese no es el caso ya que este último término ya está en forma canónica, mientras que (λ y.λ x.x) (Δ Δ) diverge al evaluarse eager. En la evaluación eager, nunca se alcanza a utilizar el operador λ y.λ x.x ya que el operando Δ Δ, que debe evaluarse antes de sustituir y en λx.x, diverge.

En general, un operador recién será utilizado cuando ya se haya evaluado el operando. El operador, entonces, no debe interpretarse como una función de D -> D, ya que sólo se aplica a formas canónicas. Debe interpretarse como una función de V -> D, ya que V es el dominio donde se interpretan las (expresiones que tienen) formas canónicas.

Definimos

D = V_⊥ donde V ≈ V -> D

con

ϕ ∈ V -> (V -> D)
ψ ∈ (V -> D) -> V

Observar que:

ϕ_⊥⊥ ∈ D -> (V -> D)
Lbottom . ψ ∈ (V -> D) -> D

Gracias a estas funciones podemos reescribir las tres ecuaciones una vez más:

Env = <var> -> V
[[_]] ∈ <exp> -> Env -> D

[[v]]η = Lbottom (η v)
[[e0 e1]]η = (ϕ_⊥⊥([[e0]]η))_⊥⊥ ([[e1]]η)
[[λx.e]]η = (Lbottom . ψ)(λz∈V. [[e]][η|v:z])

Los teoremas que vimos antes (salvo el de sustitución) siguen valiendo porque no dependen de las identidades. Pero no valen las reglas β ni η . El teorema de sustitución puede adecuarse a ambientes que asocian valores de V a las variables, pero el significado del teorema resultante es más limitado.

Nuevamente la semántica denotacional ahora puede verse que tanto si [[e0]]η = ⊥ o si [[e1]]η = ⊥ tendremos que [[e0 e1]]η = ⊥. Eso indica que tanto e0 como e1 deben evaluarse para evaluar e0 e1.