Procedimientos con parámetros (Ejercicio 3.15)

Necesitamos "generar memoria" para los parámetros, que "son fijos" si no se soporta la recursión, pero se han de "recrear dinámicamente" si se desea un comportamiento recursivo "razonable". Ello hace que podamos realizar dicha generación de manera "estática" en el primer caso, pero debamos "retrasarla" (y reiterarla) hasta cada llamada si se desea ir "generando" la correspondiente "pila" para manejar cada parámetro. Curiosamente el mecanismo de bloque que hemos incorporado permitiría "simular" ambos comportamientos (según luego usemos $[call_{ns}]$ o $[call^{rec}_{nec}]$ "moviendo" las variables-parámetro bien "fuera" del procedimiento, al bloque que lo contiene, o "dentro" del mismo, incorporando un nuevo bloque conteniendo las declaraciones de los parámetros (inicializados "arbitrariamente") y el "cuerpo" del procedimiento. Naturalmente, en ambos casos el "paso de parámetros" será realizado al llamar al procedimiento, generando entonces el efecto de las correspondientes "asignaciones" a las variables-parámetro.

Veamos como quedarían las dos reglas $[call_{ns}]$ y $[call^{rec}_{ns}]$ que incorporan directamente los efectos arriba "sugeridos":

$$[call^{p}_{ns}] \quad \frac{env'_V, env'_p \vdash \langle S, sto'' \rangle \rightarrow sto'}{env_V, env_p \vdash \langle call\ p(a_1,a_2), sto \rangle \rightarrow sto'}$$

where $env_p\ p = (x_1, x_2, S, env'_V, env'_p)$

$v_1 = \mathcal{A}[\![a_1]\!](sto \circ env_V) \qquad v_2 = \mathcal{A}[\![a_2]\!](sto \circ env_V)$

$sto''_1 = sto_1[env'_V(x_1) \rightarrow v_1][env'_V(x_2) \rightarrow v_2]$

cambiando upd_p en la siguiente forma

upd_p (proc $p(x_1,x_2)$ is S ; D_p, env_V, env_p, sto)

$upd_p(D_p, env'_V, env_p[p \rightarrow (x_1, x_2, S, env'_V, env_p)], sto')$

donde $env'_V = env_V[x_1 \rightarrow sto_2(next)][x_2 \rightarrow new(next)]$, $sto'_1 = sto_1$, $sto'_2 = new(new(next))$

donde hemos tenido que añadir sto como nuevo argumento de upd_p para que pueda lograr el efecto deseado de "crear memoria" para las variables-parámetro. No nos molestamos en "inicializarlas", pues ya se hará cuando efectuemos cada llamada. Obsérvese que en todas ellas se utilizará la misma posición de memoria.

$$[call^{prec}_{ns}] \quad \frac{env''_V, env'_p[p \to env_p\, p] \vdash (S, sto''_0) \to sto'}{env_V, env_p \vdash \langle call\ p(a_1,a_2), sto\rangle \to sto'}$$

where $env_p\, p = (x_1, x_2, S, env'_V, env'_p)$, $\begin{cases} v_1 = \mathcal{A}[\![a_1]\!](sto \circ env_V) \\ v_2 = \mathcal{A}[\![a_2]\!](sto \circ env_V) \end{cases}$

$env''_V = env'_V[x_1 \to sto_2][x_2 \to new(next)]$, $sto''_1 = sto_1[sto_2 \to v_1][new\ next \to v_2]$,

$sto''_2 = new(new\ next)$

(next)

Ahora ejecutamos S tras "recuperar" el entorno "estático" de p, env'_V, añadiendole la "nueva memoria" para los parámetros. Cada llamada, recursiva o no, genera siempre direcciones nuevas que nunca se reutilizan, aunque "al volver" tras cada llamada recursiva sí que "se vuelve a tener acceso" a los parámetros de la llamada recursiva "anterior" (anidada) que la generó.

En este caso no necesitamos modificar upd_p (incorporando sto), ya que la declaración no crea memoria para los parámetros.

Naturalmente, podríamos utilizar $[call^{rec}_{ns}]$ para todos los procedimientos, recursivos o no, pero en tal caso cada llamada genera siempre ("innecesariamente") nuevas direcciones de memoria para generar los procedimientos.

Observación final: Realmente, si no tenemos procedimientos anidados podríamos usar 2 únicas direcciones "fijas" para manejar las llamadas a todos los procedimientos. ¿Cómo se haría?