Más sobre semántica operacional. Ejercicios

Ejercicio 3.2

Enunciado

Extender la semántica operacional de paso corto de **WHILE** con la instrucción:

```
assert b before S
```

Demostrar la equivalencia semántica entre `assert true before S` y `S`, pero que `assert false before S` no es equivalente a `skip` ni a `while true do skip`.

Resolución

En primer lugar, la definición de su semántica será:

$$[\text{assert}_{\text{sos}}] := \langle \texttt{assert}\ b\ \texttt{before}\ S, s\rangle \Rightarrow \langle S, s\rangle,\ \text{si}\ \mathcal{B}[\![b]\!] = \texttt{tt}$$

Es decir, solo está definido para el caso en que la condición se cumpla. En cualquier otro, simplemente tenemos *undefined*.

Veamos ahora la equivalencia. Claramente, $\langle \texttt{assert true before}\ S, s\rangle \Rightarrow \langle S, s\rangle$ con lo que es equivalente a S.

Veamos ahora las *no* equivalencias. Por la propia definición, para $\langle \texttt{assert false before}\ S, s\rangle$ no existe ninguna derivación que se le pueda aplicar. Sin embargo, $\langle \texttt{skip}, s\rangle \Rightarrow s$ y $\langle \texttt{while true do}\ S, s\rangle \Rightarrow$
$\langle \texttt{if true then}\ (S\ ;\ \texttt{while true do}\ S\ )\ \texttt{else skip}, s\rangle \Rightarrow$
$\langle S\ ;\ \texttt{while true do}\ S\ s\rangle$ con lo que es un bucle infinito, pero está definido en todo paso.

Ejercicio 3.9

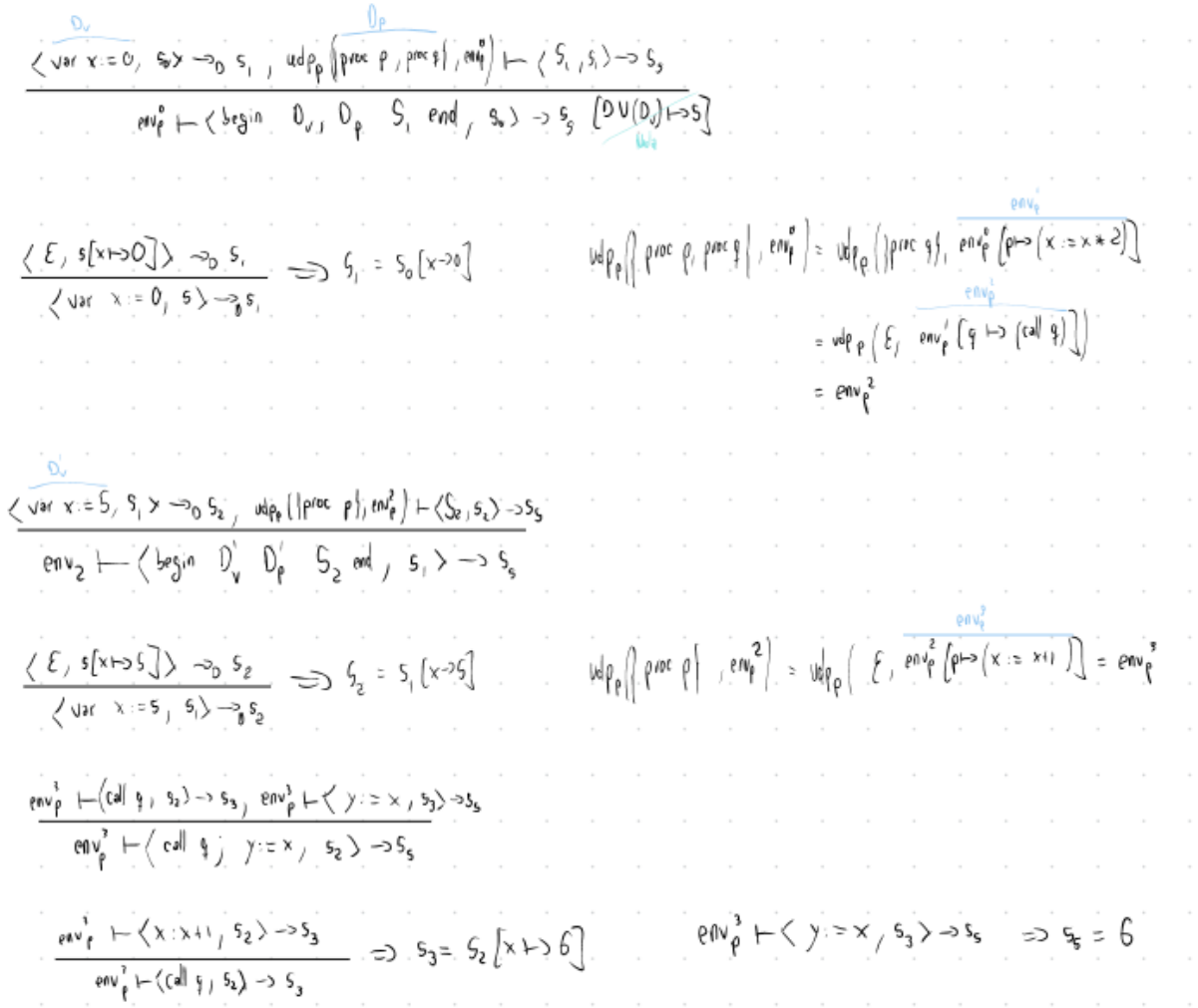
Enunciado

Usar semántica para asegurar que el resultado de y tras ejecutar el siguiente programa es 6 usando entornos dinámicos.

```
begin var x := 0;
      proc p is x := x * 2;
      proc q is call p;
      begin var x := 5;
            proc p is x := x + 1;
            call q;
            y := x;
      end
end
```

