

Ejercicios recomendados TProg

Tema 1: El lenguaje While: Semántica de sus Expresiones

Ejercicio

Siendo la sintaxis para n

$$n ::= 0 \mid 1 \mid 0\ n \mid 1\ n$$

¿Se puede definir N correctamente?

$$N[0] = 0 ; N[1] = 1 ; N[0\ n] = N[n] ; N[1\ n] = ???$$

No es posible definir N , ya que los números 11 y 101 cumplen que $N[1] = N[01]$, luego no podemos definir una regla composicional para la semántica de $N[1\ n]$.

Ejercicio

Demostrar que las ecuaciones para \mathcal{A} definen una función total.

Procedemos por inducción estructural.

Casos base:

- $A[n] s = N[n]$ está definida por ser N completo por construcción
- $A[x] s = s x$ está definida ya que s es una función completa por definición

Casos compuestos:

- $A[a_1 \text{ op } a_2] = A[a_1] \text{ op } A[a_2]$. Por inducción, $A[a_i]$ está bien definido y las operaciones binarias involucradas son completas.

Ejercicio

Demostrar que las ecuaciones para \mathcal{B} definen una función total.

Procedemos por inducción estructural.

Casos base:

- $B[\text{true}]s$ y $B[\text{false}]s$ son funciones constantes siempre definidas
- $B[a_1 \text{ comp } a_2]s$ son funciones totales ya que los casos en los que están definidos son exhaustivos.

Casos compuestos:

- $B[\neg b]$
- $B[b_1 \wedge b_2]$

Ejercicio

Se extiende Bexp a Bexp':

$$\begin{aligned} b ::= & \text{ true } | \text{ false } | \\ & a_1 = a_2 \mid a_1 \neq a_2 \mid a_1 \leq a_2 \mid a_1 \geq a_2 \mid a_1 < a_2 \mid \\ & a_1 > a_2 \mid \neg b \mid b_1 \wedge b_2 \mid b_1 \vee b_2 \mid b_1 \Rightarrow b_2 \mid \\ & b_1 \Leftrightarrow b_2 \end{aligned}$$

- Extender la función semántica \mathcal{B} definiéndola sobre todo Bexp'.
- Las expresiones b_1 y b_2 son equivalentes si para todo estado s

$$\mathcal{B}[b_1]s = \mathcal{B}[b_2]s$$

Demostrar que para cada $b' \in \text{Bexp}'$ existe $b \in \text{Bexp}$ tal que b' y b son equivalentes (azúcar sintáctico).

Función semántica extendida:

$$\mathcal{B}[a_1 \geq a_2] = \mathcal{B}[a_2 \leq a_1]$$

$$\mathcal{B}[a_1 < a_2] = \mathcal{B}[\neg(a_2 \leq a_1)]$$

...

Por cómo hemos extendido la semántica, el azúcar sintáctico es inmediato.

Ejercicio

Definir el conjunto de variables libres para una expresión booleana y demostrar un resultado similar al Lema 1.

$$\text{FV}(\text{true}) = \text{FV}(\text{false}) = \emptyset$$

$$\text{FV}(x_b) = \{ x_b \}$$

$$\text{FV}(a_1 = a_2) = \text{FV}(a_1 \leq a_2) = \text{FV}(a_1) \cup \text{FV}(a_2)$$

$$\text{FV}(\neg b) = \text{FV}(b)$$

$$\text{FV}(b_1 \wedge b_2) = \text{FV}(b_1) \cup \text{FV}(b_2)$$

Lema 2: Sean s, s' State tales que $\forall x \text{ FV}(b) s x = s' x$, entonces $\mathcal{B}[b]s = \mathcal{B}[b]s'$.

Demostración: procedemos por inducción estructural.

Casos base:

- $\mathcal{B}[\text{true}]s$ y $\mathcal{B}[\text{false}]s$ son constantes respecto a s
- $\mathcal{B}[x]s = \mathcal{B}[x]s'$ por hipótesis

Casos compuestos:

- $\mathcal{B}[\neg b]s$
- $\mathcal{B}[b_1 \wedge b_2]s$

Procedemos por inducción estructural.

Casos base:

- $A[[n[y \mapsto a_0]]] s = A[[n]] s = N[[n]] = A[[n]](s[y \mapsto A[[a_0]]s])$
- $A[[x[y \mapsto a_0]]] s = A[[x]] s = A[[x]](s[y \mapsto A[[a_0]]s])$
- $A[[y[y \mapsto a_0]]] s = A[[a_0]] s = A[[y]](s[y \mapsto A[[a_0]]s])$

Casos compuestos:

- $A[[a_1 op a_2][y \mapsto a_0]] s = A[[a_1[y \mapsto a_0]] op (a_2[y \mapsto a_0])] = A[[a_1[y \mapsto a_0]]] op A[[a_2[y \mapsto a_0]]] = (h.i.) = \dots$

Ejercicio

Definir la sustitución para expresiones booleanas $b[y \mapsto a_0]$ y demostrar un resultado similar al del ejercicio anterior.

$\text{true}[y \mapsto a_0] = \text{true}$

$\text{false}[y \mapsto a_0] = \text{false}$

$(a_1 op a_2)[y \mapsto a_0] = ((a_1[y \mapsto a_0]) op (a_2[y \mapsto a_0]))$

$(\neg b)[y \mapsto a_0] = \neg(b[y \mapsto a_0])$

$(b_1 \wedge b_2)[y \mapsto a_0] = (b_1[y \mapsto a_0]) \wedge (b_2[y \mapsto a_0])$

Demostrar que $B[[b[y \mapsto a_0]]] s = B[[b]](s[y \mapsto B[[a_0]]s])$ para todo s

...

Tema 2: Semántica operacional

Ejercicios "obligatorios": 2.7, 2.10, 2.11, 2.17, 2.20, 2.21, 2.24 y 2.29

Ejercicio 2.4

Determinar cuáles de las sentencias siguientes terminan/ciclan **siempre** (esto es, sea cual sea el estado inicial), y cuáles lo hacen dependiendo de dicho estado:

- `while $\neg(x = 1)$ do ($y := y \times x$; $x := x - 1$)`

Caso 1: En el estado inicial S, $[x \mapsto 1]$. Entonces la ejecución acaba en un paso por la regla operacional [while ff].

Caso 2: En el estado inicial S, $[x \mapsto n < 1]$. Entonces tras la ejecución del bucle tenemos que $[x \mapsto n - 1 < n < 1]$. Tenemos por tanto como invariante que $[x \mapsto n < 1]$, luego la condición de parada $[x = 1]$ del while nunca se cumple y el árbol de derivación es por tanto infinito.

Caso 3: En el estado inicial S, $[x \mapsto n > 1]$. Entonces tras n ejecuciones se cumple la condición de parada, como puede comprobarse fácilmente por inducción sobre n.

- `while $1 \leq x$ do ($y := y \times x$; $x := x - 1$)`

Caso 1: $1 \leq x = n$. La ejecución acaba en n pasos.

Caso 2: $1 > x$. La ejecución acaba en un paso.

- while true do skip

Supongamos que $\langle \text{while true do skip} , s \rangle \rightarrow s'$.

Caso 1: El último paso de derivación es $[\text{while}_{\text{bs}} \text{ ff}]$.

Entonces tenemos que $B[[\text{true}]]s = \text{ff}$, lo que es imposible.

Caso 2: El último paso de derivación es $[\text{while}_{\text{bs}} \text{ tt}]$.

Entonces tenemos que $B[[\text{true}]]s = \text{ff}$, $\langle \text{skip} , s \rangle \rightarrow s_1$, $\langle \text{while true do skip} , s_1 \rangle \rightarrow s'$.

Por el determinismo de $\text{bs} + [\text{skip}_{\text{bs}}]$, tenemos que $s_1 = s$, luego tenemos como antecesor en el árbol de derivación de esta frase a $\langle \text{while true do skip} , s \rangle \rightarrow s'$.

Pero si todo nodo $\langle \text{while true do skip} , s \rangle \rightarrow s'$ tiene como antecesor en su árbol de derivación a otro nodo $\langle \text{while true do skip} , s \rangle \rightarrow s'$, el árbol de derivación en cuestión contiene infinitos nodos, lo cual es imposible porque los árboles de derivación son finitos.

Concluimos que no existe árbol de derivación para $\langle \text{while true do skip} , s \rangle \rightarrow s'$.

Ejercicio 2.6

- Demostrar que $S_1 ; S_2 ; S_3$ y $(S_1 ; S_2) ; S_3$ son semánticamente equivalentes, para cualesquiera sentencias $S_1, (S_2, S_3)s$.

Sea $s \in \text{State}$. Queremos demostrar que $\langle S_1 ; S_2 ; S_3 , s \rangle \rightarrow s' \Leftrightarrow \langle (S_1 ; S_2) ; S_3 , s \rangle \rightarrow s'$.

\Rightarrow Supongamos que $\langle S_1 ; S_2 ; S_3 , s \rangle \rightarrow s'$. Entonces existe un árbol de derivación que acaba en esta proposición. Su último paso ha de ser $[\text{comp}_{\text{bs}}]$, luego los hijos del nodo principal son $\langle S_1 , s \rangle \rightarrow s_1$ y $\langle S_2 ; S_3 , s_1 \rangle \rightarrow s'$. A su vez, el nodo $\langle S_2 ; S_3 , s_1 \rangle \rightarrow s'$ tiene como hijos a $\langle S_2 , s_1 \rangle \rightarrow s_2$ y a $\langle S_3 , s_2 \rangle \rightarrow s_3$. Luego aplicando de nuevo $[\text{comp}_{\text{bs}}]$ sobre las premisas $\langle S_1 , s \rangle \rightarrow s_1$ y $\langle S_2 , s_1 \rangle \rightarrow s_2$ llegamos a $\langle S_1 ; S_2 , s \rangle \rightarrow s_2$; usando que $\langle S_3 , s_2 \rangle \rightarrow s_3$ podemos llegar a que $\langle (S_1 ; S_2) ; S_3 , s \rangle \rightarrow s'$.

\Leftarrow Usamos un razonamiento similar al paso anterior.

- Demostrar que $S_1 ; S_2$ no es semánticamente equivalente a $S_2 ; S_1$.

Consideramos $x = 1$; $y = x$ frente a $y = x$; $x = 1$ en el estado $s[x \mapsto 2]$.

Ejercicio 2.7

Extender el lenguaje **While** con la sentencia **repeat S until b**.

- Dar las reglas que definen su semántica).

$[\text{repeat}_{\text{bs}} \text{ ff}]$

$\langle \text{repeat } S \text{ until } b , s \rangle \rightarrow s' :- \langle S , s \rangle \rightarrow s_1, \langle \text{repeat } S \text{ until } b , s_1 \rangle \rightarrow s' , B[[b]] s_1 == \text{false}$

$[\text{repeat}_{\text{bs}} \text{ tt}]$ $\langle \text{repeat } S \text{ until } b , s \rangle \rightarrow s' :- \langle S , s \rangle \rightarrow s' , B[[b]] s' == \text{true}$

- Demostrar que `repeat S until b` es semánticamente equivalente a `S; if b then skip else (repeat S until b)`.

Sea $s \in \text{State}$.

\Rightarrow Supongamos que $\langle \text{repeat } S \text{ until } b, s \rangle \rightarrow s'$. Entonces $\langle S, s \rangle \rightarrow s_1$.

Caso 1: $B[[b]] s_1 == \text{false}$

Entonces $\langle \text{repeat } S \text{ until } b, s_1 \rangle \rightarrow s'$. Deducimos que $\langle \text{if } b \text{ then skip else (repeat } S \text{ until } b) , s_1 \rangle \rightarrow s'$ ya que $\langle \text{repeat } S \text{ until } b, s_1 \rangle \rightarrow s'$ y $B[[b]] s_1 == \text{false}$. Por la regla de composición, $\langle S ; \text{if } b \text{ then skip else (repeat } S \text{ until } b) , s \rangle \rightarrow s'$.

Caso 2: $B[[b]] s_1 == \text{true}$

Entonces $s_1 = s'$ luego $\langle S, s \rangle \rightarrow s'$. Además, tenemos que $\langle \text{if } b \text{ then skip else (repeat } S \text{ until } b) , s' \rangle \rightarrow s'$ ya que $\langle \text{skip} , s' \rangle \rightarrow s'$ y $B[[b]] s' == \text{true}$. Por la regla de composición, $\langle S ; \text{if } b \text{ then skip else (repeat } S \text{ until } b) , s \rangle \rightarrow s'$.

\Leftarrow Supongamos que $\langle S ; \text{if } b \text{ then skip else (repeat } S \text{ until } b) , s \rangle \rightarrow s'$. Por la regla [comp_{bs}], tenemos que $\langle S, s \rangle \rightarrow s_1$ y que $\langle \text{if } b \text{ then skip else (repeat } S \text{ until } b) , s_1 \rangle \rightarrow s'$.

Caso 1: $B[[b]] s_1 == \text{false}$

Entonces por [if_{bs} false] tenemos que $\langle \text{repeat } S \text{ until } b, s_1 \rangle \rightarrow s'$. Podemos aplicar [repeat_{bs} ff] sobre esta premisa junto con $\langle S, s \rangle \rightarrow s_1$ para deducir $\langle \text{repeat } S \text{ until } b, s \rangle \rightarrow s'$.

Caso 2: $B[[b]] s_1 == \text{true}$

En este caso, podemos aplicar [repeat_{bs} tt] sobre la premisa $\langle S, s \rangle \rightarrow s_1$ para obtener $\langle \text{repeat } S \text{ until } b, s \rangle \rightarrow s'$.

Ejercicio 2.10

- Demostrar que `repeat S until b` es semánticamente equivalente a `S; while $\neg b$ do S`.

Sea $s \in \text{State}$.

\Rightarrow Supongamos que $\langle \text{repeat } S \text{ until } b, s \rangle \rightarrow s'$. Entonces $\langle S, s \rangle \rightarrow s_1$.

Razonamos por inducción en premisas.

Caso base: $B[[b]] s_1 == \text{true}$

Entonces $\langle S, s \rangle \rightarrow s'$ y $B[[\neg b]] s_1 == \text{false}$. De esto último deducimos por [while_{bs} ff] que $\langle \text{while } \neg b \text{ do } S, s' \rangle \rightarrow s'$, y por tanto por [comp_{bs}] tenemos que $\langle S ; \text{while } \neg b \text{ do } S, s \rangle \rightarrow s'$.

Caso inductivo: $B[[b]] s_1 == \text{false}$

Entonces $\langle S, s \rangle \rightarrow s_1$, $\langle \text{repeat } S \text{ until } b, s_1 \rangle \rightarrow s'$ y $B[[\neg b]] s_1 == \text{true}$. Aplicamos la hipótesis de inducción sobre $\langle \text{repeat } S \text{ until } b, s_1 \rangle \rightarrow s'$, de lo que deducimos que $\langle S ; \text{while } \neg b \text{ do } S, s_1 \rangle \rightarrow s'$. Por [comp_{bs}], deducimos que $\langle S, s_1 \rangle \rightarrow s_2$, $\langle \text{while } \neg b \text{ do } S, s_2 \rangle \rightarrow s'$. Aplicando

[while_{bs} tt] deducimos que $\langle \text{while } \neg b \text{ do } S, s_1 \rangle \rightarrow s'$. Aplicando [comp_{bs}] con la premisa $\langle S, s \rangle \rightarrow s_1$ deducimos que $\langle S ; \text{while } \neg b \text{ do } S, s \rangle \rightarrow s'$.

\Leftarrow Supongamos que $\langle S ; \text{while } \neg b \text{ do } S, s \rangle \rightarrow s'$. Por tanto, $\langle S, s \rangle \rightarrow s_1, \langle \text{while } \neg b \text{ do } S, s_1 \rangle \rightarrow s'$.

Razonamos por inducción en premisas.

Caso base: $B[\neg b] s_1 == \text{false}$

Entonces $B[b] s_1 == \text{true}$ y $\langle \text{while } \neg b \text{ do } S, s_1 \rangle \rightarrow s_1$, luego por el determinismo de la semántica operacional $s_1 = s'$. Aplicamos [repeat_{bs} tt] sobre $\langle S, s \rangle \rightarrow s'$ y obtenemos $\langle \text{repeat } S \text{ until } b, s \rangle \rightarrow s'$.

Caso inductivo: $B[\neg b] s_1 == \text{true}$

Entonces revertimos [while_{bs} tt] para deducir que $\langle S, s_1 \rangle \rightarrow s_2, \langle \text{while } \neg b \text{ do } S, s_2 \rangle \rightarrow s'$. Por [comp_{bs}], deducimos que $\langle S ; \text{while } \neg b \text{ do } S, s_1 \rangle \rightarrow s'$. Ahora aplicamos la hipótesis de inducción para obtener $\langle \text{repeat } S \text{ until } b, s_1 \rangle \rightarrow s'$. Por otra parte, $B[b] s_1 == \text{false}$ y $\langle S, s \rangle \rightarrow s_1$, luego por [repeat_{bs} ff] tenemos que $\langle \text{repeat } S \text{ until } b, s \rangle \rightarrow s'$.

Ejercicio 2.11

- Definir una semántica operacional de paso largo para las expresiones aritméticas, con una relación de transición: $\langle a, s \rangle \rightarrow_{A\text{exp}} z$.

[cons_{bs}] $\langle n, s \rangle \rightarrow N[n]$.

[var_{bs}] $\langle x, s \rangle \rightarrow s[x]$

[op_{bs}] $\langle a_1 \text{ op } a_2, s \rangle \rightarrow z \text{ :- } \langle a_1, s \rangle \rightarrow z_1, \langle a_2, s \rangle \rightarrow z_2, z_1 \text{ op } z_2 = z$

- Demostrar que el significado de cada expresión según la misma, coincide con el definido por \mathcal{A} .

Procedemos a demostrarlo por inducción sobre normas:

- $\langle n, s \rangle \rightarrow N[n] = A[n] s$ por definición de A
- $\langle x, s \rangle \rightarrow s[x] = A[x] s$ por definición de A
- $\langle a_1 \text{ op } a_2, s \rangle \rightarrow z = z_1 \text{ op } z_2$, donde $\langle a_1, s \rangle \rightarrow z_1, \langle a_2, s \rangle \rightarrow z_2$. Por inducción sobre las premisas, $z_1 = A[a_1] s, z_2 = A[a_2] s$. Por tanto, $z = A[a_1] s \text{ op } A[a_2] s = A[a_1 \text{ op } a_2] s$ por definición de A .

Ejercicio 2.12

- Definir una semántica operacional de paso largo para las expresiones booleanas, con una relación de transición: $\langle b, s \rangle \rightarrow_{B\text{exp}} t$.
- Demostrar que el significado de cada expresión según la misma, coincide con el definido por \mathcal{B} .

Ejercicio 2.17

Dar la(s) regla(s) que define(n) la semántica de **repeat S until b**.

[repeat_{ss}] $\langle \text{repeat } S \text{ until } b, s \rangle \Rightarrow \langle S ; \text{if } b \text{ then skip else repeat } S \text{ until } b, s \rangle$

Ejercicio 2.20 + 2.21

- Demostrar que la ejecución de una sentencia es independiente de las sentencias posteriores, enlazando siempre con las de éstas:
si $\langle S_1, s \rangle \Rightarrow^k s'$ entonces $\langle S_1 ; S_2, s \rangle \Rightarrow^k \langle S_2, s' \rangle$.

Lo demostramos por inducción sobre la longitud de la cadena de derivación.

Caso base: $k = 1$.

Entonces tenemos que $\langle S_1, s \rangle \Rightarrow s'$. Aplicando [comp2_{ss}] deducimos que $\langle S_1 ; S_2, s \rangle \Rightarrow \langle S_2, s' \rangle$

Caso inductivo: $k + 1$.

Suponemos que $\langle S_1, s \rangle \Rightarrow^{k+1} s'$. Entonces tenemos que $\langle S_1, s \rangle \Rightarrow \langle S_{\text{aux}}, s_1 \rangle$, $\langle S_{\text{aux}}, s_1 \rangle \Rightarrow^k s'$. Aplicando [comp1_{ss}] deducimos que $\langle S_1 ; S_2, s \rangle \Rightarrow \langle S_{\text{aux}} ; S_2, s_1 \rangle$. Por otra parte, aplicando la h.i. a $\langle S_{\text{aux}}, s_1 \rangle \Rightarrow^k s'$ deducimos que $\langle S_{\text{aux}} ; S_2, s_1 \rangle \Rightarrow^k \langle S_2, s' \rangle$. Por tanto, $\langle S_1 ; S_2, s \rangle \Rightarrow^{k+1} \langle S_2, s' \rangle$.

- Sin embargo, si $\langle S_1 ; S_2, s \rangle \Rightarrow^* \langle S_2, s' \rangle$, no necesariamente $\langle S_1, s \rangle \Rightarrow^* s'$.

