

SEMÁNTICA DE CONTINUACIONES O INVERSA

Como vimos, sucesivas extensiones al lenguaje forzaron redefiniciones de f_* , f_* y f_+ . Esto se debe a que (en el caso de f_*) la semántica de la secuencia está definida así:

$$[[c_0;c_1]]s = [[c_1]]_*([[c_0]]s)$$

que es en cierta forma artificial, ya que $[[c_1]]$ parece estar aplicada al resultado de $[[c_0]]$ y por lo tanto hay que forzar la propagación de lo que $[[c_0]]$ produce, con la única excepción del estado final de $[[c_0]]$ que es en realidad lo único que $[[c_1]]$ debería recibir.

Para evitar esto se propone otra forma de definir la semántica: denotemos a esta nueva semántica $\{\{c\}\}$ en vez de $[[c]]$ que va a seguir denotando la semántica que hemos estado viendo (se llama semántica directa). La idea de $\{\{c\}\}$ es que en vez de devolver el estado final, dice "decime qué querés hacer con el estado final y yo lo hago por vos". En el caso del lenguaje imperativo simple (sin ninguna de las extensiones) teníamos

$$[[_]] \in \langle \text{comm} \rangle \rightarrow \Sigma \rightarrow \Sigma_{\perp}$$

La frase "decime qué querés hacer con el estado final y yo lo hago por vos" se podría escribir así:

$$(*) \ \{\{c\}\}ks = k_{\perp\perp} ([[c]]s)$$

donde k es "lo que se quiere hacer con el estado final". La ecuación muestra que $\{\{c\}\}$ recibe "lo que se quiere hacer con el estado final", y también recibe el estado inicial s . No devuelve el estado final $[[c]]$ sino que le aplica k a dicho resultado.

En la ecuación (*) se ve que $\{\{_ \}\}$ recibe 3 argumento en vez de 2 que recibe $[[_]]$. El argumento nuevo, k , es "lo que se quiere hacer con el estado final", se suele llamar "continuación". Como se ve en (*), k debe poder aplicarse a un estado (al estado final):

$$\{\{_ \}\} \in \langle \text{comm} \rangle \rightarrow (\Sigma \rightarrow X_{\perp}) \rightarrow \Sigma \rightarrow X_{\perp}$$

donde A puede ser cualquier cosa, se agrega \perp ya que en (*) se ve que el resultado final puede ser \perp ($k_{\perp\perp}$ propaga \perp y por lo tanto puede devolver \perp). Como se ve, las continuaciones son funciones $\in \Sigma \rightarrow X_{\perp}$.

No debe interpretarse (*) como una definición de $\{\{_ \}\}$, sólo es una propiedad que se desea que tenga. La intención es justamente definir $\{\{_ \}\}$ desde cero, a través de ecuaciones, como hemos definido $[[_]]$. El propósito es resolver las molestias causadas por ecuaciones como

$$[[c_0;c_1]]s = [[c_1]]_*([[c_0]]s)$$

que fuerzan a propagar siempre el comportamiento de $[[c_0]]$, salvo su estado final.

Definamos entonces $\{\{_ \}\}$ teniendo en cuenta que recibe una continuación k , y que no devuelve el estado final sino que aplica k a dicho estado:

$$\begin{aligned} \{\{\text{skip}\}\}ks &= k s \\ \{\{v:=e\}\}ks &= k [s \mid v : [[e]]s] \\ \{\{\text{if } b \text{ then } c_0 \text{ else } c_1\}\}ks &= \begin{cases} \{\{c_0\}\}ks & \text{si } [[b]]s \\ \{\{c_1\}\}ks & \text{c.c.} \end{cases} \\ \{\{c_0;c_1\}\}ks &= \{\{c_0\}\}(\{\{c_1\}\}k)s \end{aligned}$$

Seguramente ésta es la ecuación más difícil de entender: $\{\{c_0\}\}$ tiene 2 argumentos, el primero es $\{\{c_1\}\}k$ y el segundo es el estado inicial s . Primero que nada habría que confirmar que esto está bien escrito, que $\{\{c_1\}\}k$ es una continuación (si no lo fuera no podría ser el primer argumento de $\{\{c_0\}\}$). Efectivamente, por el tipo de $\{\{_ \}\}$, $\{\{c_1\}\}$ recibe una continuación y devuelve una continuación, o sea que, aplicado a k es una continuación. Tenemos entonces que $\{\{c_1\}\}k$ es una continuación.