Resolución

Simplemente tenemos que ir aplicando las reglas de derivación que correspondan:



Ejercicio 3.14

Enunciado

Demostrar la equivalencia entre la semántica de **While** dada por direcciones de memoria y la dada originalmente con la función estado.

Resolución

En primer lugar, si $s \in \mathbf{State}$ la relación que tendrá con el estado dado por *store*s y entornos de variables vendrá dada por la composición de estos, $s = sto \circ env_V$. Por tanto, lo que tratamos de ver es que $\langle S, s\rangle \to s' \Leftrightarrow env_V, env_P \vdash \langle S, sto\rangle \to sto'$. Las reglas de derivación con el símbolo «$'$» se referirán a las relacionadas con direcciones de memoria. Razonaremos por inducción estructural.

- Asignaciones:

$$[\text{ass}] := \langle x := a, s\rangle \to s\,[x \mapsto \mathcal{A}[\![a]\!]s] = s'$$

$$[\text{ass}'] := env_V, env_P \vdash \langle x := a, sto\rangle \to sto\,[l \mapsto v] = sto'$$

  donde $l = env_V\ x$ y $v = \mathcal{A}[\![a]\!]\underbrace{sto \circ env_V}_{s}$. Por tanto,

$$(sto' \circ env_V)y = \begin{cases} s\ y, & \text{si } y \neq x \\ \mathcal{A}[\![a]\!]s, & \text{si } y = x \end{cases}$$

  que es lo mismo a s', por lo que son equivalentes.

- *Skip*, trivial.

- Composición: Supongamos que $s = sto \circ env_V$. Aplicando $[\text{comp}]$ tenemos que se cumple $\langle S, s\rangle \to s'$ y $\langle S, s'\rangle \to s''$ y, aplicando la hipótesis de inducción, $sto' \circ env_V = s'$ y $sto'' \circ env_V = s''$ y se cumplen $env_V, env_P \vdash \langle S, sto\rangle \to sto''$ y $env_V, env_P \vdash \langle S, sto\rangle \to sto''$. Con esto podemos aplicar $[\text{comp}']$ y tenemos el resultado.

- Condicional: Supongamos que la condición es (o no) cierta y que $s = sto \circ env_V$. Aplicando $\left[\text{if}^{\text{tt}}\right]$ tenemos que $\langle S_1, s\rangle \to s'$ sobre el cuál podemos aplicar la hipótesis de inducción y obtener $env_V, env_P \vdash \langle S_1, sto\rangle \to sto'$ además de $s' = sto' \circ env_V$. Como $s = sto \circ env_V$, es seguro que $\mathcal{B}[\![b]\!] = \mathbf{tt}$ y podemos aplicar $\left[\text{if}^{\text{tt}'}\right]$ con lo que obtenemos el resultado buscado.

- Bucle: Similar al condicional.

Todos los recíprocos salen simplemente al invertir los pasos.

Ejercicio 3.15

Enunciado

Modificar la semántica sobre los procedimientos para que ahora acepten dos parámetros con paso por valor.

Resolución

La mayoría de las reglas se mantienen sin ninguna variación. Tenemos que cambiar:

- Declaración de bloques: Tenemos que cambiar la actualización del entorno de procedimientos:
  - Caso base:

$$\mathrm{udp}_P\left(\varepsilon, env_V, env_P\right) = env_P$$

  - Caso recursivo:

$$\mathrm{udp}_P\left(\texttt{proc}\ p\left(x_1, x_2\right)\ \texttt{is}\ S; D_P, env_V, env_P\right) =$$
$$\mathrm{udp}_P\left(D_P, env_V, env_P\left[p \mapsto \left(x_1, x_2, S, env_V\left[x_1 \mapsto \mathrm{next}, x_2 \mapsto \mathrm{next}\right], env_P\right)\right]\right)$$

- Llamadas:
  - No recursivas:

$$[\mathrm{call}_{\mathrm{ns}}] := \frac{\begin{array}{r} env'_V, env'_P \vdash \langle S, sto[sto \circ env_V\ x_1 \mapsto \mathcal{A}[\![a_1]\!] sto \circ env_V, \\ sto \circ env_V\ x_2 \mapsto \mathcal{A}[\![a_2]\!] sto \circ env_V] \rangle \to sto' \end{array}}{env_V, env_P \vdash \langle \texttt{call}\ p(a_1, a_2), sto \rangle \to sto'}, \\ \text{si } env_P\ p = (a_1, a_2, S, env'_V, env'_P)$$

  - Recursivas: Serán similares a las no recursivas con la excepción de que el entorno de procedimientos de la hipótesis será:

$$env'_P\left[p \mapsto \left(x_1, x_2, S, env'_V, env'_P\right)\right]$$