Consideramos $\langle \text{skip} ; \text{while true do } x:=1, s \rangle \Rightarrow \langle \text{while true do } x:=1, s \rangle \Rightarrow^* \langle \text{while true do } x:=1, s[x \rightarrow 1] \rangle$.

No es cierto que $\langle \text{skip}, s \rangle \Rightarrow^* s[x \rightarrow 1]$ para todo s .

Ejercicio 2.22

Demostrar que la semántica de paso corto es (paso a paso) **determinista**:

$$\langle S, s \rangle \Rightarrow \gamma \wedge \langle S, s \rangle \Rightarrow \gamma' \implies \gamma = \gamma'$$

Supongamos que $\langle S, s \rangle \Rightarrow \gamma$.

Razonamos por inducción sobre normas:

Casos base:

Si S es de la forma $x:=a$, skip ó while b do S solo existe una regla aplicable (respectivamente, [ass_{ss}], [skip_{ss}], [while_{ss}]).

Si S es de la forma if b then S_1 else S_2 entonces o bien $B[b]s == \text{true}$ o bien $B[b]s == \text{false}$, y estos son casos incompatibles y exhaustivos con sendas únicas norma de derivación asociadas, [if_{ss} tt], [if_{ss} ff].

Caso inductivo:

Si S es de la forma $S_1 ; S_2$, entonces por la h.i. tenemos que $\langle S_1, s \rangle \Rightarrow \gamma$ para un único γ .

Tenemos dos casos:

Caso 1: $\gamma = \langle S', s'' \rangle$. En este caso la única regla aplicable es [comp_{ss} 1]

Caso 2: $\gamma = s''$. En este caso la única regla aplicable es [comp_{ss} 2]

Ejercicio 2.23

Demostrar que las siguientes sentencias son semánticamente equivalentes:

- $S ; \text{skip}$ y S

Sea $s \in \text{State}$. Razonamos por inducción sobre longitud de la cadena de derivación.

Caso base: $\langle S ; \text{skip}, s \rangle \Rightarrow \langle \text{skip}, s' \rangle \Rightarrow s'$

Entonces por [comp_{ss} 2] tenemos que $\langle S, s \rangle \Rightarrow s'$, luego ambas son equivalentes.

Caso inductivo: $\langle S ; \text{skip}, s \rangle \Rightarrow \langle S_1 ; \text{skip}, s_1 \rangle$

Entonces por [comp_{ss} 1] tenemos que $\langle S, s \rangle \Rightarrow^k \langle S_1, s_1 \rangle$. Por otra parte, aplicando la h.i., tenemos que $\langle S_1 ; \text{skip}, s_1 \rangle$ es equivalente a $\langle S_1, s_1 \rangle$. Como $\langle S ; \text{skip}, s \rangle$ y $\langle S, s \rangle$ derivan en normas equivalentes, deducimos que son equivalentes.

- $\text{while } b \text{ do } S \text{ y if } b \text{ then } (S ; \text{while } b \text{ do } S) \text{ else skip}$

Sea $s \in \text{State}$.

[while_{ss}] $\langle \text{while } b \text{ do } S, s \rangle \Rightarrow \langle \text{if } b \text{ then } (S ; \text{while } b \text{ do } S) \text{ else skip}, s \rangle$

Luego ambas sentencias son semánticamente equivalentes.

- $S_1 ; (S_2 ; S_3)$ y $(S_1 ; S_2) ; S_3$

Sea $s \in \text{State}$. Razonamos por inducción sobre longitud de la cadena de derivación.

Caso base: $\langle S_1, s \rangle \Rightarrow s_1$

Entonces por [comp_{ss} 2] tenemos que $\langle S_1 ; (S_2 ; S_3), s \rangle \Rightarrow \langle S_2 ; S_3, s_1 \rangle$ y además que $\langle S_1 ; S_2, s \rangle \Rightarrow \langle S_2, s_1 \rangle$. Aplicando [comp_{ss} 1] a la última premisa obtenemos

$\langle (S_1 ; S_2) ; S_3, s \rangle \Rightarrow \langle S_2 ; S_3, s_1 \rangle$. Por tanto, como $\langle S_1 ; (S_2 ; S_3), s \rangle$ y $\langle (S_1 ; S_2) ; S_3, s \rangle$ derivan en $\langle S_2 ; S_3, s_1 \rangle$ tenemos que son equivalentes.

Caso inductivo: $\langle S_1, s \rangle \Rightarrow \langle S', s' \rangle$

Entonces aplicando [comp_{ss} 1] tenemos que $\langle S_1 ; (S_2 ; S_3), s \rangle \Rightarrow \langle S' ; (S_2 ; S_3), s_1 \rangle$ y además que $\langle S_1 ; S_2, s \rangle \Rightarrow \langle S' ; S_2, s_1 \rangle$, luego aplicando de nuevo [comp_{ss} 1] obtenemos $\langle (S_1 ; S_2) ; S_3, s \rangle \Rightarrow \langle (S' ; S_2) ; S_3, s_1 \rangle$. Aplicando la hipótesis de inducción, tenemos que $\langle S' ; (S_2 ; S_3), s_1 \rangle$ y $\langle (S' ; S_2) ; S_3, s_1 \rangle$ son equivalentes. Como las sentencias de interés derivan en sentencias equivalentes, son equivalentes.

Ejercicio 2.24

Demostrar que $\text{repeat } S \text{ until } b$ es semánticamente equivalente a $S ; \text{while } \neg b \text{ do } S$.

Razonamos por inducción sobre longitud de la cadena de derivación.

[repeat_{ss}] $\langle \text{repeat } S \text{ until } b, s \rangle \Rightarrow \langle S ; \text{if } b \text{ then skip else repeat } S \text{ until } b, s \rangle$

Caso 1: $\langle S, s \rangle \Rightarrow^* s'$

[comp_{ss} 3] $\langle S ; \text{if } b \text{ then skip else repeat } S \text{ until } b, s \rangle \Rightarrow^* \langle \text{if } b \text{ then skip else repeat } S \text{ until } b, s' \rangle$

[comp_{ss} 3] $\langle S ; \text{while } \neg b \text{ do } S, s \rangle \Rightarrow^* \langle \text{while } \neg b \text{ do } S, s' \rangle$

[while_{ss}] $\langle \text{while } \neg b \text{ do } S, s' \rangle \Rightarrow \langle \text{if } \neg b \text{ then } (S ; \text{while } \neg b \text{ do } S) \text{ else skip}, s' \rangle$

Caso 1.1: $B[[b]] s' == \text{true}$

[if_{ss} tt] $\langle \text{if } b \text{ then skip else repeat } S \text{ until } b, s' \rangle \Rightarrow \langle \text{skip}, s' \rangle$

[if_{ss} ff] $\langle \text{if } \neg b \text{ then } (S ; \text{while } \neg b \text{ do } S) \text{ else skip}, s' \rangle \Rightarrow \langle \text{skip}, s' \rangle$

Combinando las afirmaciones anteriores, comprobamos que ambas sentencias de interés derivan en $\langle \text{skip}, s' \rangle$.

Caso 1.2: $B[[b]] s' == \text{false}$

[if_{ss} ff] $\langle \text{if } b \text{ then skip else repeat } S \text{ until } b, s' \rangle \Rightarrow \langle \text{repeat } S \text{ until } b, s' \rangle$

[if_{ss} tt] $\langle \text{if } \neg b \text{ then } (S ; \text{while } \neg b \text{ do } S) \text{ else skip}, s' \rangle \Rightarrow \langle S ; \text{while } \neg b \text{ do } S, s' \rangle$

Por h.i. deducimos que $\langle \text{repeat } S \text{ until } b, s' \rangle$ es equivalente a $\langle S ; \text{while } \neg b \text{ do } S, s' \rangle$, luego las sentencias de interés son equivalentes.

Caso 2: $\langle S, s \rangle$ no termina. Entonces $\langle S ; \text{if } b \text{ then skip else repeat } S \text{ until } b, s \rangle$ y $\langle S ; \text{while } \neg b \text{ do } S, s \rangle$ no terminan, luego las sentencias de interés no terminan.

Ejercicio 2.25

Determinar si la **equivalencia semántica** de S_1 y S_2 nos dice más o menos que el hecho de que $\mathcal{S}_{ss}[[S_1]] = \mathcal{S}_{ss}[[S_2]]$.

La equivalencia semántica nos dice más, puesto que afirma igualdad incluso en caso de bloqueo, lo cual no está garantizado por la igualdad de significado.

Teorema 5

$$\forall S \in \mathbf{Stm} \quad \mathcal{S}_{bs}[[S]] = \mathcal{S}_{ss}[[S]]$$

Ejercicio 2.29

Extender la demostración del Teorema 5 incluyendo el tratamiento de la sentencia **repeat S until b** .

Lema 6: $\forall S \in \mathbf{Stm} \quad \forall s, s' \in \mathbf{State} \quad \langle S, s \rangle \rightarrow s' \implies \langle S, s \rangle \Rightarrow^* s'$

Supongamos que $\langle \text{repeat } S \text{ until } b, s \rangle \rightarrow s'$. Razonamos por inducción completa sobre el árbol de derivación.

[repeat_{ss}] $\langle \text{repeat } S \text{ until } b , s \rangle \Rightarrow \langle S ; \text{if } b \text{ then skip else repeat } S \text{ until } b , s \rangle$

Caso 1: $\langle \text{repeat } S \text{ until } b , s \rangle \rightarrow s'$ se deriva por la regla [repeat_{bs} tt]

Entonces tenemos que $\langle S , s \rangle \rightarrow s'$, $B[\![b]\!] s' == \text{true}$.

Por h.i. tenemos que $\langle S , s \rangle \Rightarrow^* s'$.

[comp_{ss} 3] $\langle S ; \text{if } b \text{ then skip else repeat } S \text{ until } b , s \rangle \Rightarrow^* \langle \text{if } b \text{ then skip else repeat } S \text{ until } b , s' \rangle$

[if_{ss} tt] $\langle \text{if } b \text{ then skip else repeat } S \text{ until } b , s' \rangle \Rightarrow \langle \text{skip} , s' \rangle$

[skip_{ss}] $\langle \text{skip} , s' \rangle \Rightarrow s'$

Luego $\langle \text{repeat } S \text{ until } b , s \rangle \Rightarrow^* s'$

Caso 2: $\langle \text{repeat } S \text{ until } b , s \rangle \rightarrow s'$ se deriva por la regla [repeat_{bs} ff]

Entonces tenemos que $\langle S , s \rangle \rightarrow s_1$, $\langle \text{repeat } S \text{ until } b , s_1 \rangle \rightarrow s'$, $B[\![b]\!] s_1 == \text{false}$.

Por la h.i. tenemos que $\langle S , s \rangle \Rightarrow^* s'$, $\langle \text{repeat } S \text{ until } b , s_1 \rangle \Rightarrow^* s'$.

[comp_{ss} 3] $\langle S ; \text{if } b \text{ then skip else repeat } S \text{ until } b , s \rangle \Rightarrow^* \langle \text{if } b \text{ then skip else repeat } S \text{ until } b , s_1 \rangle$

[if_{ss} ff] $\langle \text{if } b \text{ then skip else repeat } S \text{ until } b , s_1 \rangle \Rightarrow \langle \text{repeat } S \text{ until } b , s_1 \rangle$.

Luego $\langle \text{repeat } S \text{ until } b , s \rangle \Rightarrow^* s'$

Lema 7: $\forall S \in \mathbf{Stm} \forall s, s' \in \mathbf{State} \forall k \in \mathbb{N} \quad \langle S, s \rangle \Rightarrow^k s' \implies \langle S, s \rangle \rightarrow s'$

Supongamos que $\langle \text{repeat } S \text{ until } b , s \rangle \Rightarrow^* s'$. Razonamos por inducción sobre la longitud de la cadena de derivación.

[repeat_{ss}] $\langle \text{repeat } S \text{ until } b , s \rangle \Rightarrow \langle S ; \text{if } b \text{ then skip else repeat } S \text{ until } b , s \rangle$

Como la ejecución termina, deducimos que existe s_1 tal que $\langle S , s \rangle \Rightarrow^* s_1$. Por h.i. deducimos que $\langle S , s \rangle \rightarrow s_1$

[comp_{ss} 3] $\langle S ; \text{if } b \text{ then skip else repeat } S \text{ until } b , s \rangle \Rightarrow^* \langle \text{if } b \text{ then skip else repeat } S \text{ until } b , s_1 \rangle$.

Caso 1: $B[\![b]\!] s_1 == \text{true}$

[if_{ss} tt] $\langle \text{if } b \text{ then skip else repeat } S \text{ until } b , s_1 \rangle \Rightarrow \langle \text{skip} , s_1 \rangle$

[skip_{ss}] $\langle \text{skip} , s_1 \rangle \Rightarrow s_1$. Luego $s_1 = s'$ por determinismo de la semántica.

Por otra parte,

[repeat_{bs} tt] $\langle \text{repeat } S \text{ until } b , s \rangle \rightarrow s' :- \langle S , s \rangle \rightarrow s' , B[\![b]\!] s' == \text{true}$.

Es aplicable, luego $\langle \text{repeat } S \text{ until } b , s \rangle \rightarrow s'$.

Caso 2: $B[\![b]\!] s_1 == \text{false}$

[if_{ss} ff] $\langle \text{if } b \text{ then skip else repeat } S \text{ until } b , s_1 \rangle \Rightarrow \langle \text{repeat } S \text{ until } b , s_1 \rangle$

Por determinismo de la semántica, tenemos que $\langle \text{repeat } S \text{ until } b , s_1 \rangle \Rightarrow^* s'$.

Luego por la h.i. tenemos que $\langle \text{repeat } S \text{ until } b , s_1 \rangle \rightarrow s'$

Luego podemos aplicar:

[repeat_{bs} ff] $\langle \text{repeat } S \text{ until } b , s \rangle \rightarrow s' :- \langle S , s \rangle \rightarrow s_1 , \langle \text{repeat } S \text{ until } b , s_1 \rangle \rightarrow s' , B[\![b]\!] s_1 == \text{false}$.

Y por tanto $\langle \text{repeat } S \text{ until } b , s \rangle \rightarrow s'$.

Tema 3: Más sobre semántica operacional

Ejercicio 3.2

Extender el lenguaje **While** con la sentencia **assert b before S**.

- Comprobamos que **b se cumple antes** de ir a ejecutar **S**
- Definir adecuadamente su semántica de paso corto.

[**assert_{ss}**] $\langle \text{assert } b \text{ before } S, s \rangle \Rightarrow \langle S, s \rangle :- B[b] \text{ s == true}$

- Demostrar que **assert true before S** es semánticamente equivalente a **S**, pero
assert false before S no es semánticamente equivalente ni a **skip** ni a **while true do skip**

[**assert_{ss}**] $\langle \text{assert true before } S, s \rangle \Rightarrow \langle S, s \rangle$

Luego $\langle \text{assert true before } S, s \rangle$ es semánticamente equivalente a $\langle S, s \rangle$.

$\langle \text{assert false before } S, s \rangle$ termina inmediatamente para todo s , mientras que $\langle \text{skip}, s \rangle \Rightarrow s$ y $\langle \text{while true do skip}, s \rangle$ cicla.

- ¿A quién sí sería equivalente?.

assert false before S es semánticamente equivalente a **abort**.

Ejercicio 3.14

En particular, la semántica con ámbito estático de la extensión del lenguaje **While** que acabamos de definir, es una semántica alternativa del lenguaje **While** original.

Formular y demostrar la equivalencia entre esta nueva semántica y la de **paso largo** definida originalmente.

Demostración de equivalencia

Queremos ver que si S es una sentencia del lenguaje While no extendido, $s = sto \circ env_V$ y $s' = sto' \circ env_V$; entonces $\langle S, s \rangle \rightarrow s' \text{ sii } env_V, env_P \vdash \langle S, sto \rangle \rightarrow sto'$.

Lo demostraremos por inducción sobre reglas.

[**ass_{bs}**] $\langle x := a, s \rangle \rightarrow s[x \mapsto A[a](s)] = s'$

[**ass'_{bs}**] $env_V, env_P \vdash \langle x := a, sto \rangle \rightarrow sto[env_V x \mapsto A[a](sto \circ env_V)] = sto'$

Observamos que si $s = sto \circ env_V$, entonces $s' = s[x \mapsto A[a](s)] = sto' \circ env_V$. En efecto, $s' x = A[a](s) = A[a](sto \circ env_V) = sto' \circ env_V x$; $s' y = s y = sto \circ env_V y = sto' \circ env_V y$ (requerimos que env_V sea inyectivo). Por tanto cuando una es derivable también lo es la otra.

[**skip_{bs}**] $\langle skip, s \rangle \rightarrow s$

[**skip'_{bs}**] $env_V, env_P \vdash \langle skip, sto \rangle \rightarrow sto$

Claramente si $s = sto \circ env_V$ entonces $s' = s = sto \circ env_V = sto' \circ env_V$.

[comp'_{bs}] $\langle S_1 ; S_2 , s \rangle \rightarrow s'' :- \langle S_1 , s \rangle \rightarrow s', \langle S_2 , s' \rangle \rightarrow s''$

[comp'_{bs}] $env_V, env_P \vdash \langle S_1 ; S_2 , sto \rangle \rightarrow sto'' :-$

$env_V, env_P \vdash \langle S_1 , sto \rangle \rightarrow sto', env_V, env_P \vdash \langle S_2 , sto' \rangle \rightarrow sto''$

Supongamos que $s = sto \circ env_V$.

\Rightarrow Supongamos que $\langle S_1 ; S_2 , s \rangle \rightarrow s''$ es derivable. Entonces tenemos que $\langle S_1 , s \rangle \rightarrow s'$, $\langle S_2 , s' \rangle \rightarrow s''$ son derivables. Aplicando la h.i. obtenemos que $env_V, env_P \vdash \langle S_1 , sto \rangle \rightarrow sto', env_V, env_P \vdash \langle S_2 , sto' \rangle \rightarrow sto''$, con $s = sto \circ env_V$, $s' = sto' \circ env_V$, $s'' = sto'' \circ env_V$.

Aplicando [comp'_{bs}] llegamos al resultado deseado, $env_V, env_P \vdash \langle S_1 ; S_2 , sto \rangle \rightarrow sto''$.

\Leftarrow Similar al recíproco.

[if'^{tt}_{bs}] ...

[if'^{ff}_{bs}] ...

[while'^{tt}_{bs}]

[while'^{ff}_{bs}]

Ejercicio 3.15

Modificar la sintaxis de la declaración de procedimientos para que reciban ahora siempre dos **parámetros con paso por valor**:

$$D_P ::= proc\ p(x_1, x_2)\ is\ S; D_P \mid \epsilon$$

$$S ::= x := a \mid \dots \mid call\ p(a_1, a_2)$$

Los entornos de procedimiento serán ahora elementos de:

$$Env_P = Pname \hookrightarrow Var \times Var \times Stm \times Env_V \times Env_P.$$

Modificar la semántica convenientemente, dando en particular nuevas reglas para las llamadas a procedimientos (con o sin recursión).

Las siguientes reglas no requieren modificación: [ass'_{bs}], [skip'_{bs}], [comp'_{bs}], [if'^{tt}_{bs}], [if'^{ff}_{bs}], [while'^{tt}_{bs}], [while'^{ff}_{bs}].

La regla [block_{bs}] se actualiza cambiando udp_P . La nueva definición de udp_P seguirá las siguientes reglas:

$$udp_P(\ proc\ p(x_1, x_2)\ is\ S ; D_P, env_V, env_P) =$$

$$udp_P(S ; D_P, env_V, env_P[p \mapsto (x_1, x_2, S, env_V[x_1 \mapsto next, x_2 \mapsto next], env_P)])$$

$$udp_P(\ \epsilon, env_V, env_P) = env_P$$

Las reglas [var_{bs}] y [none_{bs}] no requieren modificación.