Lo que la ecuación dice, entonces, es dado que k es "lo que se quiere hacer con el estado final de $c_0;c_1$ ", se evalúa c_0 y se le pasa como argumento "lo que se quiere hacer con el estado final de c_0 ". ¿Y qué se quiere hacer con el estado final de c_0 ? Se lo quiere usar para evaluar c_1 . Eso explica que la continuación de $\{\{c_0\}\}$ sea de la forma $\{\{c_1\}\}...$ Para explicar que el argumento de $\{\{c_1\}\}$ sea k , hay que

preguntarse "qué va a querer hacerse con el estado final de $c1$ ". ¿Y qué va a querer hacerse con él? Justamente ése va a ser el estado final de $c0;c1$ y habíamos dicho que lo que quería hacerse con él estaba dado por k .

Para terminar de convencerse uno podría intentar demostrar la propiedad (*) por inducción en la estructura de c :

$c = \text{skip}$
 $\{\{\text{skip}\}\}ks = k \ s = k_{\perp\perp} \ s = k_{\perp\perp} \ (\{\{\text{skip}\}\}s)$

$c = v := e$
 $\{\{v := e\}\}ks = k \ [s \mid v : \{\{e\}\}s] = k_{\perp\perp} \ [s \mid v : \{\{e\}\}s] = k_{\perp\perp} \ (\{\{v := e\}\}s)$

$c = \text{if } b \text{ then } c0 \text{ else } c1$, caso $\{\{b\}\}s$ verdadero
 $\{\{\text{if } b \text{ then } c0 \text{ else } c1\}\}ks = \{\{c0\}\}ks$
 $= k_{\perp\perp} \ (\{\{c0\}\}s)$ por hipótesis inductiva
 $= k_{\perp\perp} \ (\{\{\text{if } b \text{ then } c0 \text{ else } c1\}\}s)$

caso $\{\{b\}\}s$ falso es similar

$c = c0;c1$
 primero, observar que la propiedad (*) también se puede escribir $\{\{c\}\}k = k_{\perp\perp} \ . \ \{\{c\}\}$ donde $.$ es composición de funciones.

$\{\{c0;c1\}\}ks = \{\{c0\}\}(\{\{c1\}\}ks)$
 $= (\{\{c1\}\}k)_{\perp\perp} \ (\{\{c0\}\}s)$ por hipótesis inductiva en $c0$
 $= (k_{\perp\perp} \ . \ \{\{c1\}\})_{\perp\perp} \ (\{\{c0\}\}s)$ por hipótesis inductiva en $c1$
 $= (k_{\perp\perp} \ . \ \{\{c1\}\}_{\perp\perp}) \ (\{\{c0\}\}s)$ por $(f_{\perp\perp} \ . \ g)_{\perp\perp} = f_{\perp\perp} \ . \ g_{\perp\perp}$
 $= k_{\perp\perp} \ (\{\{c1\}\}_{\perp\perp}(\{\{c0\}\}s))$
 $= k_{\perp\perp} \ (\{\{c0;c1\}\}s)$

Esto termina de demostrar la propiedad (*) para skip, asignación, condicional y secuencia.

La propiedad $(f_{\perp\perp} \ . \ g)_{\perp\perp} = f_{\perp\perp} \ . \ g_{\perp\perp}$ que utilizamos puede comprobarse así:

Sean

$f \in Q \rightarrow D$
 $g \in P \rightarrow Q_{\perp}$

entonces

$f_{\perp\perp} \in Q_{\perp} \rightarrow D$
 $f_{\perp\perp} \ . \ g \in P \rightarrow D$
 $(f_{\perp\perp} \ . \ g)_{\perp\perp} \in P_{\perp} \rightarrow D$
 $g_{\perp\perp} \in P_{\perp} \rightarrow Q_{\perp}$
 $f_{\perp\perp} \ . \ g_{\perp\perp} \in P_{\perp} \rightarrow D$

Para comparar $(f_{\perp\perp} \ . \ g)_{\perp\perp}$ y $f_{\perp\perp} \ . \ g_{\perp\perp}$ lo hacemos punto a punto. Sea $x \in P_{\perp}$. Si $x = \perp$, entonces

$(f_{\perp\perp} \ . \ g)_{\perp\perp} x = (f_{\perp\perp} \ . \ g)_{\perp\perp} \perp$
 $= \perp$
 $= f_{\perp\perp} \ \perp$
 $= f_{\perp\perp} (g_{\perp\perp} \ \perp)$
 $= (f_{\perp\perp} \ . \ g_{\perp\perp}) \ \perp$
 $= (f_{\perp\perp} \ . \ g_{\perp\perp}) x$

Si $x \neq \perp$

$(f_{\perp\perp} \ . \ g)_{\perp\perp} x = (f_{\perp\perp} \ . \ g) x$
 $= f_{\perp\perp} (g x)$
 $= f_{\perp\perp} (g_{\perp\perp} x)$
 $= (f_{\perp\perp} \ . \ g_{\perp\perp}) x$

Continuamos con la definición de $\{\{c\}\}$ para los casos restantes: while y newvar.

