SOBRE LA SEMANTICA OPERACIONAL DEL FOR

Primera aproximación: vía una "codificación sencilla" utilizando while

- Si tuviesemos que "simular" cada uno de los elementos que conforman un for, utilizando las construcciones de While, parece probable que optemos por algo "muy parecido" a lo siguiente:

  for $x := a_1$ to a_2 do S $\equiv$

  $x := a_1$; while $x \leq a_2$ do (S ; $x := x+1$)

- Como "no nos dejan" utilizar la instrucción while en la definición de la semántica de paso corto, hemos de "desdoblarlo" utilizando la definición de su propia semantica, lo que nos conduce a

  if $x \leq a_2$ then (S ; $x := x+1$; while $x \leq a_2$ do (S; $x := x+1$)) else skip

- "Plegamos" ahora la combinación $x := x+1$; while $x \leq a_2$ do (S; $x := x+1$)) en el correspondiente for $x := x+1$ to a_2 do S, y nos queda una traducción "asumible" que generaría la semántica del for "a imagen" de lo que hicimos para el while

- Y mirando ahora la definición de la semántica de paso corto sintetizaríamos las siguientes reglas para el for:

[for$^{\text{not-empty}}_{\text{bs}}$]
$$\frac{\langle x := a_1, s\rangle \to s',\ \langle S, s'\rangle \to s'',\ \langle \text{for } x := x+1 \text{ to } a_2 \text{ do } S, s''\rangle \to s'''}{\langle \text{for } x := a_1 \text{ to } a_2 \text{ do } S, s\rangle \longrightarrow s'''}$$

aplicable cuando $\mathcal{B}[\![x \leq a_2]\!]s' = tt$

[for$^{\text{empty}}_{\text{bs}}$]
$$\frac{\langle x := a_1, s\rangle \to s'}{\langle \text{for } x := a_1 \text{ to } a_2 \text{ do } S, s\rangle \longrightarrow s'} \quad \text{if } \mathcal{B}[\![x \leq a_2]\!]s' = ff$$

Alternativa, simplemente dejamos $s' = s$ cuando $\mathcal{B}[\![a_1 \leq a_2]\!] = ff$

Segunda aproximación: ejecutamos "repetidamente" S con $x \in a_1 .. a_2$

- Buscamos una segunda simulación que "blinde" el significado buscado. Al efecto, estudiamos cuándo y porqué la primera simulación podría no funcionar de este modo. Vemos que a_1 sólo se maneja al principio, así que no da problema. En cambio, a_2 sí que lo reevaluamos, por lo que cambiará su valor si en S se modifica el valor de cualquier $v \in VL(a_2)$. Y lo mismo podría sucederle a la propia variable x en el mismo caso. En principio, resolvemos el problema del mismo modo, introducimos sendas variables "nuevas" xa, $x' \notin VL(S) \cup \{x\}$, y extendemos el "preámbulo" $x := a_1$, tornándolo en $x' := a_1$; $x := x'$; $xa := a_2$, mientras que el for interno quedaría for $x := x'+1$ to xa do S.
- Obsérvese que S <u>no</u> se modifica en absoluto, lo cual significa que si en S se modifica x, esto se seguirá haciendo, y la correspondiente "vuelta" del for se seguirá ejecutando con ese valor modificado. Sin embargo, al terminar cada vuelta, la introducción de x' garantiza que la vuelta siguiente se inicia en efecto con el valor siguiente del intervalo "inicial" $a_1 .. a_2$.
- Obsérvese que si modificamos el valor de x durante la ejecución de S se supone que sería porque por "alguna extraña razón" así lo deseamos. Sin embargo, entendemos que la semántica "global" del for es "sagrada", por lo que la susodicha modificación "deja de tener efecto" al terminar cada vuelta.
- En principio, "nos hemos curado en salud" introduciendo xa, x' en todos los casos. Naturalmente, podríamos evitar "añadidos innecesarios" controlando las variables asignadas (y por tanto potencialmente modificadas) en S, Vasign(S), introduciendo

x_a sii $VL(a_2) \cap Varsign(S) \neq \emptyset$, y x' sii $x \in Varsign(S)$. En particular, de este modo evitamos "nuevos añadidos" al definir la semántica de cada for "remanente" en la premisa de la regla. En efecto, si introducimos nuevas x_a y x' en todos los casos estaríamos "multiplicando" la modificación, lo que en efecto funcionaría como se desea, pero introduciendo "molestas" variables "de usar y tirar" que "morirían" al final de cada vuelta, cuando se podrían perfectamente "rentilizar" para todo el for, lo que precisamente se haría con la generación "condicionada" comentada antes.

- Una solución alternativa pasaría por la "extensión" del lenguaje con una construcción "auxiliar" for_aux: el for ordinario generaría las variables nuevas para definir su semántica, pero al "desplegarse" (o incluso directamente a su comienzo) genera "llamadas" a for_aux, que por su parte generaría llamadas "a sí mismo", y <u>no</u> necesitaría generar "más" variables nuevas para la definición de la semántica.
- Finalmente, si no tenemos en cuenta el entorno global donde se enmarca cada for, las variables nuevas "para S" podrían no serlo para dicho entorno, en cuyo caso las "machacaría" inevitablemente. La solución más elegante nos la facilitarían los bloques simples que hemos estudiado, que como vimos lidian precisamente con esta situación: crean siempre variables 100% nuevas, a pesar de que se llamen siempre como otras variables "viejas" (recordar que en todo estado existen <u>todas</u>), a las que sin embargo no perturban <u>nunca</u>). Justo, justo, lo que necesitamos aquí.