La regla [call_{bs}] se modifica de la siguiente manera:

[call_{bs}] env_V, env_P ⊢ ⟨call p(a₁, a₂), sto⟩ → sto' :-
 env'_V, env'_P ⊢ ⟨S, sto[sto ∘ env'_V x₁ ↦ A[[a₁]](sto ∘ env_V), sto ∘ env'_V x₂ ↦ A[[a₂]](sto ∘ env_V)]] → sto'
 donde env_P p = (x₁, x₂, S, env'_V, env'_P)

La regla [call^{rec}_{bs}] se modifica de la siguiente manera:

[call^{rec}_{bs}] env_V, env_P ⊢ ⟨call p(a₁, a₂), sto⟩ → sto' :-
 env'_V, env'_P [p ↦ (x₁, x₂, S, env'_V, env'_P)] ⊢ ⟨S, sto[sto ∘ env'_V x₁ ↦ A[[a₁]](sto ∘ env_V), sto ∘ env'_V x₂ ↦ A[[a₂]](sto ∘ env_V)]] → sto'
 donde env_P p = (x₁, x₂, S, env'_V, env'_P)

Tema 4: Implementación correcta

Ejercicios recomendados: 4.7, 4.8, 4.14, 4.16, 4.17, 4.24, 4.25, 4.27 y 4.28

Ejercicio 4.4

Demostrar que se puede extender al tiempo el código y la pila de evaluación de MA, sin que cambie su comportamiento, en tanto se va ejecutando dicho código:

$$\langle c_1, e_1, s \rangle \triangleright^k \langle c', e', s' \rangle \implies \langle c_1 : c_2, e_1 : e_2, s \rangle \triangleright^k \langle c' : c_2, e' : e_2, s' \rangle$$

Observamos que ninguna de las instrucciones máquina individuales modifica la cola del código o la pila.

Ahora procedemos por inducción sobre la longitud de la secuencia de cómputo. Si el cómputo de interés lleva k+1 pasos, realizamos 1 paso. Este paso no modifica la cola, por la observación anterior. Ahora quedan k pasos, y podemos aplicar la h.i.

Por tanto si $\langle c_1, e_1, s \rangle \triangleright^k \langle c', e', s' \rangle$ entonces $\langle c_1 : c_2, e_1 : e_2, s \rangle \triangleright^k \langle c' : c_2, e' : e_2, s' \rangle$.

Ejercicio 4.5

Demostrar que la ejecución completa de una secuencia de instrucciones siempre puede fraccionarse, instrucción a instrucción:

$$\begin{aligned} \langle c_1 : c_2, e, s \rangle \triangleright^k \langle \epsilon, e'', s'' \rangle &\implies \\ \exists \langle \epsilon, e', s' \rangle \exists k_1, k_2 \quad &\langle c_1, e, s \rangle \triangleright^{k_1} \langle \epsilon, e', s' \rangle \wedge \langle c_2, e', s' \rangle \triangleright^{k_2} \langle \epsilon, e'', s'' \rangle \\ \text{con } k &= k_1 + k_2. \end{aligned}$$

Supongamos que $\langle c_1, e, s \rangle$ tras k pasos no ha acabado; ie $\langle c_1, e, s \rangle \triangleright^k \langle c', e', s' \rangle$ con $c' \neq \epsilon$. Entonces por el ejercicio anterior tenemos que $\langle c_1 : c_2, e, s \rangle \triangleright^k \langle c' : c_2, e', s' \rangle$, en contradicción con la hipótesis del enunciado.

Por tanto, existe un $k_1 \leq k$ tal que $\langle c_1, e, s \rangle \triangleright^{k_1} \langle \epsilon, e', s' \rangle$, y por el ejercicio anterior $\langle c_1 : c_2, e, s \rangle \triangleright^{k_1} \langle c_2, e', s' \rangle$.

Por el determinismo de MA, tenemos que $\langle c_2, e', s' \rangle \triangleright^{k_2} \langle \epsilon, e'', s'' \rangle$, donde $k_2 = k - k_1$

Ejercicio 4.6

Demostrar que la semántica de MA es determinista:

$$\gamma \triangleright \gamma' \wedge \gamma \triangleright \gamma'' \implies \gamma' = \gamma''$$

Observamos que por cada instrucción sólo existe a lo sumo una regla aplicable asociada.

Ejercicio 4.7

Modificar MA para referirse a las variables por su dirección:

- las configuraciones pasarán a ser $\langle c, e, m \rangle$, donde $m \in \mathbb{Z}^*$ representa una memoria RAM ($m[n]$ selecciona el n -ésimo valor de la lista m);
- sustituir $\text{FETCH-}x$ y $\text{STORE-}x$ por $\text{GET-}n$ y $\text{PUT-}n$, con $n \in \mathbb{N}$ (representando una dirección).

Dar la semántica operacional de la máquina modificada MA_1 .

$$\langle \text{GET-}n : c, e, m \rangle \triangleright \langle c, m[n] : e, m \rangle$$

$$\langle \text{PUT-}n : c, z : e, m \rangle \triangleright \langle c, e, m[n] \rightarrow z \rangle$$

Ejercicio 4.8

Modificar MA_1 para pasar a tener instrucciones de salto no estructuradas:

- las configuraciones pasan a ser $\langle pc, c, e, m \rangle$, donde $pc \in \mathbb{N}$ es el contador de programa, que apunta a la siguiente instrucción a ejecutar en c ($c[pc]$).
- sustituir $\text{BRANCH}(\dots, \dots)$ y $\text{LOOP}(\dots, \dots)$ por $\text{LABEL-}l$, $\text{JUMP-}l$ y $\text{JUMPFALSE-}l$, representando $\text{LABEL-}l$, con $l \in \mathbb{N}$, una posición en c .

Dar la semántica operacional de la máquina modificada MA_2 .

$$\langle pc, c, e, m \rangle \triangleright \langle pc+1, c, e, m \rangle$$

donde $c[pc] = \text{LABEL-}l$

$$\langle pc, c, e, m \rangle \triangleright \langle pc', c, e, m \rangle$$

donde $c[pc] = \text{JUMP-}l$, $pc' = \max \{ i \mid 0 \leq i \leq pc, c[i] = \text{LABEL-}l \}$

$$\langle pc, c, z:e, m \rangle \triangleright \langle pc', c, e, m \rangle$$

Donde $z = \text{ff}$, $c[pc] = \text{JUMPFALSE-}l$, $pc' = \max \{ i \mid 0 \leq i \leq pc, c[i] = \text{LABEL-}l \}$

$$\langle pc, c, z:e, m \rangle \triangleright \langle pc, c, e, m \rangle$$

Donde $z = \text{tt}$, $c[pc] = \text{JUMPFALSE-}l$

Ejercicio 4.11

Aunque $\mathcal{A}[(a_1 + a_2) + a_3] = \mathcal{A}[a_1 + (a_2 + a_3)]$, demostrar que, en general, $\mathcal{CA}[(a_1 + a_2) + a_3] \neq \mathcal{CA}[a_1 + (a_2 + a_3)]$. Sin embargo, podemos capturar en qué sentido se *comportan* de forma *suficientemente similar*.

Ejercicio 4.19

Demostrar un resultado similar para las expresiones booleanas:

$$\forall b \in \mathbf{Bexp}, \forall s \in \mathbf{State} \quad \langle \mathcal{CB}[b], \varepsilon, s \rangle \triangleright^* \langle \varepsilon, \mathcal{B}[b]s, s \rangle$$

Demostrar que además, en todas las configuraciones intermedias de esta secuencia de cálculo, la pila de evaluación es no vacía.

Ejercicio 4.14

Considerar la extensión de **While** con la sentencia **repeat S until b**, y generar código para esta sentencia, sin ampliar el conjunto de instrucciones de MA.

$$CS[\text{repeat } S \text{ until } b] = CS[S] : \text{LOOP}(\mathcal{CB}[b] : \text{NEG}, CS[S])$$

Ejercicio 4.16

Generar el código para traducir **While** a la máquina MA₁.

Utilizar entornos $env : \mathbf{Var} \rightarrow \mathbb{N}$, que asocian a cada variable su dirección.

$$CS[x := a] = CA[a] : \text{PUT-}(env x)$$

CB sin cambios

$$CA[x] = \text{GET-}(env x)$$

Ejercicio 4.17

Generar el código para traducir **While** a la máquina MA₂.

Garantizar la unicidad de las etiquetas incorporando un parámetro adicional:
siguiente etiqueta sin usar.

$$CS[\text{if } b \text{ then } S_1 \text{ else } S_2, \text{next}] = (\mathcal{CB}[b] : \text{JUMPFALSE-}(\text{next}) : CS[S_1, \text{next+2}] : \\ \text{JUMP-}(\text{next+1}) : \text{LABEL-}(\text{next}) : CS[S_2, \text{next'}] : \text{LABEL-}(\text{next+1}), \text{next''}) \\ \text{donde next'} = \text{snd } CS[S_1, \text{next+2}], \text{next''} = \text{snd } CS[S_2, \text{next'}]$$

$$CS[\text{while } b \text{ do } S, \text{next}] = (\text{LABEL-}(\text{next}) : \mathcal{CB}[b] : \text{JUMPFALSE-}(\text{next+1}) : CS[S, \text{next+2}] : \\ \text{JUMP-}(\text{next}) : \text{LABEL-}(\text{next+1}), \text{next'}) \\ \text{donde next'} = \text{snd } CS[S_1, \text{next+2}]$$

($\text{CS}[\![S, \text{next}]\!]$ denota por abuso de notación a $\text{fst } \text{CS}[\![S, \text{next}]\!]$)

Ejercicio 4.23

Modificar la función de traducción pasando a tener $\text{CS}[\![\text{skip}]\!] = \varepsilon$.

¿Cómo afecta este cambio a la demostración del Teorema 9?

⇒ No hay cambios

⇐ Queda demostrar el caso básico $k = 0$ pues ya no es trivial.

Ahora, $\text{CS}[\![S]\!] = \varepsilon$ para programas del estilo $S_n = \text{skip}; \text{skip}; \text{skip} \dots (n) \dots; \text{skip}$; y sólamente para este tipo de programas.

Supongamos que $\langle \text{CS}[\![S_n]\!] = \varepsilon, \varepsilon, s \rangle \triangleright^0 \langle \varepsilon, \varepsilon, s' \rangle$. Claramente $s' = s$.

Caso base: $n=1$

Aplicando $[\text{skip}_{\text{bs}}]$ obtenemos que $\langle S_1 = \text{skip}, s \rangle \rightarrow s$

Caso inductivo:

Por la h.i. tenemos que $\langle S_n, s \rangle \rightarrow s$.

Aplicando $[\text{comp}_{\text{bs}}]$ obtenemos que $\langle \text{skip}; S_n = S_{n+1}, s \rangle \rightarrow s$.

Concluimos que $\langle S_n, s \rangle \rightarrow s$ para todo n , lo que concluye la prueba para el caso $k=0$.

El resto de la prueba es idéntico.

Ejercicio 4.24

Extender la demostración del Teorema 9 para incluir la sentencia
repeat S until b.

⇒ Supongamos que $\langle \text{repeat } S \text{ until } b, s \rangle \rightarrow s'$ es derivable.

Caso 1: Se deriva mediante $[\text{repeat}_{\text{bs}} \text{ ff}]$

$\langle \text{repeat } S \text{ until } b, s \rangle \rightarrow s' :- \langle S, s \rangle \rightarrow s_1, \langle \text{repeat } S \text{ until } b, s_1 \rangle \rightarrow s', B[\![b]\!] s_1 == \text{false}$

Entonces tenemos que $\langle S, s \rangle \rightarrow s_1, \langle \text{repeat } S \text{ until } b, s_1 \rangle \rightarrow s', B[\![b]\!] s_1 == \text{false}$.

Aplicando la hipótesis de inducción, tenemos que

$\langle \text{CS}[\![S]\!], \varepsilon, s \rangle \triangleright^* \langle \varepsilon, \varepsilon, s_1 \rangle,$

$\langle \text{CS}[\![\text{repeat } S \text{ until } b]\!], \varepsilon, s_1 \rangle \triangleright \langle \text{CB}[\![b]\!] : \text{NEG} : \text{BRANCH}(\text{CS}[\![S]\!]) : \text{LOOP}(\text{CB}[\![b]\!] : \text{NEG}, \text{CS}[\![S]\!]), \varepsilon, s_1 \rangle \triangleright^* \langle \varepsilon, \varepsilon, s' \rangle$

Ahora bien,

$\text{CS}[\![\text{repeat } S \text{ until } b]\!] = \text{CS}[\![S]\!] : \text{LOOP}(\text{CB}[\![b]\!] : \text{NEG}, \text{CS}[\![S]\!])$

Por tanto $\langle \text{CS}[\![\text{repeat } S \text{ until } b]\!], \varepsilon, s \rangle \triangleright^* \langle \text{LOOP}(\text{CB}[\![b]\!] : \text{NEG}, \text{CS}[\![S]\!]), \varepsilon, s_1 \rangle \triangleright \langle \text{CB}[\![b]\!] : \text{NEG} : \text{BRANCH}(\text{CS}[\![S]\!]) : \text{LOOP}(\text{CB}[\![b]\!] : \text{NEG}, \text{CS}[\![S]\!]), \text{NOOP}, \varepsilon, s_1 \rangle$

Por el ejercicio 4.19, tenemos que $\langle \text{CB}[\![b]\!], \varepsilon, s_1 \rangle \triangleright^* \langle \varepsilon, B[\![b]\!] == \text{ff}, s_1 \rangle$

Por el ejercicio 4.4 tenemos que

$\langle \text{CB}[\![b]\!] : \text{NEG} : \text{BRANCH}(\text{CS}[\![S]\!]) : \text{LOOP}(\text{CB}[\![b]\!] : \text{NEG}, \text{CS}[\![S]\!]), \text{NOOP}, \varepsilon, s_1 \rangle \triangleright^*$

$\langle \text{NEG} : \text{BRANCH}(\text{CS}[\![S]\!]) : \text{LOOP}(\text{CB}[\![b]\!]) : \text{NEG}, \text{CS}[\![S]\!]), \text{NOOP} \rangle, \text{ff}, s_1 \rangle \triangleright$
 $\langle \text{BRANCH}(\text{CS}[\![S]\!]) : \text{LOOP}(\text{CB}[\![b]\!]) : \text{NEG}, \text{CS}[\![S]\!]), \text{NOOP} \rangle, \text{tt}, s_1 \rangle \triangleright$
 $\langle \text{CS}[\![S]\!] : \text{LOOP}(\text{CB}[\![b]\!]) : \text{NEG}, \text{CS}[\![S]\!]), \varepsilon, s_1 \rangle \text{ (h.i.)} \triangleright^* \langle \varepsilon, \varepsilon, s' \rangle$

Caso 2: Se deriva mediante [repeat_{bs} tt]

$\langle \text{repeat } S \text{ until } b, s \rangle \rightarrow s' : - \langle S, s \rangle \rightarrow s', B[\![b]\!] s' == \text{true}$

Entonces tenemos que $\langle S, s \rangle \rightarrow s', B[\![b]\!] s' == \text{true}$

Aplicando la hipótesis de inducción, tenemos que $\langle \text{CS}[\![S]\!], \varepsilon, s \rangle \triangleright^* \langle \varepsilon, \varepsilon, s' \rangle$

Aplicando una derivación similar al caso anterior, llegamos a que

$\langle \text{CS}[\![\text{repeat } S \text{ until } b]\!], \varepsilon, s \rangle \triangleright^* \langle \text{BRANCH}(\text{CS}[\![S]\!]) : \text{LOOP}(\text{CB}[\![b]\!]) : \text{NEG}, \text{CS}[\![S]\!]), \text{NOOP} \rangle, \text{ff}, s' \rangle \triangleright \langle \text{NOOP}, \varepsilon, s' \rangle \triangleright \langle \varepsilon, \varepsilon, s' \rangle$

\Leftarrow Supongamos que $\langle \text{CS}[\![\text{repeat } S \text{ until } b]\!], \varepsilon, s \rangle \triangleright^* \langle \varepsilon, e, s' \rangle$

Entonces $e = \varepsilon$. En efecto, las traducciones CS mantienen el invariante de dejar la pila vacía si empezó estando vacía, y repeat no es excepción.

Ahora queremos demostrar que $\langle \text{repeat } S \text{ until } b, s \rangle \rightarrow s'$ es derivable.

Por la h.i. tenemos que como $\langle S, \varepsilon, s \rangle \rightarrow s_1$

$\langle \text{CS}[\![\text{repeat } S \text{ until } b]\!], \varepsilon, s \rangle = \langle \text{CS}[\![S]\!] : \text{LOOP}(\text{CB}[\![b]\!]) : \text{NEG}, \text{CS}[\![S]\!], \varepsilon, s \rangle \triangleright^*$
 $\langle \text{LOOP}(\text{CB}[\![b]\!]) : \text{NEG}, \text{CS}[\![S]\!], \varepsilon, s_1 \rangle \triangleright \langle \text{CB}[\![b]\!] : \text{NEG} : \text{BRANCH}(\text{CS}[\![S]\!]) : \text{LOOP}(\text{CB}[\![b]\!]) : \text{NEG}, \text{CS}[\![S]\!], \text{NOOP} \rangle, \varepsilon, s_1 \rangle$

Caso 1: $B[\![b]\!] s_1 == \text{tt}$

$\triangleright \langle \text{NEG} : \text{BRANCH}(\text{CS}[\![S]\!]) : \text{LOOP}(\text{CB}[\![b]\!]) : \text{NEG}, \text{CS}[\![S]\!]), \text{NOOP} \rangle, \text{tt}, s_1 \rangle \triangleright$
 $\langle \text{BRANCH}(\text{CS}[\![S]\!]) : \text{LOOP}(\text{CB}[\![b]\!]) : \text{NEG}, \text{CS}[\![S]\!]), \text{NOOP} \rangle, \text{ff}, s_1 \rangle \triangleright$
 $\langle \text{NOOP}, \varepsilon, s_1 \rangle \triangleright \langle \varepsilon, \varepsilon, s_1 \rangle$

Por tanto por el determinismo de MA $s_1 = s'$.

Por tanto se cumplen las premisas de:

[repeat_{bs} tt] $\langle \text{repeat } S \text{ until } b, s \rangle \rightarrow s' : - \langle S, s \rangle \rightarrow s', B[\![b]\!] s' == \text{true}$

Y concluimos que $\langle \text{repeat } S \text{ until } b, s \rangle \rightarrow s'$

Caso 2: $B[\![b]\!] s_1 == \text{ff}$

$\triangleright \langle \text{NEG} : \text{BRANCH}(\text{CS}[\![S]\!]) : \text{LOOP}(\text{CB}[\![b]\!]) : \text{NEG}, \text{CS}[\![S]\!]), \text{NOOP} \rangle, \text{tt}, s_1 \rangle \triangleright$
 $\langle \text{BRANCH}(\text{CS}[\![S]\!]) : \text{LOOP}(\text{CB}[\![b]\!]) : \text{NEG}, \text{CS}[\![S]\!]), \text{NOOP} \rangle, \text{ff}, s_1 \rangle \triangleright$
 $\langle \text{CS}[\![S]\!] : \text{LOOP}(\text{CB}[\![b]\!]) : \text{NEG}, \text{CS}[\![S]\!], \varepsilon, s_1 \rangle = \langle \text{CS}[\![\text{repeat } S \text{ until } b]\!], \varepsilon, s_1 \rangle \triangleright \langle \varepsilon, \varepsilon, s' \rangle$

Por la h.i. tenemos que $\langle \text{repeat } S \text{ until } b, s_1 \rangle \rightarrow s'$

Por tanto se cumplen las premisas de [repeat_{bs} ff] :

$\langle \text{repeat } S \text{ until } b, s \rangle \rightarrow s' : - \langle S, s \rangle \rightarrow s_1, \langle \text{repeat } S \text{ until } b, s_1 \rangle \rightarrow s', B[\![b]\!] s_1 == \text{false}$

Concluimos que $\langle \text{repeat } S \text{ until } b, s \rangle \rightarrow s'$.

Ejercicio 4.25

Demostrar la corrección del código generado para MA₁.

¿Qué hay que asumir sobre el valor de *env*?

Asumimos que $\text{env}: \text{Var} \rightarrow \mathbb{Z}$ es una función inyectiva.

Solo requerimos modificar en la prueba las partes correspondientes a $\text{CS}[[x := a]]$ y $\text{CA}[[x]]$.

$x := a$

$[\text{ass}_{\text{bs}}] \langle x := a, s \rangle \rightarrow s[x \mapsto A[[a]]s] = s'$

$\langle \text{CS}[[x := a]], \varepsilon, m \rangle = \langle \text{CA}[[a]] : \text{PUT}-(\text{env } x), \varepsilon, m \rangle \triangleright \langle \text{PUT}-(\text{env } x), A[[a]]s, m \rangle \triangleright$

$\langle \varepsilon, \varepsilon, m[\text{env } x \mapsto A[[a]]s] = m' \rangle$

En efecto, si $s = m \circ \text{env}$ y env es inyectivo, entonces $s' = m' \circ \text{env}$, luego $x := a$ es equivalente en ambas semánticas.t

Necesitamos demostrar el lema 8 para MA₁.

$\langle \text{CA}[[x]], \varepsilon, m \rangle = \langle \text{GET}-(\text{env } x), \varepsilon, m \rangle \triangleright \langle \varepsilon, m[\text{env } x], m \rangle$.

Si $s = m \circ \text{env}$, entonces $m[\text{env } x] = s$ $x = A[[x]]s$, luego se cumple el lema 8.