$\{\{\text{while } b \text{ do } c\}\}ks = \{\{\text{if } b \text{ then } c; \text{while } b \text{ do } c \text{ else skip}\}\}ks$
 $\quad \quad \quad \{\{c; \text{while } b \text{ do } c\}\}ks \quad \quad \quad \text{si } \{\{b\}\}s$

$$\begin{aligned}
 &= \begin{cases} \{\{\text{skip}\}\}ks & \text{c.c.} \\ \{\{c\}\}(\{\{\text{while } b \text{ do } c\}\}k)s & \text{si } [[b]]s \end{cases} \\
 &= \begin{cases} k s & \text{c.c.} \end{cases}
 \end{aligned}$$

por lo tanto

$$\{\{\text{while } b \text{ do } c\}\}k = w$$

donde

$$w s = \begin{cases} \{\{c\}\}ws & \text{si } [[b]]s \\ k s & \text{c.c.} \end{cases}$$

o, como reformulamos en similares situaciones,

$$\{\{\text{while } b \text{ do } c\}\}k = Y_{\{\Sigma \rightarrow X_{\perp}\}} F$$

donde

$$F w s = \begin{cases} \{\{c\}\}ws & \text{si } [[b]]s \\ k s & \text{c.c.} \end{cases}$$

Finalmente,

$$\{\{\text{newvar } v := e \text{ in } c\}\}ks = \{\{c\}\}(\lambda s' \in \Sigma. k [s' | v:sv])[s | v:[[e]]s]$$

Se asume que k es lo que se quiere hacer con el estado final del `newvar`. El estado en que se evalúa c es $[s | v:[[e]]s]$, es decir, uno en el que v ya se inicializó. El argumento $(\lambda s' \in \Sigma. k [s' | v:sv])$ dice "qué se quiere hacer con el estado final de c ". Lo que se quiere hacer es dárselo a k , salvo que k debe recibir el estado final de todo el `newvar`, es decir, sin el valor local de v . Por ello se restaura el valor global antes de dárselo a k .

NOTACION

Al dar la semántica de continuaciones, es habitual suprimir mencionar el estado en las ecuaciones en que no es necesario. En este caso solo se puede suprimir en `skip` y `secuencia`:

$$\begin{aligned}
 \{\{\text{skip}\}\}k &= k \\
 \{\{c_0; c_1\}\}k &= \{\{c_0\}\}(\{\{c_1\}\}k)
 \end{aligned}$$

FALLAS

Para dar la semántica de continuaciones para un lenguaje que tiene fallas, hacen falta 2 continuaciones: una es la que dice "lo que se quiere hacer con el estado final de c si c termina bien" y otra que dice "lo que se quiere hacer con el estado final de c si c termina mal".

$$\{\{_ \}\} \in \langle \text{comm} \rangle \rightarrow (\Sigma \rightarrow X_{\perp}) \rightarrow (\Sigma \rightarrow X_{\perp}) \rightarrow \Sigma \rightarrow X_{\perp}$$

$$\begin{aligned}
 \{\{\text{skip}\}\}kjs &= k s \\
 \{\{\text{fail}\}\}kjs &= j s \\
 \{\{v:=e\}\}kjs &= k [s | v : [[e]]s] \\
 \{\{\text{if } b \text{ then } c_0 \text{ else } c_1\}\}kjs &= \begin{cases} \{\{c_0\}\}kjs & \text{si } [[b]]s \\ \{\{c_1\}\}kjs & \text{c.c.} \end{cases} \\
 \{\{c_0; c_1\}\}kjs &= \{\{c_0\}\}(\{\{c_1\}\}kj)js
 \end{aligned}$$

Esta ecuación es interesante, dice que la continuación en caso de que termine mal `c0;c1`, o `c0` o `c1` es la misma. Es lógico ya que si `c0` termina mal, `c0;c1` termina mal; y si `c1` termina mal, también.

$$\{\{\text{catchin } c_0 \text{ with } c_1\}\}kjs = \{\{c_0\}\}k(\{\{c_1\}\}kj)s$$

En el caso del catchin los roles de k y j se invierten. La dualidad entre skip y ; por un lado y fail y catchin por el otro, es notoria.

$$\{\{while\ b\ do\ c\}\}kj = Y_{\{\Sigma \rightarrow X_{\perp}\}} F$$

donde

$$F\ w\ s = \begin{cases} \{\{c\}\}wjs & \text{si } [[b]]s \\ k\ s & \text{c.c.} \end{cases}$$

En el caso del newvar, el valor global de la variable debe restaurarse en ambos casos: termine bien o mal:

$$\{\{newvar\ v := e\ in\ c\}\}kjs = \{\{c\}\}(\lambda s' \in \Sigma. k\ [s' | v:sv])(\lambda s' \in \Sigma. j\ [s' | v:sv])[s | v: [[e]]s]$$