El resto de la prueba funciona sin modificaciones.

Ejercicio 4.26

Suponer que la pila de evaluación solamente puede contener valores enteros.

En consecuencia, tendremos que representar ff y tt mediante 0 y 1.

Introducir todos los cambios necesarios para que la traducción de While siga funcionando correctamente.

Tema 5: Semántica denotacional

Ejercicios recomendados: 5.41, 5.51, 5.54, 5.59 y 5.61

Ejercicio 5.8

Demostrar la caracterización alternativa de \sqsubseteq para State \hookrightarrow State:

$$g_1 \sqsubseteq g_2 \iff \text{grafo}(g_1) \subseteq \text{grafo}(g_2)$$

$$g_1 \sqsubseteq g_2 \Leftrightarrow \forall s, s' \in \text{State } g_1 s = s' \Rightarrow g_2 s = s' \Leftrightarrow$$

$$\Leftrightarrow \forall s, s' \in \text{State } (s, s') \in \text{grafo}(g_1) \Rightarrow (s, s') \in \text{grafo}(g_2) \Leftrightarrow \text{grafo}(g_1) \subseteq \text{grafo}(g_2)$$

Ejercicios 5.18 + 5.19

Considerar $(\mathcal{P}(S), \subseteq)$. Demostrar que todo subconjunto de $\mathcal{P}(S)$ tiene una cota superior mínima. Repetir para $(\mathcal{P}(S), \supseteq)$. ¿Qué ocurre con $(\mathcal{P}_{\text{fin}}(S), \subseteq)$?

Sea $A \in \mathcal{P}(S)$. Consideramos $\cup A \subset S \Rightarrow \cup A \in \mathcal{P}(S)$. Para todo $C \in A$, $C \subseteq \cup A$. Supongamos que $\cup A$ no es cota mínima. Entonces $\exists B \subset \cup A$ cota. Sea $c \in \cup A \setminus B$. Entonces $\exists C \in A$ tal que $c \in C$. Pero entonces $C \not\subseteq B$. Contradicción. Igual con $\cap A$. Con $\mathcal{P}_{\text{fin}}(S)$, la unión de conjuntos finitos no es necesariamente finita, luego existen subconjuntos de $\mathcal{P}_{\text{fin}}(S)$ sin cota superior mínima.

Ejercicio 5.21

Construir un subconjunto de $\text{State} \hookrightarrow \text{State}$ que no tenga cotas superiores.

Consideramos $\{f s = s[x \mapsto 0], g s = s[x \mapsto 1]\}$.

Ejercicio 5.22

Siendo: $g_n s = \begin{cases} s[y \mapsto (s x)!, x \mapsto 1] & \text{si } 0 < s x \wedge s x \leq n \\ \text{INDEFINIDO} & \text{e.c.c.} \end{cases}$

Demostrar que $Y_0 = \{g_n \mid n \geq 0\}$ es una cadena.

Caracterizar las cotas superiores de Y_0 y determinar la mínima de ellas.

Caso 1: $0 < s x < n + 1 \Rightarrow g_n s = g_{n+1} s = s[y \mapsto (s x)!, x \mapsto 1]$

Caso 2: $s x \leq 0$ or $n + 1 < s x \Rightarrow g_n s = g_{n+1} s = \text{UNDEFINED}$

Caso 3: $s x = n + 1 \Rightarrow g_n s = \text{UNDEFINED}, g_{n+1} s = s[y \mapsto (s x)!, x \mapsto 1]$

Concluimos que $g_n \sqsubseteq g_{n+1}$.

$g s = \sqcup Y_0 s = s[y \mapsto (s x)!, x \mapsto 1]$ si $0 < s x$, else UNDEFINED.

Ejercicio 5.36

Demostrar que si f y f' son estrictas, entonces $f' \circ f'$ también es estricta.

$$f \circ f'(\perp) = f(\perp) = \perp$$

Dominio de funciones continuas - Ejercicio 5.41 (Primera parte)

Siendo (D, \sqsubseteq) , (D', \sqsubseteq') ccpo's, definimos $(D \longrightarrow D', \sqsubseteq_F)$, tomando sólo las funciones continuas entre D y D' , y con

$$f_1 \sqsubseteq_F f_2 \stackrel{\text{def}}{=} \forall d \in D \ (f_1 d) \sqsubseteq (f_2 d).$$

Demostrar que $(D \longrightarrow D', \sqsubseteq_F)$ es un ccpo.

¿Qué función es su elemento mínimo?

Teorema 5.37:

- Si $f : D \rightarrow D$ es monótona, existe $\bigcup\{f^n \perp \mid n \geq 0\} \in D$.
- Si $f : D \rightarrow D$ es continua tomaremos

$$\text{FIX } f = \bigcup\{f^n \perp \mid n \geq 0\} \in D$$

lo que está justificado pues $\text{FIX } f$ es el mínimo punto fijo de f :

Punto fijo: $f(\text{FIX } f) = \text{FIX } f$.

Ejercicio 5.40: $f d \sqsubseteq d \implies \text{FIX } f \sqsubseteq d$.

Supongamos que $f d \sqsubseteq d$.

Por definición de \perp , tenemos que $\perp \sqsubseteq d$. Como f es monótona, $f \perp \sqsubseteq f d$. Por nuestra suposición, llegamos a que $f \perp \sqsubseteq f d \sqsubseteq d$. Por inducción, deducimos que $f^n \perp \sqsubseteq d$ para todo n . Por tanto, d es una cota superior de $\{f^n \perp : n > 0\}$.

Como $\text{FIX } f$ es cota superior mínima concluimos que $\text{FIX } f \sqsubseteq d$ ■

Notamos como corolario que $\text{FIX } f$ es el menor punto fijo de f .

$(D \rightarrow D')$ es trivialmente un orden parcial (cumple reflexividad, transitividad y antisimetría). Para ver que además las cadenas tienen cota superior, dada una cadena Y de funciones en $(D \rightarrow D')$, y un elemento d en D , $X = \{f(d) \mid f \text{ en } Y\}$ es una cadena en D' . Para verlo, dados d_1 y d_2 en X , $d_1 = f_1(d)$ y $d_2 = f_2(d)$, y como Y es una cadena, por la definición de \sqsubseteq_F , $d_1 \sqsubseteq_F d_2$ o viceversa; por tanto X tiene cota superior mínima por ser X cadena en D' ccpo, que llamaremos d_x . Ahora, definimos la función g en $(D \rightarrow D')$ como $g(d) = d_x$, y vamos a comprobar que dicha función es cota superior de Y . Dada una función f en Y y un elemento d en D , por construcción, $f(d) \sqsubseteq g(d) = d_x$. Lo que nos da el resultado $f \sqsubseteq_F g$. Además, si hubiese otra cota superior g' de Y . Para cada elemento d en D , se tiene que $f(d) \sqsubseteq g'(d)$ para todo d en D y f en Y , por tanto $g(d) = d_x \sqsubseteq g'(d)$ por ser d_x cota superior mínima, lo que nos da que $g \sqsubseteq_F g'$. Esto nos da el resultado.

El elemento mínimo será $f(d) = \perp$ para todo d en D .

Definición y continuidad del funcional FIX - Ejercicio 5.41 (Segunda parte)

Consideramos el funcional $\text{FIX} : (D \rightarrow D) \rightarrow D$.

Demostar que FIX es monótono y continuo, o sea que

$$\text{FIX}(\bigcup_F \mathcal{F}) = \bigcup\{\text{FIX } f \mid f \in \mathcal{F}\}.$$

para toda cadena $\mathcal{F} \subseteq (D \rightarrow D)$ de funciones continuas.

Funcional cond. Su continuidad (separada)

Fijados $g_0 : \text{State} \hookrightarrow \text{State}$, $p : \text{State} \rightarrow T$, consideramos

$F g = \text{cond}(p, g, g_0)$. $F : (\text{State} \hookrightarrow \text{State}) \rightarrow (\text{State} \hookrightarrow \text{State})$, es continua.

Ejercicio 5.44: Demostrar que $F' g = \text{cond}(p, g_0, g)$ también es continua.

Primero veremos que F' es monótona. Hay que ver que $g_1 \sqsubseteq g_2^* \Rightarrow F' g_1 \sqsubseteq F' g_2$. Dado s , si $p[s] = tt$, entonces $F' g_1 s = g_0 s = F' g_2 s$. Si por otro lado, $p[s] = ff$, entonces $F' g_1 s = g_1 s \sqsubseteq^* g_2 s = F' g_2 s$. Lo que nos da la monotonía de F' .

Dada una cadena Y de f en $(\text{State} \rightarrow \text{State})$, por la monotonía de F' , se tiene que

$\sqcup\{F' f \mid f \text{ en } Y\} \sqsubseteq F'(\sqcup Y)$. Para ver la continuidad de F' basta ver entonces $G = F'(\sqcup Y) \sqsubseteq \sqcup\{F' f \mid f \text{ en } Y\} = H$. Comprobamos que $\text{graph}(G) \subseteq \text{graph}(H)$. Para esto, sea $s \in \text{State}$ tal que $G s = s'$, es decir, $\text{cond}(p, g_0, \sqcup Y)s = s'$. Si $p[s] = tt$, entonces, $\text{cond}(p, g_0, \sqcup Y)s = g_0 s = s'$; como para cada elemento f en Y se tiene que $F' f s = g_0 s = s'$, tenemos lo que buscábamos. Si por el contrario, $p[s] = ff$, $\text{cond}(p, g_0, \sqcup Y)s = \sqcup Y s = s'$. Por tanto, $\langle s, s' \rangle$ está en $\text{graph}(\sqcup Y) = \sqcup\{\text{graph}(f) \mid f \text{ en } Y\}$. Por tanto, tenemos f en Y tal que $\langle s, s' \rangle$ pertenece a $\text{graph}(f)$. Como $p[s] = ff$, $F' f s = f s = s'$, y por tanto $\langle s, s' \rangle$ pertenece a $\text{graph}(H)$, lo que nos da el resultado.

Funcional composición \circ . Su continuidad (separada)

Fijada $g_0 : \text{State} \hookrightarrow \text{State}$, se define $F g = g \circ g_0$.

$F : (\text{State} \hookrightarrow \text{State}) \rightarrow (\text{State} \hookrightarrow \text{State})$, es continua.

Ejercicio 5.46: Demostrar que $F' g = g_0 \circ g$ también es continua.

Ejercicio 5.49

Desarrollar la semántica de la sentencia

$z := 0; \text{while } y \leq x \text{ do } (z := z + 1; x := x - y)$.

$$S_d[S = z:=0 ; \text{while } y \leq x \text{ do } (z:=z+1; x:=x-y)] = S_d[\text{while } y \leq x \text{ do } (z:=z+1; x:=x-y)]$$

$$S_d[z:=0] = \text{FIX } F \circ S_d[z:=0] \text{ donde } F g = \text{cond}(B[y \leq x], g \circ S_d[z:=z+1; x:=x-y], \text{id})$$

$$S_d[z:=z+1; x:=x-y] s = s[z \mapsto s z + 1, x \mapsto s x - s y]$$

$$\text{FIX } F = \sqcup\{F^n \perp\}$$

$$S_d[S] s = s[z \mapsto s x / s y, x \mapsto s x \bmod s y] \text{ si } x, y > 0$$

Ejercicio 5.50

Demostrar que $S_d[\text{while true do skip}] = \perp$, o sea,
es la función completamente indefinida.

$$S_d[\text{while true do skip}] = \text{FIX } F \text{ donde } F g = \text{cond}(B[\text{true}], g \circ S_d[\text{skip}], \text{id}) = g.$$

$$F \perp = \perp, \text{ y como } \perp \text{ es el elemento mínimo entonces } \perp \text{ es el mínimo punto fijo.}$$

Concluimos que $S_d[\text{while true do skip}] = \text{FIX } F = \perp$.

Ejercicio 5.51

Extender el lenguaje **While** con la sentencia **repeat S until b** y dar la cláusula semántica que define S_d para sus aplicaciones.

Demostrar, de manera directa, que S_d sigue estando bien definida tras la extensión.

$S_d[\text{repeat } S \text{ until } b] = \text{FIX } F$ donde $F g = \text{cond}(B[b], \text{id}, g) \circ S_d[S]$.

$F = F_2 \circ F_1$, donde $F_2 g = S_d[S] \circ g$, $F_1 g = \text{cond}(B[b], \text{id}, g)$.

F_2 es continua por la continuidad de la composición (ej 5.46).

F_1 es continua por la continuidad de cond (ej 5.44).

Por tanto F es continua, y $S_d[\text{repeat } S \text{ until } b]$ está bien definido.

Ejercicio 5.53

Demostrar que los siguientes pares de sentencias son equivalentes:

- $S ; \text{skip}$ y S

$S_d[S ; \text{skip}] = S_d[S] \circ S_d[\text{skip}] = S_d[S] \circ \text{id} = S_d[S]$

- $\text{while } b \text{ do } S$ y $\text{if } b \text{ then } (S ; \text{while } b \text{ do } S) \text{ else skip}$

$S_d[\text{while } b \text{ do } S] = \text{FIX } F$ donde $F g = \text{cond}(B[b], g \circ S_d[S], \text{id})$

Por su naturaleza como punto fijo, $\text{FIX } F = F \circ \text{FIX } F$. Por tanto,

$S_d[\text{while } b \text{ do } S] = \text{FIX } F = F \circ \text{FIX } F = \text{cond}(B[b], \text{FIX } F \circ S_d[S], \text{id}) = \text{cond}(B[b], S_d[\text{while } b \text{ do } S] \circ S_d[S], \text{id}) = S_d[\text{if } b \text{ then } (S ; \text{while } b \text{ do } S) \text{ else skip}]$

- $S_1 ; (S_2 ; S_3)$ y $(S_1 ; S_2) ; S_3$

$S_d[S_1 ; (S_2 ; S_3)] = S_d[S_1] \circ S_d[S_2 ; S_3] = S_d[S_1] \circ S_d[S_2] \circ S_d[S_3] = S_d[S_1 ; S_2] \circ S_d[S_3] = S_d[(S_1 ; S_2) ; S_3]$

Ejercicio 5.54

Demostrar que `repeat S until b` es semánticamente equivalente a
`S; while $\neg b$ do S.`

$S_d[[\text{repeat } S \text{ until } b]] = \text{FIX } F$ donde $F g = \text{cond}(B[[b]], \text{id}, g) \circ S_d[[S]]$
 $S_d[[S ; \text{while } \neg b \text{ do } S]] = \text{FIX } F' \circ S_d[[S]]$ donde $F' g = \text{cond}(B[[\neg b]], g \circ S_d[[S]], \text{id}) =$
 $\text{cond}(B[[b]], \text{id}, g \circ S_d[[S]])$

Queremos comprobar que $S_d[[S ; \text{while } \neg b \text{ do } S]]$ es el menor punto fijo de F .

$F S_d[[S ; \text{while } \neg b \text{ do } S]] = F (\text{FIX } F' \circ S_d[[S]]) = \text{cond}(B[[b]], \text{id}, \text{FIX } F' \circ S_d[[S]]) \circ S_d[[S]] =$
 $(F' \text{ FIX } F') \circ S_d[[S]] = \text{FIX } F' \circ S_d[[S]] = S_d[[S ; \text{while } \neg b \text{ do } S]].$

Por tanto $S_d[[S ; \text{while } \neg b \text{ do } S]]$ es un punto fijo de F .

Concluimos que $S_d[[\text{repeat } S \text{ until } b]] \sqsubseteq S_d[[S ; \text{while } \neg b \text{ do } S]].$

Por otra parte, vamos a probar que $F^n \perp \circ S_d[[S]] \sqsubseteq F^n \perp$

Caso base: $n=0$

$\perp \circ S_d[[S]] = \perp$ por la continuidad de la composición.

Caso inductivo:

$F^{n+1} \perp \circ S_d[[S]] = \text{cond}(B[[\neg b]], F^n \perp \circ S_d[[S]], \text{id}) \circ S_d[[S]] \sqsubseteq \text{cond}(B[[b]], \text{id}, F^n \perp) \circ S_d[[S]] = F^{n+1} \perp$

\perp

Por tanto, por la continuidad del supremo tenemos que

$S_d[[S ; \text{while } \neg b \text{ do } S]] = \text{FIX } F' \circ S_d[[S]] = \sqcup\{F^n \perp\} \circ S_d[[S]] = \sqcup\{F^n \perp \circ S_d[[S]]\} \sqsubseteq \sqcup\{F^n \perp\} = \text{FIX } F = S_d[[\text{repeat } S \text{ until } b]]$

Poniendo ambos resultados juntos, concluimos que

$S_d[[S ; \text{while } \neg b \text{ do } S]] = S_d[[\text{repeat } S \text{ until } b]]$

Ejercicio 5.59

Extender la demostración del Teorema 5.55 para incluir la sentencia
`repeat S until b.`

Queremos comprobar que 1) para cada paso de ss , la semántica denotacional asociada no cambia y 2) que $S_{ss} [[S]] \sqsupseteq \Psi(\dots S_{ss} [[S']] \dots)$ para Ψ la definición de S_{sd}

Notamos que `repeat S until b` es denotacionalmente equivalente a `S ; if b then skip else repeat S until b.`

En efecto,

$S_d[[\text{repeat } S \text{ until } b]] = \text{FIX } F$

$S_d[[S ; \text{if } b \text{ then skip else repeat } S \text{ until } b]] = \text{cond}(B[[b]], \text{id}, \text{FIX } F) \circ S_d[[S]] = F \text{ FIX } F = \text{FIX } F$
donde $F g = \text{cond}(B[[b]], \text{id}, g) \circ S_d[[S]]$

Luego $S_d[[\text{repeat } S \text{ until } b]] = S_d[[S ; \text{if } b \text{ then skip else repeat } S \text{ until } b]]$

\Rightarrow

Supongamos que $\langle \text{repeat } S \text{ until } b , s \rangle \Rightarrow^* s'$
 $[repeat_{ss}] \langle \text{repeat } S \text{ until } b , s \rangle \Rightarrow \langle S ; \text{if } b \text{ then skip else repeat } S \text{ until } b , s \rangle$

Según la h.i., $S_d[[S ; \text{if } b \text{ then skip else repeat } S \text{ until } b]] = s'$

Luego $S_d[[\text{repeat } S \text{ until } b]] = S_d[[S ; \text{if } b \text{ then skip else repeat } S \text{ until } b]] = s'$

\Leftarrow

Supongamos que $S_d[[\text{repeat } S \text{ until } b]] = \text{FIX } F = s'$.

Queremos probar que $S_d[[\text{repeat } S \text{ until } b]] = \text{FIX } F \sqsubseteq S_{ss}[[\text{repeat } S \text{ until } b]]$.

Para ello basta con probar que $F S_{ss}[[\text{repeat } S \text{ until } b]] \sqsubseteq S_{ss}[[\text{repeat } S \text{ until } b]]$, por el ejercicio 5.40.

$[repeat_{ss}] \langle \text{repeat } S \text{ until } b , s \rangle \Rightarrow \langle S ; \text{if } b \text{ then skip else repeat } S \text{ until } b , s \rangle$

Como la función comp es estricta,

$S_{ss}[[\text{repeat } S \text{ until } b]] = \text{cond}(B[[b]], \text{id}, S_{ss}[[\text{repeat } S \text{ until } b]]) \circ S_{ss}[[S]]$.

Aplicando la h.i. tenemos que $S_{ss}[[S]] \sqsupseteq S_{sd}[[S]]$

Por la monotonía,

$\text{cond}(B[[b]], \text{id}, S_{ss}[[\text{repeat } S \text{ until } b]]) \circ S_{ss}[[S]] \sqsupseteq \text{cond}(B[[b]], \text{id}, S_{ss}[[\text{repeat } S \text{ until } b]]) \circ$

$S_d[[S]] = F S_{ss}[[\text{repeat } S \text{ until } b]]$

Por tanto, $F S_{ss}[[\text{repeat } S \text{ until } b]] \sqsubseteq S_{ss}[[\text{repeat } S \text{ until } b]]$, luego por las propiedades de FIX,
 $S_d[[\text{repeat } S \text{ until } b]] = \text{FIX } F \sqsubseteq S_{ss}[[\text{repeat } S \text{ until } b]]$.

Concluimos que $S_{ss}[[\text{repeat } S \text{ until } b]] = S_d[[\text{repeat } S \text{ until } b]] = \text{FIX } F = s'$.

Ejercicio 5.61

Demostrar de forma directa que $\forall S \in \text{Stm } S_{bs}[[S]] = S_d[[S]]$.

$$S_{bs}[[S]] = S_{ss}[[S]] = S_{bs}[[S]]$$

Tema 6: Teoría de dominios

Ejercicio 8.6

¿Cómo son las funciones continuas desde un cpo con bottom a un cpo discreto?

Constantes. Sea x elemento del cpo con bottom, y f la función continua. Entonces f es monótona, y como $\perp \sqsubseteq x$ entonces $f \perp \sqsubseteq f x$. Pero en un CPO discreto, eso implica que $f \perp = f x$, luego f es constante.

Tema 7: Más sobre semántica denotacional

Ejercicios recomendados: 6.2, 6.9 y 6.10

Ejercicio 6.2

Demostrar que las dos funciones semánticas S_d y S'_d satisfacen:

$$S_d[S] \circ (\text{lookup } env_V) = (\text{lookup } env_V) \circ (S'_d[S] env_V)$$

para todo entorno env_V que sea inyectivo.

$$\begin{aligned} S_d[x:=a] \circ (\text{lookup } env_V) sto &= [\text{lookup } env_V sto][x \mapsto A[a]] (\text{lookup } env_V sto) = \\ &= [sto \circ env_V][x \mapsto A[a]] (\text{lookup } env_V sto) \\ (\text{lookup } env_V) \circ (S'_d[x:=a] env_V) sto &= (\text{lookup } env_V) (sto[env_V x \mapsto A[a]](\text{lookup } env_V sto)) = \\ &= (sto[env_V x \mapsto A[a]](\text{lookup } env_V sto)) \circ env_V \end{aligned}$$

Por tanto,

$$\begin{aligned} S_d[x:=a] \circ (\text{lookup } env_V) sto x &= [sto \circ env_V][x \mapsto A[a]] (\text{lookup } env_V sto) x = A[a] (\text{lookup } env_V sto) = \\ &= (sto[env_V x \mapsto A[a]](\text{lookup } env_V sto)) \circ env_V x = (\text{lookup } env_V) \circ (S'_d[x:=a] env_V) \\ sto \end{aligned}$$

Asumiendo que env_V es inyectivo:

$$\begin{aligned} (\text{lookup } env_V) \circ (S'_d[x:=a] env_V) sto y &= (sto[env_V \mapsto A[a]](\text{lookup } env_V sto)) \circ env_V y = \text{lookup } env_V sto y = \\ &= (\text{lookup } env_V) (sto[env_V x \mapsto A[a]](\text{lookup } env_V sto)) y = (\text{lookup } env_V) \circ (S'_d[x:=a] env_V) sto y \end{aligned}$$

$$S_d[skip] \circ (\text{lookup } env_V) = \text{lookup } env_V = (\text{lookup } env_V) \circ (S'_d[x:=a] env_V)$$

$$\begin{aligned} (\text{lookup } env_V) \circ (S'_d[S_1 ; S_2] env_V) &= (\text{lookup } env_V) \circ (S'_d[S_2] env_V) \circ (S'_d[S_1] env_V) = (\text{h.i.}) = \\ &= S_d[S_2] \circ (\text{lookup } env_V) \circ (S'_d[S_1] env_V) = S_d[S_2] \circ S_d[S_1] \circ (\text{lookup } env_V) = S_d[S_1 ; \\ &S_2] \circ (\text{lookup } env_V) \end{aligned}$$

$$\begin{aligned} (\text{lookup } env_V) \circ (S'_d[\text{if } b \text{ then } S_1 \text{ else } S_2] env_V) &= \\ &= (\text{lookup } env_V) \circ \text{cond}(B[b] \circ (\text{lookup } env_V), S'_d[S_1] env_V, S'_d[S_2] env_V) = \\ &= \text{cond}(B[b] \circ (\text{lookup } env_V), (\text{lookup } env_V) \circ (S'_d[S_1] env_V), (\text{lookup } env_V) \circ (S'_d[S_2] env_V)) = \\ &\text{h.i.} = \text{cond}(B[b] \circ (\text{lookup } env_V), S_d[S_1] \circ (\text{lookup } env_V), S_d[S_2] \circ (\text{lookup } env_V)) = \\ &= \text{cond}(B[b], S_d[S_1], S_d[S_2]) \circ (\text{lookup } env_V) = S_d[\text{if } b \text{ then } S_1 \text{ else } S_2] \circ (\text{lookup } env_V) \end{aligned}$$

$$(\text{lookup } env_V) \circ (S'_d[\text{while } b \text{ do } S] env_V) = (\text{lookup } env_V) \circ \text{FIX} [\text{cond}(B[b] \circ (\text{lookup } env_V), g \circ S'_d[S] env_V, \text{id})]$$

Definimos $F^n g = \text{cond}(B[b], g \circ S_d[S], \text{id})$. Vamos a probar que $(\text{lookup } env_V) \circ F^n \perp = F^n \perp \circ (\text{lookup } env_V)$ para todo n por inducción.

Para $n=0$, tenemos que $(\text{lookup } env_V) \circ \perp = \perp \circ (\text{lookup } env_V) = \perp$.

Para $n+1$, tenemos que $(\text{lookup } env_V) \circ F^{n+1} \perp = (\text{lookup } env_V) \circ \text{cond}(B[b] \circ (\text{lookup } env_V), F^n \perp \circ S'_d[S] env_V, \text{id}) = \text{cond}(B[b] \circ (\text{lookup } env_V), (\text{lookup } env_V) \circ F^n \perp \circ S'_d[S] env_V, (\text{lookup } env_V)) \triangleq \text{cond}(B[b] \circ (\text{lookup } env_V), F^n \perp \circ (S'_d[S] env_V, (\text{lookup } env_V)))$

$= \text{cond}(B[b]) \circ (\text{lookup } \text{env}_V), F^n \perp \circ S_d[S] \circ (\text{lookup } \text{env}_V), (\text{lookup } \text{env}_V) = \text{cond}(B[b]),$
 $F^n \perp \circ S_d[S], \text{id} \circ (\text{lookup } \text{env}_V) = F^{n+1} \perp \circ (\text{lookup } \text{env}_V)$

Por otra parte, para todo $s \in \text{State}$ tenemos que $\text{FIX } F \text{ env}_V s = F^n \perp \text{ env}_V s$ para cierto n . Por tanto, $(\text{lookup } \text{env}_V) \circ \text{FIX } F \text{ env}_V s = (\text{lookup } \text{env}_V) \circ F^n \perp \text{ env}_V s = F^n \perp \circ (\text{lookup } \text{env}_V) s = \text{FIX } F' \circ (\text{lookup } \text{env}_V) s$.

Por tanto,

$(\text{lookup } \text{env}_V) \circ (S'_d[\text{while } b \text{ do } S] \text{ env}_V) = (\text{lookup } \text{env}_V) \circ \text{FIX } F \text{ env}_V =$
 $\text{FIX } [\text{cond}(B[b]), g \circ S_d[S], \text{id}] \circ (\text{lookup } \text{env}_V) = S_d[\text{while } b \text{ do } S] \circ (\text{lookup } \text{env}_V)$

Ejercicio 6.9

Modificar la sintaxis de la declaración de procedimientos de modo que estos siempre reciban dos parámetros de paso por valor:

$$D_P ::= \text{proc } p(x_1, x_2) \text{ is } S; D_P \mid \epsilon$$

$$S ::= x := a \mid \dots \mid \text{call } p(a_1, a_2)$$

Los entornos de procedimiento serán ahora elementos de:

$$\text{Env}_P = \text{Pname} \longrightarrow ((\mathbb{Z} \times \mathbb{Z}) \longrightarrow (\text{Store} \hookrightarrow \text{Store}))$$

Modificar la semántica convenientemente y dar nuevas cláusulas semánticas para las llamadas a procedimientos (no recursivos y recursivos).

$D_d^P[\text{proc } p(x_1, x_2) \text{ is } S ; D_P] \text{ env}_V \text{ env}_P = D_d^P[D_P] \text{ env}_V (\text{env}_P[p] \mapsto \lambda.a_1a_2. S_d[S] \text{ env}_V[x_1 \mapsto a_1, x_2 \mapsto a_2] \text{ env}_P)$

$D_d^P[\epsilon] = \text{id}$

$S_d[\text{begin } D_V D_P \text{ end}] \text{ env}_V \text{ env}_P \text{ sto} = S_d[S] \text{ env}'_V \text{ env}'_P \text{ sto}'$ donde
 $(\text{env}'_V, \text{sto}') = D_d^V[D_V](\text{env}_V, \text{sto}), \text{env}'_P = D_d^P[D_P] \text{ env}'_V \text{ env}_P$

$S_d[\text{call } p(a_1, a_2)] \text{ env}_V \text{ env}_P = \text{env}_P p A[[a_1]] A[[a_2]]$

$D_d^P[\text{proc } p(x_1, x_2) \text{ is } S ; D_P] \text{ env}_V \text{ env}_P = D_d^P \text{ env}_V (\text{env}_P[p] \mapsto \lambda.a_1a_2 \text{ FIX } F a_1 a_2)$
 donde $F a_1 a_2 g = S_d[S] \text{ env}_V[x_1 \mapsto a_1, x_2 \mapsto a_2] (\text{env}_P[p \mapsto g])$

Ejercicio 6.10

Modificar la semántica denotacional para tener ámbito dinámico para variables y procedimientos.

$D_d^P[\text{proc } p \text{ is } S ; D_P] \text{ env}_P = D_d^P[D_P] (\text{env}_P[p] \mapsto \lambda \text{ env}_V \text{ env}'_P. S_d[S] \text{ env}_V \text{ env}'_P)$
 $D_d^P[\epsilon] = \text{id}$

$S_d[[\text{begin } D_V \text{ } D_P \text{ end}]] \text{ env}_V \text{ env}_P \text{ sto} = S_d[[S]] \text{ env}'_V \text{ env}'_P \text{ sto}'$
donde $(\text{env}'_V, \text{sto}') = D_V^P(D_V)(\text{env}_V, \text{sto})$, $\text{env}'_P = D_P^P(D_P) \text{ env}_P$

$S_d[[\text{call } p(a_1, a_2)]] \text{ env}_V \text{ env}_P = \text{env}_P \text{ p env}_V \text{ env}_P$

$D_d^P[[\text{proc } p \text{ is } S ; D_P]] \text{ env}_P = D_d^P \text{ env}_V (\text{env}_P[p] \mapsto \lambda \text{ env}_V \text{ env}'_P . \text{ FIX } F)$
donde $F g = S_d[[S]] \text{ env}_V (\text{env}'_P[p \mapsto g])$

Tema 8: Análisis de programas

Ejercicios recomendados: 7.9, 7.16, 7.17, 7.23 y 7.24

Ejercicio 7.9

Un análisis de **propagación de constantes** intenta predecir si una expresión (aritmética o booleana) evalúa siempre a un valor constante.

Propiedades para números enteros: $\text{Const} = \mathbb{Z} \cup \{\text{CUALQUIERA}, \text{NINGUNO}\}$, orden \sqsubseteq_C definido por

- $\forall p \in \text{Const} . \text{NINGUNO} \sqsubseteq_C p,$
- $\forall p \in \text{Const} . p \sqsubseteq_C \text{CUALQUIERA}.$

Dibujar el diagrama para $(\text{Const}, \sqsubseteq_C)$.

Especificar el análisis de propagación de constantes definiendo:

- $\mathcal{CA} : Aexp \rightarrow (\text{PState} \rightarrow \text{Const})$
- $\mathcal{CB} : Bexp \rightarrow (\text{PState} \rightarrow \text{TT})$
- $\mathcal{CS} : \text{Stm} \rightarrow (\text{PState} \rightarrow \text{PState})$

$$\text{NINGUNO} \sqsubseteq (\dots, -1, 0, 1, \dots) \sqsubseteq \text{CUALQUIERA}$$

$$\text{CA}[[x]]ps = ps \ x$$

$$\text{CA}[[n]]ps = N[[n]]$$

$$\text{CA}[[a_1 \pm a_2]]ps$$

$$\begin{aligned} | \text{CA}[[a_1]]ps \in \mathbb{Z}, \text{CA}[[a_1]]ps \in \mathbb{Z} &= \text{CA}[[a_1]]ps \pm \text{CA}[[a_2]]ps \\ | \text{otherwise} &= \text{CUALQUIERA} \end{aligned}$$

$$\text{CA}[[a_1 * a_2]]ps$$

$$\begin{aligned} | \text{CA}[[a_1]]ps \in \mathbb{Z} \&\& \text{CA}[[a_2]]ps \in \mathbb{Z} &= \text{CA}[[a_1]]ps * \text{CA}[[a_2]]ps \\ | \text{CA}[[a_1]]ps == 0 \parallel \text{CA}[[a_2]]ps == 0 &= 0 \\ | \text{otherwise} &= \text{CUALQUIERA} \end{aligned}$$

$$\text{CB}[[\text{true}]]ps = \text{TT}$$

$$\text{CB}[[\text{false}]]ps = \text{FF}$$

$$\text{CB}[[a_1 \leq a_2]]ps =$$

$$\begin{aligned} | \text{CA}[[a_1]]ps \in \mathbb{Z} \&\& \text{CA}[[a_2]]ps \in \mathbb{Z} &= \text{CA}[[a_1]]ps \leq \text{CA}[[a_2]]ps \\ | \text{otherwise} &= \text{CUALQUIERA} \end{aligned}$$

$$\text{CB}[[a_1 == a_2]]ps =$$

$$\begin{aligned} | \text{CA}[[a_1]]ps \in \mathbb{Z} \&\& \text{CA}[[a_2]]ps \in \mathbb{Z} &= \text{CA}[[a_1]]ps == \text{CA}[[a_2]]ps \\ | \text{otherwise} &= \text{CUALQUIERA} \end{aligned}$$

$$\text{CB}[[\neg b]]ps =$$

$$\begin{aligned} | \text{CB}[[b]]ps == \text{tt} &= \text{ff} \\ | \text{CB}[[b]]ps == \text{ff} &= \text{tt} \\ | \text{otherwise} &= \text{CUALQUIERA} \end{aligned}$$

$$\text{CB}[[b_1 \wedge b_2]]ps =$$

$| CB[[b_1]]ps == ff \parallel CB[[b_2]]ps == ff = ff$
 $| CB[[b_1]]ps == tt \&& CB[[b_2]]ps == tt = tt$
 $| \text{otherwise} = \text{CUALQUIERA}$

$CS[[x := a]] ps = ps[x \mapsto CS[[a]]ps]$

$CS[[\text{skip}]] = \text{id}$

$CS[[S_1 ; S_2]] = CS[[S_2]] \circ CS[[S_1]]$

$CS[[\text{if } b \text{ then } S_1 \text{ else } S_2]] = \text{cond}_C(CB[[b]], CS[[S_1]], CS[[S_2]])$

Donde

$\text{cond}_C(f, h_1, h_2) ps$

$| f ps == \text{NINGUNO} = \text{INIT}$

$| f ps == tt = h_1 ps$

$| f ps == ff = h_2 ps$

$| f ps == CQ = (h_1 ps) \sqcup (h_2 ps)$

$CS[[\text{while } b \text{ do } S]] = \text{FIX } F \text{ donde } F g = \text{cond}(CB[[b]], g \circ CS[[S]], \text{id})$

Ejercicio 7.16

Extender el lenguaje **While** con la instrucción **repeat S until b** y dar la cláusula que define \mathcal{DS} para ella.

Demostrar que la versión extendida de \mathcal{DS} sigue estando bien definida.

$DS[[\text{repeat } S \text{ until } b]] = \text{FIX } F \text{ donde } F g = \text{cond}_D(DB[[b]], \text{id}, g) \circ DS[[S]]$

Lema 1: La función $H g = \text{cond}_D(b, h_0, g)$ es continua.

En efecto, supongamos que $f \sqsubseteq g$, y sea $ps \in PState$.

Caso 1: $b ps == \text{NINGUNO} \Rightarrow$ Entonces $H f ps = \text{INIT} \sqsubseteq \text{INIT} = H g ps$

Caso 2: $b ps == tt \Rightarrow$ Entonces $H f ps = h_0 ps \sqsubseteq h_0 ps = H g ps$

Caso 3: $b ps == ff \Rightarrow$ Entonces $H f ps = f ps \sqsubseteq g ps = H g ps$

Caso 4: $b ps == CQ \Rightarrow$ Entonces $H f ps = f ps \sqcup h_0 ps \sqsubseteq g ps \sqcup h_0 ps = H g ps$

Luego H es monótona.

Ahora, sea $Y \subseteq (Pstate \rightarrow Pstate)$ una cadena no vacía.

Queremos ver que $\sqcup H Y = H \sqcup Y$. Sea $ps \in PState$.

Caso 1: $b ps == \text{NINGUNO} \Rightarrow$ Entonces $H \sqcup Y ps = \text{INIT} \sqsubseteq \text{INIT} = \sqcup H Y ps$

Caso 2: $b ps == tt \Rightarrow$ Entonces $H \sqcup Y ps = h_0 ps = \sqcup h_0 ps = \sqcup H Y ps$

Caso 3: $b ps == ff \Rightarrow$ Entonces $H \sqcup Y ps = \sqcup Y ps = \sqcup H Y ps$

Caso 4: $b ps == CQ \Rightarrow$ Entonces $H \sqcup Y ps = \sqcup Y ps \sqcup h_0 ps = \sqcup H Y ps$

Luego H es continua.

Lema 2: La función $H g = g \circ h$ es continua.

Por tanto como estamos tomando puntos fijos de funciones continuas, el análisis sigue estando bien definido.

Ejercicio 7.17

Demostrar que el análisis de propagación de constantes especificado en el ejercicio 7.9, está bien definido.

Estamos trabajando sobre un ccpo y los puntos fijos se toman sobre funciones continuas, luego el análisis está bien definido.

Ejercicio 7.18

Demostrar que $\mathcal{A}[[a]] \text{ seguro}_A \mathcal{D}\mathcal{A}[[a]]$, para toda $a \in A\text{exp}$.

Procedemos por inducción estructural.

Sean $s \in \text{State}$ y $ps \in \text{PState}$ tal que $\text{abs } s \sqsubseteq ps$.

$$\text{abs } A[[n]] s = \text{abs } N[[n]] = DA[[n]] ps$$

$$\text{abs } A[[x]] s = \text{abs } s x \sqsubseteq ps x = DA[[n]] ps$$

$$DA[[a_1 + a_2]] ps$$

$$\text{Por h.i., abs } \$ A[[a_i]] s \sqsubseteq DA[[a_i]] ps$$

Caso 1: $DA[[a_1]] ps = \text{NING}$ ó $DA[[a_2]] ps = \text{NING} \Rightarrow \text{Imposible, ya que contradice la h.i.}$

Caso 2: $DA[[a_1]] ps = \text{POS}$ y $DA[[a_2]] ps = \text{POS} \Rightarrow A[[a_i]] s > 0$ por la h.i., luego $\text{abs } A[[a_1 + a_2]] = \text{POS} = DA[[a_1 + a_2]] ps$

Caso 3: $DA[[a_i]] ps = \text{POS}$ y $DA[[a_2]] ps = \text{NEG} \Rightarrow$

....

Expresiones booleanas - Ejercicio 7.19

Partiendo de la semántica $g : \text{State} \rightarrow T$ y el análisis $h : \text{PState} \rightarrow TT$; diremos que h es seguro con respecto a g ($g \text{ seguro}_A h$), si

$$\forall s \in \text{State}, ps \in \text{PState}. \text{abs } s \sqsubseteq_{PS} ps \implies \text{abs}_T(g s) \sqsubseteq_T h ps$$

Demostrar que $\mathcal{B}[[b]] \text{ seguro}_B \mathcal{D}\mathcal{B}[[b]]$, para toda $b \in B\text{exp}$.

Ejercicio 7.24

Demostrar que el análisis de propagación de constantes especificado en el Ejercicio 7.9 es seguro.

Sean $s \in \text{State}$ y $ps \in \text{PState}$ tal que $\text{abs } s \sqsubseteq ps$.

Lema 1: CA es seguro con respecto a A

Lema 2: CB es seguro con respecto a B

Lema 3: CS es seguro con respecto a S

Procedemos por inducción estructural.

$$S[x := a] s = s[x \mapsto A[a] s]$$

$$CS[x := a] ps = ps[x \mapsto CS[a] ps]$$

$$\text{Por tanto, } \text{abs } S[x := a] s x = \text{abs } A[a] s \sqsubseteq CA[a] ps = CS[x := a] ps x$$

$$\text{abs } S[\text{skip}] s = \text{abs id } s = \text{abs } s \sqsubseteq ps = \text{id } ps = CS[\text{skip}] ps$$

Por h.i., tenemos que $\text{abs } S[S_1] s \sqsubseteq CS[S_1] ps$.

Por tanto, de nuevo por h.i., tenemos que $\text{abs } S[S_2] s' \sqsubseteq CS[S_2] ps'$, donde $s' = S[S_1] s$, $ps' = CS[S_1] ps$. Es decir,

$$\text{abs } S[S_1 ; S_2] s = \text{abs } S[S_2] \circ S[S_1] s \sqsubseteq CS[S_2] \circ CS[S_1] ps = CS[S_1 ; S_2] ps$$

Por la h.i., $\text{abs } S[S_i] s \sqsubseteq CS[S_i] ps$. Por el lema 2, $\text{abs } B[b] s \sqsubseteq B[b] ps$.

Caso 1: $B[b] s == tt$

$$\text{abs } S[\text{if } b \text{ then } S_1 \text{ else } S_2] s = \text{abs } S[S_1] s \sqsubseteq CS[S_1] ps \sqsubseteq \text{cond}_C(CB[b], CS[S_1], CS[S_2]) ps = CS[\text{if } b \text{ then } S_1 \text{ else } S_2] ps$$

Caso 2: $B[b] s == ff$

$$\text{abs } S[\text{if } b \text{ then } S_1 \text{ else } S_2] s = \text{abs } S[S_2] s \sqsubseteq CS[S_2] ps \sqsubseteq \text{cond}_C(CB[b], CS[S_1], CS[S_2]) ps = CS[\text{if } b \text{ then } S_1 \text{ else } S_2] ps$$

Para demostrar que $CS[S_2] ps \sqsubseteq \text{cond}_C(CB[b], CS[S_1], CS[S_2]) ps$, nos damos cuenta que ya que $B[b] s == tt \sqsubseteq CB[b] ps$ entonces o bien $CB[b] ps == tt$, en cuyo caso $CS[S_1] ps = \text{cond}_C(CB[b], CS[S_1], CS[S_2]) ps$; o bien $CB[b] ps == \text{CUALQUIERA}$, en cuyo caso $\text{cond}_C(CB[b], CS[S_1], CS[S_2]) ps = (CS[S_1] ps) \sqcup (CS[S_2] ps) \sqsupseteq CS[S_1] ps$.

$$CS[\text{while } b \text{ do } S] = \text{FIX } F \text{ donde } F g = \text{cond}_C(CB[b], g \circ CS[S], \text{id})$$

Procedemos por inducción sobre el número de iteraciones

$$\text{abs } S[\text{while } b \text{ do } S] s = \text{abs cond}(B[b], S[\text{while } b \text{ do } S] \circ S[S], \text{id}) s$$

Caso base: $B[b] s == ff \sqsubseteq ps \Rightarrow ps == ff \text{ ó } ps == \text{CUALQUIERA}$

Entonces $\text{abs } S[\text{while } b \text{ do } S] s = \text{abs } s \sqsubseteq ps$

Por otra parte, $CS[\text{while } b \text{ do } S] ps = \text{cond}_C(CB[b], CS[\text{while } b \text{ do } S] \circ CS[S], \text{id}) ps \sqsupseteq ps$

Por tanto, $\text{abs } S[\text{while } b \text{ do } S] s \sqsubseteq CS[\text{while } b \text{ do } S] ps$

Caso inductivo: $B[b] s == tt \sqsubseteq ps \Rightarrow ps == tt \text{ ó } ps == \text{CUALQUIERA}$

Entonces $\text{abs } S[\text{while } b \text{ do } S] s = \text{abs } S[\text{while } b \text{ do } S] \circ S[S] s$.

Por inducción estructural, $\text{abs } S[S] s \sqsubseteq CS[S] ps$.

Sea $s' = S[S] s$, $ps' = CS[S] ps$, luego $\text{abs } s' \sqsubseteq ps'$

Por inducción sobre el número de iteraciones restantes,

$\text{abs } S[\text{while } b \text{ do } S] s' \sqsubseteq CS[\text{while } b \text{ do } S] ps'$

Es decir, $\text{abs } S[\text{while } b \text{ do } S] \circ S[S] s \sqsubseteq CS[\text{while } b \text{ do } S] \circ CS[S] ps$

$$\sqsubseteq \text{cond}_C(\text{CB}[[b]], \text{CS}[[\text{while } b \text{ do } S]] \circ \text{CS}[[S]], \text{id})ps = \text{CS}[[\text{while } b \text{ do } S]] ps$$

Tema 9: Semántica axiomática

Ejercicios recomendados: 9.11, 9.14, 9.18, 9.24, 9.25, 9.26 y 9.27

Ejercicio 9.10

Expresar una propiedad de corrección parcial para el programa

$z := 0; \text{while } y \leq x \text{ do } (z := z + 1; x := x - y)$

que exprese que el mismo calcula la **división entera** y el **resto** de x entre y .

Construir el árbol de inferencia que demuestre tal propiedad.

$\{x = \text{dividendo}, y = \text{divisor}, x \geq 0, y > 0\}$

$z := 0; \text{while } y \leq x \text{ do } (z := z + 1; x := x - y)$

$\{z = \text{dividendo / divisor}, x = \text{dividendo mod divisor}\}$

[ass_p]

$\{x = \text{dividendo}, y = \text{divisor}\}$

$z := 0$

$\{x = \text{dividendo}, y = \text{divisor}, z=0\}$

[ass_p] + [comp_p]

$\{z^*y + x = \text{dividendo}, y = \text{divisor}, x \text{ mod divisor} = \text{dividendo mod divisor}, 0 < y \leq x\}$

$z := z + 1; x := x - y$

$\{z^*y + x = \text{dividendo}, y = \text{divisor}, x \text{ mod divisor} = \text{dividendo mod divisor}, 0 < y, 0 \leq x\}$

[while_p]

$\{z^*y + x = \text{dividendo}, y = \text{divisor}, x \text{ mod divisor} = \text{dividendo mod divisor}, 0 < y, 0 \leq x\}$

$\text{while } y \leq x \text{ do } (z := z + 1; x := x - y)$

$\{z^*y + x = \text{dividendo}, y = \text{divisor}, x \text{ mod divisor} = \text{dividendo mod divisor}, y > x \geq 0\}$

[cons_p]

$\{x = \text{dividendo}, y = \text{divisor}, z=0, 0 < y, 0 \leq x\}$

$\text{while } y \leq x \text{ do } (z := z + 1; x := x - y)$

$\{z = \text{dividendo / divisor}, x = \text{dividendo mod divisor}\}$

[comp_p]

$\{x = \text{dividendo}, y = \text{divisor}, 0 < y, 0 \leq x\}$

$z := 0; \text{while } y \leq x \text{ do } (z := z + 1; x := x - y)$

$\{z = \text{dividendo / divisor}, x = \text{dividendo mod divisor}\}$

[repeat_p] $\vdash_p \{P\} \text{repeat } S \text{ until } b \{R \wedge B[[b]]\} :- \vdash_p \{P\} S \{R\}, \vdash_p \{R \wedge B[[\neg b]]\} S \{R\}$

Ejercicio 9.15

Demostrar que $\vdash_p \{P\} S \{\text{true}\}$, para cualquier sentencia S y cualquier predicado P .

Procedemos por inducción estructural.

$S = x := a$

[ass_p] {true} $x := a$ {true}

[cons_p] {P} $x := a$ {true}, ya que $P \Rightarrow \text{true}$

$S = \text{skip}$

[skip_p] {true} skip {true}

[cons_p] {P} skip {true}

$S = S_1 ; S_2$

Por h.i., {P} S_1 {true}, {true} S_2 {true}

[comp_p] {P} $S_1 ; S_2$ {true}

$S = \text{if } b \text{ then } S_1 \text{ else } S_2$

Por h.i., {P} $\wedge B[b]$ S_1 {true}, {P} $\wedge B[\neg b]$ S_2 {true}

[if_p] {P} if b then S_1 else S_2 {true}

$S = \text{while } b \text{ do } S$

Por h.i., {true} $\wedge B[b]$ S {true}

[while_p] {true} while b do S {true} $\wedge B[\neg b]$

[cons_p] {P} while b do S {true}

Ejercicio 9.13

- Demostrar que $S ; \text{skip}$ y S son demostrablemente equivalentes.

\Rightarrow

Supongamos que {P} $S ; \text{skip}$ {Q}

Por [comp_p] + [cons_p], $\exists P_1, Q_1, R_1$ tales que $P \Rightarrow P_1, Q_1 \Rightarrow Q$ y

$\vdash_p \{P_1\} S \{R_1\}$

$\vdash_p \{R_1\} \text{skip} \{Q_1\}$

Por [skip_p] + [cons_p], $\exists T_2$ tal que $R_1 \Rightarrow T, T \Rightarrow Q_1$ y

$\vdash_p \{T_2\} \text{skip} \{T_2\}$

Pero entonces $R_1 \Rightarrow Q_1$, luego por [cons_p]

$\vdash_p \{P_1\} S \{Q_1\}$

$\vdash_p \{Q_1\} \text{skip} \{Q_1\}$

Aplicando [cons_p] de nuevo, obtenemos que

$\{P\} S \{Q\}$

\Leftarrow

Supongamos que $\vdash_p \{P\} S \{Q\}$

$[skip_p] \vdash_p \{Q\} skip \{Q\}$

$[comp_p] \vdash_p \{P\} S ; skip \{Q\}$

- Demostrar que $S_1 ; (S_2 ; S_3)$ y $(S_1 ; S_2) ; S_3$ son demostrablemente equivalentes.

\Rightarrow

Supongamos que $\vdash_p \{P\} S_1 ; (S_2 ; S_3) \{Q\}$.

Aplicando $[comp_p] + [cons_p]$ obtenemos que $\exists R_1$ tal que

$\vdash_p \{P\} S_1 \{R_1\}$

$\vdash_p \{R_1\} S_2 ; S_3 \{Q\}$

Aplicando $[comp_p] + [cons_p]$ obtenemos que $\exists R_2$ tal que

$\vdash_p \{R_1\} S_2 \{R_2\}$

$\vdash_p \{R_2\} S_3 \{Q\}$

Ahora aplicamos $[comp_p]$ para obtener:

$\vdash_p \{P\} S_1 ; S_2 \{R_2\}$

Aplicando de nuevo $[comp_p]$ llegamos al resultado deseado:

$\vdash_p \{P\} (S_1 ; S_2) ; S_3 \{Q\}$

\Leftarrow

Análogo al recíproco

Ejercicio 9.14

Demostrar que **repeat S until b** es demostrablemente equivalente a **S; while $\neg b$ do S**.

\Rightarrow

Supongamos que $\vdash_p \{P\} \text{repeat } S \text{ until } b \{Q\}$

Entonces tenemos que por $[\text{repeat}_p] + [\text{cons}_p]$ $\exists P_1, Q_1$ tales que

$P \Rightarrow P_1, Q_1 \wedge B[[b]] \Rightarrow Q$

$\vdash_p \{P_1\} \text{repeat } S \text{ until } b \{Q_1 \wedge B[[b]]\}$

$\vdash_p \{P_1\} S \{Q_1\}$

$\vdash_p \{Q_1 \wedge B[[\neg b]]\} S \{Q_1\}$

$\text{[while}_p\text{]} \vdash_p \{Q_1\} \text{ while } \neg b \text{ do } S \{Q_1 \wedge B[b]\}$
 $\text{[comp}_p\text{]} \vdash_p \{P_1\} S ; \text{ while } \neg b \text{ do } S \{Q_1 \wedge B[b]\}$
 $\text{[cons}_p\text{]} \vdash_p \{P\} S ; \text{ while } \neg b \text{ do } S \{Q\}$

\Leftarrow

Supongamos que $\vdash_p \{P\} S ; \text{ while } \neg b \text{ do } S \{Q\}$
Entonces por $\text{[comp}_p\text{]} + \text{[cons}_p\text{]}$ $\exists P_1, R_1, Q_1$ tales que
 $P \Rightarrow P_1, Q_1 \Rightarrow Q$
 $\vdash_p \{P_1\} S \{R_1\}$
 $\vdash_p \{R_1\} \text{ while } \neg b \text{ do } S \{Q_1\}$

Aplicando $\text{[while}_p\text{]} + \text{[cons}_p\text{]}$ $\exists R_2, Q_2$ tales que
 $R_1 \Rightarrow R_2, Q_2 \wedge B[b] \Rightarrow Q_1$
 $\vdash_p \{R_2\} \text{ while } \neg b \text{ do } S \{Q_2 \wedge B[b]\}$
 $\vdash_p \{R_2 \wedge B[\neg b]\} S \{Q_2\}$

Aplicando $\text{[cons}_p\text{]}$ deducimos que
 $\vdash_p \{P\} S \{R_1\}$
 $\vdash_p \{R_1 \wedge B[\neg b]\} S \{Q_2\}$

Aplicando $\text{[repeat}_p\text{]}$ llegamos a que
 $\vdash_p \{P\} \text{ repeat } S \text{ until } b \{Q_2 \wedge B[b]\}$

Aplicando $\text{[cons}_p\text{]}$ una vez más obtenemos el resultado deseado:
 $\vdash_p \{P\} \text{ repeat } S \text{ until } b \{Q\}$

Ejercicio 9.18

Extender la demostración del Lema 9.17 incluyendo la regla correspondiente a **repeat S until b**.

Procedemos por inducción sobre las reglas de \vdash_p

Supongamos que $\vdash_p \{P\} \text{ repeat } S \text{ until } b \{Q \wedge B[b]\}$ se deriva inmediatamente de $\text{[repeat}_p\text{]}$
Entonces $\vdash_p \{P\} S \{Q\}, \vdash_p \{Q \wedge B[\neg b]\} S \{Q\}$

Consideramos $s \in \text{State}$ tal que $P s == tt$ y que $\langle \text{repeat } S \text{ until } b, s \rangle \rightarrow s'$.
Procedemos por inducción sobre las reglas de b s.

Caso 1: $\langle \text{repeat } S \text{ until } b, s \rangle \rightarrow s'$ se ha derivado mediante $\text{[repeat}_{bs}\text{ tt]}$
En ese caso, tenemos que $\langle S, s \rangle \rightarrow s', B[b] s' == \text{true}$
Aplicando la h.i sobre las reglas de \vdash_p , tenemos que como $\{P\} S \{Q\}, P s == tt$, y $\langle S, s \rangle \rightarrow s'$ entonces $Q s' == tt$. Concluimos que $(Q \wedge B[b])s' == tt$.

Caso 2: $\langle \text{repeat } S \text{ until } b, s \rangle \rightarrow s'$ se ha derivado mediante $\text{[repeat}_{bs}\text{ ff]}$

En ese caso, tenemos que $\exists s_1$ tal que

$\langle S, s \rangle \rightarrow s_1, \langle \text{repeat } S \text{ until } b, s_1 \rangle \rightarrow s', B[b] s_1 == \text{false}$

Por la h.i sobre las reglas de \vdash_p , $Q s_1 == \text{tt}$. Por tanto $(Q \wedge B[\neg b]) s_1 == \text{tt}$.

Ahora aplicamos la h.i. sobre las reglas de bS aplicada a $\langle \text{repeat } S \text{ until } b, s_1 \rangle \rightarrow s'$.

Concluimos que $(Q \wedge B[b]) s' == \text{tt}$.

En conclusión, hemos demostrado que si la precondition P se cumple en s, y la semántica operacional deriva $\langle \text{repeat } S \text{ until } b, s \rangle \rightarrow s'$, entonces s' cumple la postcondición.

En otras palabras, $\models_p \{P\} \text{repeat } S \text{ until } b \{Q \wedge B[b]\}$.

Ejercicio 9.19

Considerar la definición alternativa de validez siguiente:

$$\models' \{P\} S \{Q\} \stackrel{\text{def}}{=} \forall s \in \text{State} [P s = \text{tt} \implies \exists s' (\langle S, s \rangle \rightarrow s' \wedge Q s' = \text{tt})]$$

Demostrar que $\vdash_p \{P\} S \{Q\} \nRightarrow \models' \{P\} S \{Q\}$.

Consideramos $S = \text{while true do skip}$.

Por el ejercicio 9.15, tenemos que $\vdash_p \{\text{true}\} \text{while true do skip} \{\text{true}\}$.

Pero while true do skip cicla, como podemos ver fácilmente por ejemplo comprobando que la cadena de derivación operacional de paso corto entra en un ciclo.

Por tanto, como la semántica operacional de paso largo es equivalente a la semántica operacional de paso corto, no existe s' tal que $\text{while true do skip}, s \rangle \rightarrow s'$, y por tanto $\models_p \{\text{true}\} \text{while true do skip} \{\text{true}\}$.

Postcondición más fuerte - Ejercicio 9.22

Describe lo que cumplen los estados alcanzables desde P:

$$\text{sp}(P, S) s' \iff \exists s \in \text{State} (\langle S, s \rangle \rightarrow s' \wedge P s)$$

Demostrar que para toda instrucción S y todo predicado P :

- $\vdash_p \{P\} S \{\text{sp}(P, S)\}$

Sean $s, s' \in \text{State}$ tales que $P s == \text{tt}$ y $\langle S, s \rangle \rightarrow s'$.

Entonces $s \in \text{State}(\langle S, s \rangle \rightarrow s' \wedge P s)$, luego concluimos que $\text{sp}(P, S) s'$.

Por tanto $\vdash_p \{P\} S \{\text{sp}(P, S)\}$.

- $\vdash_p \{P\} S \{Q\} \implies (\text{sp}(P, S) \Rightarrow Q)$

Supongamos que $\vdash_p \{P\} S \{Q\}$.

Sea s' tal que $\text{sp}(P, S) s' == \text{tt}$. Entonces $\exists s \in \text{State} (\langle S, s \rangle \rightarrow s' \wedge P s)$.

Por tanto, como $\vdash_p \{P\} S \{Q\}$ entonces $Q s' == \text{tt}$.

Concluimos que $\vdash_p \{P\} S \{Q\} \Rightarrow (\text{sp}(P, S) \Rightarrow Q)$.

Ejercicio 9.24

Extender la demostración anterior para incluir la instrucción **repeat S until b**.

Lema 1: Para todo Q, $\vdash_p \{wlp(\text{repeat } S \text{ until } b, Q)\} \text{ repeat } S \text{ until } b \{Q\}$.

$s \in wlp(\text{repeat } S \text{ until } b, Q) \Leftrightarrow (\langle \text{repeat } S \text{ until } b, s \rangle \rightarrow s' \Rightarrow Q s')$

$[\text{repeat}_p] \{P\} \text{ repeat } S \text{ until } b \{R \wedge B[[b]]\} :- \{P\} S \{R\}, \{R \wedge B[[\neg b]]\} S \{R\}$

Por hipótesis de inducción:

$\vdash_p \{wlp(S, wlp(\text{repeat } S \text{ until } b, Q))\} S \{wlp(\text{repeat } S \text{ until } b, Q)\}$

$wlp(S, wlp(\text{repeat } S \text{ until } b, Q)) \Rightarrow wlp(\text{repeat } S \text{ until } b, Q)$

Sea $s \in wlp(S, wlp(\text{repeat } S \text{ until } b, Q))$. Entonces $\langle S, s \rangle \rightarrow s' \Rightarrow s' \in wlp(\text{repeat } S \text{ until } b, Q)$.

Supongamos que $\langle \text{repeat } S \text{ until } b, s \rangle \rightarrow s'$.

Caso 1: $\langle \text{repeat } S \text{ until } b, s \rangle \rightarrow s'$ se ha derivado mediante $[\text{repeat}_{bs} ff]$

Entonces $\langle S, s \rangle \rightarrow s', B[[b]]s' == ff$.

Por tanto $s' \in wlp(\text{repeat } S \text{ until } b, Q)$.

$\vdash_p \{wlp(\text{repeat } S \text{ until } b, Q)\} S \{wlp(\text{repeat } S \text{ until } b, Q)\}$

$wlp(\text{repeat } S \text{ until } b, Q) \wedge B[[\neg b]] \Rightarrow wlp(S, wlp(\text{repeat } S \text{ until } b, Q))$

$\vdash_p \{wlp(\text{repeat } S \text{ until } b, Q) \wedge B[[\neg b]]\} S \{wlp(\text{repeat } S \text{ until } b, Q)\}$

Por tanto podemos aplicar $[\text{repeat}_p]$ para llegar a

$\vdash_p \{wlp(\text{repeat } S \text{ until } b, Q)\} \text{ repeat } S \text{ until } b \{wlp(\text{repeat } S \text{ until } b, Q) \wedge B[[b]]\}$

$wlp(\text{repeat } S \text{ until } b, Q) \wedge B[[b]] \Rightarrow Q$

$\vdash_p \{wlp(\text{repeat } S \text{ until } b, Q)\} \text{ repeat } S \text{ until } b \{Q\}$

Ejercicio 9.25

Demostrar la completitud del sistema de inferencia utilizando las **postcondiciones más fuertes**.

Lema 1: Para todo P y S, $\vdash_p \{P\} S \{\text{sp}(S, P)\}$

$S = x := a$

Lema 2: Para todo P, x, a, existe un Q tal que $P = Q[x \rightarrow A[[a]]]$

Por $[\text{ass}_p]$ tenemos que $\vdash_p \{Q[x \rightarrow A[[a]]]\} S \{Q\}$

???

$\vdash_p \{P\} S \{P\}$

$S = \text{skip}$

Por $[\text{skip}_p]$ tenemos que $\vdash_p \{P\} \text{skip} \{P\}$.

Pero $P \Leftrightarrow \text{sp}(\text{skip}, P)$. En efecto, por $[\text{skip}_p]$ tenemos que $\forall s \in \text{State} \langle \text{skip}, s \rangle \rightarrow s$.

Por tanto $s' \in \text{sp}(\text{skip}, P) \Leftrightarrow \exists s. \langle \text{skip}, s \rangle \rightarrow s' \wedge P s \Leftrightarrow \langle \text{skip}, s' \rangle \rightarrow s' \wedge P s \Leftrightarrow P s$

Donde hemos utilizado el determinismo de la semántica operacional de paso largo para deducir que el único candidato a s es s' .

Concluimos que $\vdash_p \{P\} \text{skip} \{\text{sp}(\text{skip}, P)\}$.

$S = S_1 ; S_2$

Por la h.i., tenemos que $\vdash_p \{P\} S_1 \{\text{sp}(S_1, P)\}, \vdash_p \{\text{sp}(S_1, P)\} S_2 \{\text{sp}(S_2, \text{sp}(S_1, P))\}$

Por tanto, aplicando $[\text{comp}_p]$ obtenemos

(*) $\vdash_p \{P\} S_1 ; S_2 \{\text{sp}(S_2, \text{sp}(S_1, P))\}$.

Por las propiedades de sp , tenemos que

(1) $\vdash_p \{P\} S_1 \{\text{sp}(S_1, P)\}$

(2) $\vdash_p \{\text{sp}(S_1, P)\} S_2 \{\text{sp}(S_2, \text{sp}(S_1, P))\}$

Supongamos que $\langle S_1 ; S_2, s \rangle \rightarrow s'$. Aplicando $[\text{comp}_{bs}]$ obtenemos que $\exists s_1$ tal que $\langle S_1, s \rangle \rightarrow s_1, \langle S_2, s_1 \rangle \rightarrow s'$.

Además supongamos que s cumple que $P s == \text{tt}$. Entonces por (1) tenemos que $\text{sp}(S_1, P)s_1 == \text{tt}$. Ahora aplicando (2) obtenemos que $s' \in \text{sp}(S_2, \text{sp}(S_1, P))$.

Por tanto $\vdash_p \{P\} S_1 ; S_2 \{\text{sp}(S_2, \text{sp}(S_1, P))\}$.

Por las propiedades de sp , deducimos que $\text{sp}(S_1 ; S_2, P) \Rightarrow \text{sp}(S_2, \text{sp}(S_1, P))$

Ahora, supongamos que $s' \in \text{sp}(S_2, \text{sp}(S_1, P))$.

Entonces $\exists s_1$ tal que $\langle S_2, s_1 \rangle \rightarrow s'$ y $\text{sp}(S_1, P)s_1$. Pero eso significa que $\exists s$ tal que $\langle S_1, s \rangle \rightarrow s_1$ y $P s$. Aplicando $[\text{comp}_{bs}]$ deducimos que $\exists s$ tal que $\langle S_1 ; S_2, s \rangle \rightarrow s'$ y $P s$; es decir, $s' \in \text{sp}(S_1 ; S_2, P)$.

Concluimos que $\text{sp}(S_1 ; S_2, P) \Leftrightarrow \text{sp}(S_2, \text{sp}(S_1, P))$.

Por tanto, por (*) tenemos que $\vdash_p \{P\} S_1 ; S_2 \{\text{sp}(S_1 ; S_2, P)\}$.

$S = \text{if } b \text{ then } S_1 \text{ else } S_2$

Por la h.i.,

$\vdash_p \{P \wedge B[b]\} S_1 \{\text{sp}(S_1, P \wedge B[b])\}$

$\vdash_p \{P \wedge B[\neg b]\} S_2 \{\text{sp}(S_2, P \wedge B[\neg b])\}$

Ahora queremos ver que $\text{sp}(S_1, P \wedge B[b]) \Rightarrow \text{sp}(\text{if } b \text{ then } S_1 \text{ else } S_2, P)$

En efecto, sea $s' \in \text{sp}(S_1, P \wedge B[b])$. Entonces por la definición de sp tenemos que $\exists s$ tal que $\langle S_1, s \rangle \rightarrow s'$ y además $s \in P \wedge B[b]$.

Por tanto podemos aplicar $[\text{if}_{bs}^{\text{tt}}]$ para deducir que $\exists s. \langle \text{if } b \text{ then } S_1 \text{ else } S_2, s \rangle \rightarrow s'$ y $P s == \text{tt}$.

Por tanto $s' \in \text{sp}(\text{if } b \text{ then } S_1 \text{ else } S_2, P)$.

Dedujimos que $\text{sp}(S_1, P \wedge B[b]) \Rightarrow \text{sp}(\text{if } b \text{ then } S_1 \text{ else } S_2, P)$

De manera análoga podemos ver que $\text{sp}(S_2, P \wedge B[\neg b]) \Rightarrow \text{sp}(\text{if } b \text{ then } S_1 \text{ else } S_2, P)$.

Por $[cons_p]$, deducimos que

$$\begin{aligned}\vdash_p \{P \wedge B[b]\} S_1 &\{sp(if\ b\ then\ S_1\ else\ S_2,\ P)\} \\ \vdash_p \{P \wedge B[\neg b]\} S_2 &\{sp(if\ b\ then\ S_1\ else\ S_2,\ P)\}\end{aligned}$$

Aplicando $[if_p]$ llegamos a $\vdash_p \{P\} if\ b\ then\ S_1\ else\ S_2\{sp(if\ b\ then\ S_1\ else\ S_2,\ P)\}$

$S = \text{while } b \text{ do } S$

Por la h.i. tenemos que

$$\vdash_p \{wlp(wlp(Q, \text{while } b \text{ do } S), S)\} S \{wlp(Q, \text{while } b \text{ do } S)\}$$

$$wlp(Q, \text{while } b \text{ do } S) \wedge B[b] \Rightarrow wlp(wlp(Q, \text{while } b \text{ do } S), S)$$

En efecto, sea $s \in wlp(Q, \text{while } b \text{ do } S) \wedge B[b]$

$$s \in wlp(wlp(Q, \text{while } b \text{ do } S), S) \Leftrightarrow (\langle S, s \rangle \rightarrow s_1 \Rightarrow s_1 \in wlp(Q, \text{while } b \text{ do } S))$$

Supongamos que $\langle S, s \rangle \rightarrow s_1$.

$$s_1 \in wlp(Q, \text{while } b \text{ do } S) \Leftrightarrow (\langle \text{while } b \text{ do } S, s_1 \rangle \rightarrow s' \Rightarrow s' \in Q)$$

Supongamos que $\langle \text{while } b \text{ do } S, s_1 \rangle \rightarrow s'$.

Entonces como $s \in B[b]$ podemos aplicar $[while_{bs}^{\text{ff}}]$ para deducir que $\langle \text{while } b \text{ do } S, s \rangle \rightarrow s'$

Como $s \in wlp(Q, \text{while } b \text{ do } S)$, entonces deducimos que $s' \in Q$.

Por tanto, $s_1 \in wlp(Q, \text{while } b \text{ do } S)$. Por tanto, $s \in wlp(wlp(Q, \text{while } b \text{ do } S), S)$.

Concluimos que $wlp(Q, \text{while } b \text{ do } S) \wedge B[b] \Rightarrow wlp(wlp(Q, \text{while } b \text{ do } S), S)$.

Aplicando $[cons_p]$ llegamos a

$$\vdash_p \{wlp(Q, \text{while } b \text{ do } S) \wedge B[b]\} S \{wlp(Q, \text{while } b \text{ do } S)\}$$

Aplicando $[while_p]$ llegamos a

$$\vdash_p \{wlp(Q, \text{while } b \text{ do } S)\} \text{while } b \text{ do } S \{wlp(Q, \text{while } b \text{ do } S) \wedge B[\neg b]\}$$

$$wlp(Q, \text{while } b \text{ do } S) \wedge B[\neg b] \Rightarrow Q$$

En efecto, sea $s \in wlp_d(Q, \text{while } b \text{ do } S) \wedge B[\neg b]$.

Entonces usando $[while_{bs}^{\text{ff}}]$ deducimos que $\langle \text{while } b \text{ do } S, s \rangle \rightarrow s$

Como $s \in wlp(Q, \text{while } b \text{ do } S)$ deducimos que $s \in Q$.

Por tanto concluimos que $wlp_d(Q, \text{while } b \text{ do } S) \wedge B[\neg b] \Rightarrow Q$

Aplicando $[cons_p]$ llegamos a

$$\vdash_p \{wlp(Q, \text{while } b \text{ do } S)\} \text{while } b \text{ do } S \{Q\}$$

Queremos demostrar que $\vdash_p \{P\} S \{Q\} \Rightarrow \vdash_p \{P\} S \{Q\}$

Supongamos que $\vdash_p \{P\} S \{Q\}$

Por las propiedades de sp , tenemos que $sp(P, S) \Rightarrow Q$.

Por el lema 1, tenemos que $\vdash_p \{P\} S \{sp(S, P)\}$

Aplicando $[cons_p]$ deducimos por tanto que $\vdash_p \{P\} S \{Q\}$

Ejercicios 9.26 + 9.27

Definir una noción de validez basada en la [semántica denotacional](#).

Demostrar la corrección y completitud del sistema de inferencia respecto de ella.

$$\vdash_d \{P\} S \{Q\} \triangleq \forall s \in P (S_d[S]s = s' \Rightarrow s' \in Q)$$

$$\text{Lema 1: } \vdash_p \{P\} S \{Q\} \Rightarrow \vdash_d \{P\} S \{Q\}$$

Trabajamos por inducción sobre las reglas de p

[ass_p]

Supongamos que $\vdash_p \{P[x \mapsto a]\} x := a \{P\}$

Sabemos que $S_d[x \mapsto a]s = s[x \mapsto A[a]s] = s'$.

Supongamos que $P[x \mapsto a]s = P(s[x \mapsto a]) = P s' == tt$.

Luego $\vdash_d \{P[x \mapsto a]\} x := a \{P\}$

[skip_p]

Supongamos que $\vdash_p \{P\} \text{skip } \{P\}$

Sabemos que $S_d[\text{skip}]s = s = s'$.

Supongamos que $P s == tt$. Entonces $P s' = P s == tt$.

Luego $\vdash_d \{P\} \text{skip } \{P\}$

[comp_p]

Supongamos que $\vdash_p \{P\} S_1 ; S_2 \{R\}$, $\vdash_p \{P\} S_1 \{Q\}$, $\vdash_p \{Q\} S_2 \{R\}$.

Por la h.i tenemos que $\vdash_d \{P\} S_1 \{Q\}$, $\vdash_d \{Q\} S_2 \{R\}$.

Sea $s \in P$, y supongamos que $S_d[S_1 ; S_2]s = S_d[S_2] \circ S_d[S_1] s = s'$.

Como la composición es monótona, tenemos que $\exists s_1 . S_d[S_1] s = s_1$, $S_d[S_2] s_1 = s'$.

Aplicando $\vdash_d \{P\} S_1 \{Q\}$ tenemos que $s_1 \in Q$. Aplicando $\vdash_d \{Q\} S_2 \{R\}$ tenemos que $s' \in R$.

Luego $\vdash_d \{P\} S_1 ; S_2 \{R\}$.

[if_p]

Supongamos que $\vdash_p \{P\} \text{if } b \text{ then } S_1 \text{ else } S_2 \{Q\}$, $\vdash_p \{P \wedge B[b]\} S_1 \{Q\}$, $\vdash_p \{P \wedge B[\neg b]\} S_2 \{Q\}$.

Por la h.i tenemos que $\vdash_d \{P \wedge B[b]\} S_1 \{Q\}$, $\vdash_d \{P \wedge B[\neg b]\} S_2 \{Q\}$.

Sea $s \in P$ tal que $S_d[\text{if } b \text{ then } S_1 \text{ else } S_2]s = \text{cond}(B[b], S_d[S_1], S_d[S_2]) s = s'$.

Caso 1: $B[b]s == tt$

Entonces $s' = S_d[S_1] s$, $s \in P \wedge B[b]$

Aplicando $\vdash_d \{P \wedge B[b]\} S_1 \{Q\}$ tenemos que $s' \in Q$.

Caso 2: $B[b]s == ff$

Entonces $s' = S_d[S_2] s$, $s \in P \wedge B[\neg b]$

Aplicando $\vdash_d \{P \wedge B[\neg b]\} S_2 \{Q\}$ tenemos que $s' \in Q$.

Luego $\vdash_d \{P\} \text{if } b \text{ then } S_1 \text{ else } S_2 \{Q\}$.

[while_p]

Supongamos que $\vdash_p \{P\}$ while b do S $\{P \wedge B[\neg b]\}$, $\vdash_p \{P \wedge B[b]\}$ S $\{P\}$.

Por la h.i tenemos que $\vDash_d \{P \wedge B[b]\}$ S $\{P\}$.

Sea $s \in P$ tal que $S_d[\text{while } b \text{ do } S]s = \text{FIX } F s = s'$.

Vamos a proceder por inducción sobre el mínimo n tal que $\text{FIX } F s = F^n \perp s = s'$.

Claramente $n > 0$, ya que $F^0 \perp = \perp$ no está definido en s.

Caso base: n=1

Entonces $s' = \text{cond}(B[b], \perp \circ S_d[S], \text{id})s$.

Como $\perp \circ S_d[S] = \perp$, debe ser el caso que $B[b]s == ff$, y por tanto $s' = s$.

Entonces $s' = s \in P \wedge B[\neg b]$

Caso inductivo:

Suponemos que si $s_1 \in P$ tal que $S_d[\text{while } b \text{ do } S]s_1 = \text{FIX } F s = F^k \perp s_1 = s_2$ para $k \leq n$ entonces $s_2 \in P \wedge B[\neg b]$.

Supongamos que s es tal que $F^k \perp s$ esta indefinido para $k \leq n$ pero $\text{FIX } F s = F^{n+1} \perp s = \text{cond}(B[b], F^n \perp \circ S_d[S], \text{id})s = s'$.

Claramente $B[b]s == tt$, pues de otra manera $\text{FIX } F s = F \perp s$ en contradicción con nuestra hipótesis.

Por tanto, $s' = \text{FIX } F s = F^{n+1} \perp s = \text{cond}(B[b], F^n \perp \circ S_d[S], \text{id})s = F^n \perp \circ S_d[S]s$.

Si llamamos $s_1 := S_d[S]s$ aplicando $\vDash_d \{P \wedge B[\neg b]\} S \{P\}$ tenemos que $s_1 \in P$.

Por tanto s_1 cumple las hipótesis de inducción, y por tanto $F^n \perp s_1 = s' \in P \wedge B[\neg b]$

Luego $\vDash_d \{P\}$ while b do S $\{P \wedge B[\neg b]\}$.

[cons_p]

Supongamos que $\vdash_p \{P\}$ S $\{Q\}$, $\vdash_p \{P'\}$ S $\{Q'\}$, $P \Rightarrow P'$, $Q \Rightarrow Q'$.

Por la h.i tenemos que $\vDash_d \{P'\}$ S $\{Q'\}$.

Sea $s \in P'$. Entonces como $P \Rightarrow P'$ deducimos que $s \in P$.

Supongamos que $S_d[S]s = s'$. Aplicando $\vDash_d \{P'\}$ S $\{Q'\}$ deducimos que $s' \in Q'$.

Como $Q \Rightarrow Q'$, deducimos que $s' \in Q$.

Luego $\vDash_d \{P\}$ S $\{Q\}$.

Definimos $wlp_d(Q, S) = \{s \in \text{State} : S_d[S]s = s' \Rightarrow s' \in Q\}$

Entonces tenemos que

- $\vDash_d \{wlp_d(Q, S)\} S \{Q\}$
- $\vDash_d \{P\} S \{Q\} \Rightarrow (P \Rightarrow wlp_d(Q, S))$

En efecto, sea $s \in wlp_d(Q, S)$ tal que $S_d[S]s = s'$. Entonces por la definición de wlp_d tenemos que $s' \in Q$. Por tanto $\vDash_d \{wlp_d(Q, S)\} S \{Q\}$.

Ahora supongamos que $\vDash_d \{P\}$ S $\{Q\}$.

Sea $s \in P$. Supongamos que $S_d[S]s = s'$. Entonces por la definición de \vDash_d tenemos que $s' \in Q$. Pero entonces $s \in wlp_d(Q, S)$. Concluimos que $P \Rightarrow wlp_d(Q, S)$.

Lema 2: $\vdash_p \{wlp_d(Q, S)\} S \{Q\}$

$S = x := a$

$$wlp_d(Q, x := a) = \{s \in \text{State} : S_d[[x := a]]s = s[x \mapsto A[[a]]s] = s' \Rightarrow s' \in Q\} = \\ \{s \in \text{State} : s[x \mapsto A[[a]]s] \in Q\} = \{s \in \text{State} : s \in Q[x \mapsto a]\} = Q[x \mapsto a]$$

Por [ass_p] tenemos que $\vdash_p \{Q[x \mapsto a]\} S \{Q\}$.

Luego $\vdash_p \{wlp_d(Q, x := a)\} x := a \{Q\}$

$S = \text{skip}$

$$wlp_d(Q, \text{skip}) = \{s \in \text{State} : S_d[[\text{skip}]]s = s = s' \Rightarrow s' \in Q\} = Q$$

Por [skip_p] tenemos que $\vdash_p \{Q\} \text{skip} \{Q\}$.

Luego $\vdash_p \{wlp_d(Q, \text{skip})\} \text{skip} \{Q\}$

$S = S_1 ; S_2$

Por la h.i tenemos que

$$\vdash_p \{wlp_d(wlp_d(Q, S_2), S_1)\} S_1 \{wlp_d(Q, S_2)\}$$

$$\vdash_p \{wlp_d(Q, S_2)\} S_2 \{Q\}$$

Por [comp_p] tenemos que $\vdash_p \{wlp_d(wlp_d(Q, S_2), S_1)\} S_1 ; S_2 \{Q\}$

Queremos demostrar que $wlp_d(Q, S_1 ; S_2) \Rightarrow wlp_d(wlp_d(Q, S_2), S_1)$.

Sea $s \in wlp_d(Q, S_1 ; S_2)$.

$$s \in wlp_d(wlp_d(Q, S_2), S_1) \Leftrightarrow (S_d[[S_1]]s = s_1 \Rightarrow s_1 \in wlp_d(Q, S_2))$$

Supongamos que $S_d[[S_1]]s = s_1$.

$$s_1 \in wlp_d(Q, S_2) \Leftrightarrow (S_d[[S_2]]s_1 = s' \Rightarrow s' \in Q)$$

Supongamos que $S_d[[S_2]]s_1 = s'$.

Entonces $S_d[[S_1 ; S_2]]s = S_d[[S_2]] \circ S_d[[S_1]] s = s'$. Como $s \in wlp_d(Q, S_1 ; S_2)$, podemos deducir que $s' \in Q$. Por tanto $s_1 \in wlp_d(Q, S_2)$, luego $s \in wlp_d(wlp_d(Q, S_2), S_1)$.

Concluimos que $wlp_d(Q, S_1 ; S_2) \Rightarrow wlp_d(wlp_d(Q, S_2), S_1)$.

Por tanto aplicando [cons_p] llegamos a $\vdash_p \{wlp_d(Q, S_1 ; S_2)\} S_1 ; S_2 \{Q\}$.

$S = \text{if } b \text{ then } S_1 \text{ else } S_2$

Por la h.i tenemos que

$$\vdash_p \{wlp_d(Q, S_1)\} S_1 \{Q\}$$

$$\vdash_p \{wlp_d(Q, S_2)\} S_2 \{Q\}$$

Ahora queremos demostrar que $wlp_d(Q, \text{if } b \text{ then } S_1 \text{ else } S_2) \wedge B[[b]] \Rightarrow wlp_d(Q, S_1)$

En efecto, sea $s \in wlp_d(Q, \text{if } b \text{ then } S_1 \text{ else } S_2) \wedge B[[b]]$

Supongamos que $S_d[[S_1]]s = s'$.

Entonces $S_d[[\text{if } b \text{ then } S_1 \text{ else } S_2]]s = \text{cond}(B[[b]], S_d[[S_1]], S_d[[S_2]])s = S_d[[S_1]]s = s'$.

Como $s \in wlp_d(Q, \text{if } b \text{ then } S_1 \text{ else } S_2)$, deducimos que $s' \in Q$.

Por tanto $s \in wlp_d(Q, S_1)$

Concluimos que $wlp_d(Q, \text{if } b \text{ then } S_1 \text{ else } S_2) \wedge B[[b]] \Rightarrow wlp_d(Q, S_1)$

De manera análoga podemos ver que $wlp_d(Q, \text{if } b \text{ then } S_1 \text{ else } S_2) \wedge B[[\neg b]] \Rightarrow wlp_d(Q, S_2)$.

Aplicando [cons_p] llegamos a que

$$\vdash_p \{wlp_d(Q, \text{if } b \text{ then } S_1 \text{ else } S_2) \wedge B[[b]]\} S_1 \{Q\}$$

$$\vdash_p \{wlp_d(Q, \text{if } b \text{ then } S_1 \text{ else } S_2) \wedge B[[\neg b]]\} S_2 \{Q\}$$

Por tanto aplicando [if_p] llegamos a
 $\vdash_p \{wlp_d(Q, \text{if } b \text{ then } S_1 \text{ else } S_2)\} \text{ if } b \text{ then } S_1 \text{ else } S_2 \{Q\}$.

$S = \text{while } b \text{ do } S$

Por la h.i. tenemos que

$$\vdash_p \{wlp_d(wlp_d(Q, \text{while } b \text{ do } S), S)\} S \{wlp_d(Q, \text{while } b \text{ do } S)\}$$

$$wlp_d(Q, \text{while } b \text{ do } S) \wedge B[[b]] \Rightarrow wlp_d(wlp_d(Q, \text{while } b \text{ do } S), S)$$

En efecto, sea $s \in wlp_d(Q, \text{while } b \text{ do } S) \wedge B[[b]]$

$$s \in wlp_d(wlp_d(Q, \text{while } b \text{ do } S), S) \Leftrightarrow (S_d[[S]]s = s_1 \Rightarrow s_1 \in wlp_d(Q, \text{while } b \text{ do } S))$$

Supongamos que $S_d[[S]]s = s_1$.

$$s_1 \in wlp_d(Q, \text{while } b \text{ do } S) \Leftrightarrow (S_d[[\text{while } b \text{ do } S]]s_1 = s' \Rightarrow s' \in Q)$$

Supongamos que $S_d[[\text{while } b \text{ do } S]]s_1 = s'$.

Entonces $s' = S_d[[\text{while } b \text{ do } S]] \circ S_d[[S]]s = \text{cond}(B[[b]], S_d[[\text{while } b \text{ do } S]] \circ S_d[[S]], \text{id})s = S_d[[\text{while } b \text{ do } S]]s$. (donde hemos usado que $s \in B[[b]]$)

Como $s \in wlp_d(Q, \text{while } b \text{ do } S)$, entonces deducimos que $s' \in Q$.

Por tanto, $s_1 \in wlp_d(Q, \text{while } b \text{ do } S)$. Por tanto, $s \in wlp_d(wlp_d(Q, \text{while } b \text{ do } S), S)$.

Concluimos que $wlp_d(Q, \text{while } b \text{ do } S) \wedge B[[b]] \Rightarrow wlp_d(wlp_d(Q, \text{while } b \text{ do } S), S)$.

Aplicando [cons_p] llegamos a

$$\vdash_p \{wlp_d(Q, \text{while } b \text{ do } S) \wedge B[[b]]\} S \{wlp_d(Q, \text{while } b \text{ do } S)\}$$

Aplicando [while_p] llegamos a

$$\vdash_p \{wlp_d(Q, \text{while } b \text{ do } S)\} \text{ while } b \text{ do } S \{wlp_d(Q, \text{while } b \text{ do } S) \wedge B[[\neg b]]\}$$

$$wlp_d(Q, \text{while } b \text{ do } S) \wedge B[[\neg b]] \Rightarrow Q$$

En efecto, sea $s \in wlp_d(Q, \text{while } b \text{ do } S) \wedge B[[\neg b]]$.

$$\text{Entonces } S_d[[\text{while } b \text{ do } S]]s = \text{cond}(B[[b]], S_d[[\text{while } b \text{ do } S]] \circ S_d[[S]], \text{id})s = s = s'$$

Como $s \in wlp_d(Q, \text{while } b \text{ do } S)$ deducimos que $s' = s \in Q$.

Por tanto concluimos que $wlp_d(Q, \text{while } b \text{ do } S) \wedge B[[\neg b]] \Rightarrow Q$

Aplicando [cons_p] llegamos a

$$\vdash_p \{wlp_d(Q, \text{while } b \text{ do } S)\} \text{ while } b \text{ do } S \{Q\}$$

$$\text{Lema 3: } \vDash_d \{P\} S \{Q\} \Rightarrow \vdash_p \{P\} S \{Q\}$$

Supongamos que $\vDash_d \{P\} S \{Q\}$.

Por las propiedades de wlp_d entonces $P \Rightarrow wlp_d(Q, S)$

Por el lema 2, $\vdash_p \{wlp_d(Q, S)\} S \{Q\}$

Por [cons_p], deducimos que $\vdash_p \{P\} S \{Q\}$

Exámenes

Ejercicio 1 [3,5] Demuestra que $\forall S \in \text{Stm} \forall s, s' \in \text{State}. \langle S, s \rangle \rightarrow s' \iff S_d[S]s = s'$ (sin utilizar los teoremas de equivalencia de la semántica de paso largo con la de paso corto, y de esta última con la denotacional).

Lema 1: $\langle S, s \rangle \rightarrow s' \Rightarrow S_d[S]s = s'$

$S = x := a$

Supongamos que $\langle x := a, s \rangle \rightarrow s'$.

Entonces por $[\text{ass}_{\text{bs}}]$ y el determinismo de bs tenemos que $s' = s[x \mapsto A[[a]]s] = S_d[x := a]s$

$S = \text{skip}$

Supongamos que $\langle \text{skip}, s \rangle \rightarrow s'$.

Entonces por $[\text{skip}_{\text{bs}}]$ y el determinismo de bs tenemos que $s' = s = S_d[\text{skip}]s$

$S = S_1 ; S_2$

Supongamos que $\langle S_1 ; S_2, s \rangle \rightarrow s''$

Entonces por $[\text{comp}_{\text{bs}}]$ y el determinismo de bs tenemos que $\exists s'$ tal que

$\langle S_1, s \rangle \rightarrow s'$

$\langle S_2, s' \rangle \rightarrow s''$

Por la hipótesis de inducción, tenemos que

$S_d[S_1]s = s'$

$S_d[S_2]s' = s''$

Por tanto

$s'' = S_d[S_2] \circ S_d[S_1]s = S_d[S_1 ; S_2]s$

$S = \text{if } b \text{ then } S_1 \text{ else } S_2$

Supongamos que $\langle \text{if } b \text{ then } S_1 \text{ else } S_2, s \rangle \rightarrow s'$.

Caso 1: La proposición anterior se ha derivado mediante $[\text{if}_{\text{bs}}^{\text{tt}}]$

En ese caso, tenemos que

$\langle S_1, s \rangle \rightarrow s'$

$B[[b]]s == \text{tt}$

Por la hi tenemos que

$S_d[S_1]s = s'$

Por tanto,

$S_d[\text{if } b \text{ then } S_1 \text{ else } S_2]s = \text{cond}(B[[b]], S_d[S_1], S_d[S_2])s = S_d[S_1]s = s'$

Caso 2: La proposición anterior se ha derivado mediante $[\text{if}_{\text{bs}}^{\text{ff}}]$

Análogo

$S = \text{while } b \text{ do } S$

Supongamos que $\langle \text{while } b \text{ do } S, s \rangle \rightarrow s'$.

Caso 1: La proposición anterior se ha derivado mediante $[\text{while}_{\text{bs}}^{\text{ff}}]$

Entonces $s' = s$, $B[[b]]s == ff$

Por tanto,

$$S_d[[\text{while } b \text{ do } S]]s = \text{FIX } F s = \text{cond}(B[[b]], \text{FIX } F \circ S_d[[S]], \text{id })s = s$$

Caso 2: La proposición anterior se ha derivado mediante $[\text{while}_{bs}^{tt}]$

En ese caso, tenemos que $\exists s_1$ tal que

$$\langle S, s \rangle \rightarrow s_1$$

$$\langle \text{while } b \text{ do } S, s_1 \rangle \rightarrow s'$$

$$B[[b]]s == tt$$

Por la hipótesis de inducción tenemos que

$$S_d[[S]]s = s_1$$

$$S_d s_1 = s'$$

Por tanto,

$$\begin{aligned} S_d[[\text{while } b \text{ do } S]]s &= \text{FIX } F s = \text{cond}(B[[b]], \text{FIX } F \circ S_d[[S]], \text{id })s = \text{FIX } F \circ S_d[[S]] s = \\ &= S_d[[\text{while } b \text{ do } S]] \circ S_d[[S]]s = S_d[[\text{while } b \text{ do } S]] s_1 = s' \end{aligned}$$

Lema 2: $S_d[[S]]s = s' \Rightarrow \langle S, s \rangle \rightarrow s'$

$$S = x := a$$

$$S_d[[x := a]]s = s[x \mapsto A[[a]]s] = s'$$

Por $[\text{ass}_{bs}]$, tenemos que $\langle x := a, s \rangle \rightarrow s[x \mapsto A[[a]]s]$

$$S = \text{skip}$$

$$S_d[[\text{skip}]]s = s$$

Por $[\text{skip}_{bs}]$, tenemos que $\langle \text{skip}, s \rangle \rightarrow s$

$$S = S_1 ; S_2$$

$$S_d[[S_1 ; S_2]]s = S_d[[S_2]] \circ S_d[[S_1]]s = s'$$

$$\text{Llamemos } s_1 := S_d[[S_1]]s$$

Por inducción estructural,

$$\langle S_1, s \rangle \rightarrow s_1, \langle S_2, s_1 \rangle \rightarrow s'$$

Luego aplicando $[\text{comp}_{bs}]$ tenemos que $\langle S_1 ; S_2, s \rangle \rightarrow s'$

$$S = \text{if } b \text{ then } S_1 \text{ else } S_2$$

$$S_d[[\text{if } b \text{ then } S_1 \text{ else } S_2]]s = \text{cond}(B[[b]], S_d[[S_1]], S_d[[S_2]])s = s'$$

$$\text{Caso 1: } B[[b]]s == tt$$

Entonces $s' = S_d[[S_1]]s$. Por inducción estructural, $\langle S_1, s \rangle \rightarrow s'$.

Luego podemos aplicar $[\text{if}_{bs}^{tt}]$ para deducir que $\langle \text{if } b \text{ then } S_1 \text{ else } S_2, s \rangle \rightarrow s'$

$$\text{Caso 2: } B[[b]]s == ff$$

Entonces $s' = S_d[[S_2]]s$. Por inducción estructural, $\langle S_2, s \rangle \rightarrow s'$.

Luego podemos aplicar $[\text{if}_{bs}^{ff}]$ para deducir que $\langle \text{if } b \text{ then } S_1 \text{ else } S_2, s \rangle \rightarrow s'$

$$S = \text{while } b \text{ do } S$$

Supongamos que $S_d[[\text{while } b \text{ do } S]]s = \text{FIX } F s = s'$

Por la definición de FIX, sabemos que existe un n tal que

$$S_d[[\text{while } b \text{ do } S]]s = \text{FIX } F s = F^n \perp s = s'.$$

Claramente $n > 0$ ya que $F^0 \perp = \perp$ y hemos supuesto que $F^n \perp$ está definida en s .

Vamos a proceder por inducción sobre el menor $n > 0$ tal que $\text{FIX } F s = F^n \perp s$.

Caso base: $n = 1$

$$\text{Entonces } S_d[[\text{while } b \text{ do } S]]s = \text{cond}(B[[b]], \perp, \text{id})s = s'$$

Necesariamente $B[[b]]s == \text{ff}$, o de otra manera s' no estaría bien definido.

Usando $[\text{while}_{bs}^{ff}]$ deducimos que $\langle \text{while } b \text{ do } S, s \rangle \rightarrow s$.

Caso inductivo:

Supongamos la hipótesis cierta para los s_1 tales que $S_d[[\text{while } b \text{ do } S]]s = F^k \perp s$ con $k \leq n$.

Supongamos que s es tal que $n+1$ es el menor número que cumple

$$S_d[[\text{while } b \text{ do } S]]s = F^{n+1} \perp s = \text{cond}(B[[b]], F^n \perp \circ S_d[[S]], \text{id})s = s'.$$

Claramente, $B[[b]]s == \text{tt}$, porque de otra manera $S_d[[\text{while } b \text{ do } S]]s = F \perp$, y $1 < n+1$

$$\text{Entonces } S_d[[\text{while } b \text{ do } S]]s = \text{cond}(B[[b]], \text{FIX } F \circ S_d[[S]], \text{id})s = \text{FIX } F \circ S_d[[S]]s$$

$$S_d[[\text{while } b \text{ do } S]]s = \text{cond}(B[[b]], F^n \perp \circ S_d[[S]], \text{id})s = F^n \perp \circ S_d[[S]]s = s'.$$

Llamemos $s_1 := S_d[[S]]s$. Cómo $F^n \perp s_1 = s'$, tenemos que s_1 cumple las condiciones de la hipótesis de inducción, de lo que podemos deducir que $\langle \text{while } b \text{ do } S, s_1 \rangle \rightarrow s'$

Por otra parte, por la hipótesis de inducción estructural, tenemos que $\langle S, s \rangle \rightarrow s_1$

Por tanto podemos aplicar $[\text{while}_{bs}^{tt}]$ para deducir que $\langle \text{while } b \text{ do } S, s \rangle \rightarrow s$

Ejercicio 2 [3,5] Se desea extender el lenguaje While con *procedimientos que dependan de un parámetro*.

Ahora tendremos que un programa ($P \in \text{Prog}$) consiste en una declaración de procedimientos ($D_P \in \text{Dec}_P$) seguida de una sentencia, que puede incluir llamadas a procedimientos con *paso por valor* del parámetro correspondiente. La sintaxis queda como sigue:

$$\begin{aligned} D_P &::= \text{proc } p(x) \text{ is } S ; D_P | \varepsilon \\ S &::= x := a | \dots | \text{while } b \text{ do } S | \text{call } p(a) \\ P &::= D_P S \end{aligned}$$

Considerando ámbito dinámico para variables y procedimientos, y que se admiten procedimientos recursivos y mútuamente recursivos, hay que resolver los siguientes ejercicios:

1. [1,75] Define una *semántica operacional de paso largo* para los programas del lenguaje extendido. Indica los tipos de datos de las funciones y los entornos que utilices.

En las siguientes reglas,

$$\text{env}_P: \text{Proc} \rightarrow (\text{Statement}, \text{Var})$$

$$\text{upd}_P: D_P \rightarrow \text{env}_P \rightarrow \text{env}_P$$

$$[\text{ass}'_{bs}] \text{ env}_P \vdash \langle x := a, s \rangle \rightarrow s[x \mapsto A[[a]]s]$$

$$[\text{skip}'_{bs}] \text{ env}_P \vdash \langle \text{skip}, s \rangle \rightarrow s$$

$$[\text{comp}'_{bs}] \text{ env}_P \vdash \langle S_1 ; S_2, s \rangle \rightarrow s' : - \text{ env}_P \vdash \langle S_1, s \rangle \rightarrow s', \text{ env}_P \vdash \langle S_2, s' \rangle \rightarrow s''$$

$$[\text{if}'_{bs}^{\text{tt}}] \text{ env}_P \vdash \langle \text{if } b \text{ then } S_1 \text{ else } S_2, s \rangle \rightarrow s' : - \text{ env}_P \vdash \langle S_1, s \rangle \rightarrow s', \quad B[[b]]s == \text{tt}$$

$$[\text{if}'_{bs}^{\text{ff}}] \text{ env}_P \vdash \langle \text{if } b \text{ then } S_1 \text{ else } S_2, s \rangle \rightarrow s' : - \text{ env}_P \vdash \langle S_2, s \rangle \rightarrow s', \quad B[[b]]s == \text{ff}$$

$$[\text{while}'_{bs}^{\text{tt}}]$$

$$[\text{while}'_{bs}^{\text{ff}}]$$

$$[\text{prog}_{bs}] \langle D_P S, s \rangle \rightarrow s' : - \text{ env}_P = \text{upd}_P(D_P, \text{init}_P), \text{ env}_P \vdash \langle S, s \rangle \rightarrow s'$$

$\text{upd}_P(\text{proc } p(x) \text{ is } S ; D_P, \text{env}_P) = \text{upd}_P(D_P, \text{env}_P[p \rightarrow (S, x)])$
 $\text{upd}_P(\epsilon, \text{env}_P) = \text{env}_P$
 $[\text{call}_{bs}^{\text{rec}}] \text{env}_P \vdash \langle \text{call } p(a), s \rangle \rightarrow s'[x \rightarrow s \ x] : - \text{env}_P \vdash \langle S, s[x \rightarrow A[a]]s \rangle \rightarrow s'$
 donde $(S, x) = \text{env}_P p$

2. [1,75] Define una *semántica denotacional* para los programas del lenguaje extendido equivalente a la operacional definida en el apartado anterior. Indica los tipos de datos de las funciones y los entornos que utilices.

Indicación: No son necesarios los entornos de variables; solo entornos de procedimientos.

En las siguientes definiciones,

$\text{env}_P: \text{Proc} \rightarrow Z \rightarrow \text{env}_P \rightarrow (\text{State} \rightarrow \text{State})$

$\text{upd}_P: D_P \rightarrow \text{env}_P \rightarrow \text{env}_P$

$S_d[x:=a] \text{ env}_P s = s[x \rightarrow A[a]]s$

$S_d[\text{skip}] \text{ env}_P s = s$

$S_d[S_1 ; S_2] \text{ env}_P = (S_d[S_2] \text{ env}_P) \circ (S_d[S_1] \text{ env}_P)$

$S_d[\text{if } b \text{ then } S_1 \text{ else } S_2] \text{ env}_P = \text{cond}(B[b], S_d[S_1] \text{ env}_P, S_d[S_2] \text{ env}_P)$

$S_d[\text{while } b \text{ do } S] \text{ env}_P s = \text{FIX } F \text{ env}_P,$

donde $F \text{ env}_P g = \text{cond}(B[b], g \circ S_d[S] \text{ env}_P, \text{id})$

$S_d[D_P S] \text{ s} = S_d[S] \text{ upd}_P(D_P, \text{init}_P) s$

$\text{upd}_P(\text{proc } p(x) \text{ is } S; D_P, \text{env}_P) = \text{upd}_P(D_P, \text{env}_P[p \rightarrow (\lambda \text{arg}. \lambda \text{env}'_P. \lambda s. S_d[S] \text{ env}'_P s[x \rightarrow \text{arg}])])$

$S_d[\text{call } p(a)] \text{ env}_P s = (\text{env}_P p)(A[a]s) \text{ env}_P s$

Ejercicio 3 [3] En la semántica axiomática se define la siguiente regla:

$$[\text{ass}_P] \quad \{ P[x \mapsto A[a]] \} x := a \{ P \}$$

Nos planteamos una definición alternativa

$$[\text{ass2}_P] \quad \{ P \} x := a \{ P \wedge x = A[a] \}$$

Razona formalmente las respuestas a las siguientes preguntas:

1. [1.5] Considerando de forma individual las reglas ass_P y ass2_P . ¿Son equivalentes ámbas reglas? Es decir, ¿de ambas reglas se puede deducir lo mismo?

No, no lo son. Véase el siguiente apartado.

2. [1.5] Demuestra o refuta que existe algún aserto que se deduce por el sistema de reglas inicial que no se puede deducir por el sistema de reglas que surge de sustituir la regla ass por la regla ass2 .

Consideramos $\{x!=1\} x:=1 \{x!=1 \wedge x == 1\}$, que es derivable según $[\text{ass2}_P]$.

Esta afirmación no es válida respecto a la semántica operacional. En efecto, consideramos s tal que $s[x] = 0$. Entonces s satisface la precondición de la regla. Sin embargo, tras la ejecución s queda como $s[x \rightarrow 1]$, que no satisface la postcondición.

Sin embargo, hemos probado que la semántica axiomática con $[\text{ass}_P]$ es válida, luego no puede probar este aserto.

Créditos

Ejercicios 4.23, 5.41, 5.44 realizados por Antonio Valdivia.

El resto de ejercicios han sido resueltos por Jaime Sevilla.

Agradecimientos a David Rubio, Patricia Barrios, Almudena Outeda y Carlos Germes por comentarios y correcciones.

Estos ejercicios se han extraído de las transparencias del curso de TPROG 2017-2018 elaboradas por David de Frutos Escrig (versión original por Yolanda Ortega Mallén).

Ejercicio

Demostrar que $\mathcal{A}[[a[y \mapsto a_0]]]s = \mathcal{A}[[a]](s[y \mapsto \mathcal{A}[[a_0]]s])$ para todo s .