Tema 3. Más sobre semántica operacional

Más allá del determinismo

Abortar cómputos

- Para definir la semántica de la instrucción `abort` *no la definimos*. De esta forma cuando llegamos a ella tenemos `undefined` y el programa alcanza un estado `stuck` .
- Con esta nueva instrucción encontramos una diferencia entre la semántica operacional de paso corto y paso largo. En el paso corto distinguimos entre bucles infinitos y terminaciones forzosas mientras que en la de paso largo solo importa la terminación correcta. En otras palabras, en paso corto `while true do skip` *no* es equivalente a `abort` mientras que en paso largo *sí*.

Indeterminación

- Añadimos una nueva instrucción `S1 or S2` que ejecuta cualquiera (pero solo una) de las dos sentencias de forma no determinista.
- La definición de paso largo es

$$\left[\mathrm{or}^1_{\mathrm{ns}}\right] := \frac{\langle S_1, s\rangle \to s'}{\langle S_1 \texttt{ or } S_2, s\rangle \to s'} \qquad \left[\mathrm{or}^2_{\mathrm{ns}}\right] := \frac{\langle S_2, s\rangle \to s'}{\langle S_1 \texttt{ or } S_2, s\rangle \to s'}$$

- Y la de paso corto

$$\left[\mathrm{or}^1_{\mathrm{sos}}\right] := \langle S_1 \texttt{ or } S_2, s\rangle \Rightarrow \langle S_1, s\rangle \qquad \left[\mathrm{or}^2_{\mathrm{sos}}\right] := \langle S_1 \texttt{ or } S_2, s\rangle \Rightarrow \langle S_2, s\rangle$$

- Esta instrucción nos aporta una nueva diferencia entre las dos semánticas operacionales: el paso corto *mantiene* los ciclos en presencia de indeterminación mientras que el largo los *elimina*.

Paralelismo

- Extendemos ahora con la instrucción `S1 par S2` que ejecuta paralelamente las dos sentencias, es decir, se intercalan las instrucciones de las dos.
- El paso corto será:

$$\left[\text{par}^1_{\text{sos}}\right] := \frac{\langle S_1, s\rangle \Rightarrow \langle S_1', s'\rangle}{\langle S_1 \ \textbf{par}\ S_2, s\rangle \Rightarrow \langle S_1' \ \textbf{par}\ S_2, s'\rangle}$$
$$\left[\text{par}^2_{\text{sos}}\right] := \frac{\langle S_1, s\rangle \Rightarrow s'}{\langle S_1 \ \textbf{par}\ S_2, s\rangle \Rightarrow \langle S_2, s'\rangle}$$
$$\left[\text{par}^3_{\text{sos}}\right] := \frac{\langle S_2, s\rangle \Rightarrow \langle S_2', s'\rangle}{\langle S_1 \ \textbf{par}\ S_2, s\rangle \Rightarrow \langle S_1 \ \textbf{par}\ S_2', s'\rangle}$$
$$\left[\text{par}^2_{\text{sos}}\right] := \frac{\langle S_1, s\rangle \Rightarrow s'}{\langle S_1 \ \textbf{par}\ S_2, s\rangle \Rightarrow \langle S_1, s'\rangle}$$

- Sin embargo, el paso largo no será capaz de definir esta construcción puesto que solo capta el estado final y no los pasos intermedios, es decir, en el mejor de los casos definirá una construcción en la que una sentencia se ejecuta antes que la otra, pero no más.

Bloques y procedimientos

Bloques

- Extendemos **WHILE** con bloques `begin Dv S end` donde tenemos declaraciones de variables locales: `Dv ::= var x := a; Dv |  `. Debemos añadir que $D_V \in \mathbf{Dec}_V$ que es la categoría sintáctica de *declaraciones de variables*. Con esto definimos el conjunto de variables declaradas en `Dv` :

$$\begin{cases} \text{DV}\left(\textbf{var}\ x := a; D_V\right) & = \{x\} \cup \text{DV}_V \\ \text{DV}\left(\varepsilon\right) & = \emptyset \end{cases}$$

- A la hora de definir la semántica de las declaraciones tenemos que tener en cuenta que *no* son instrucciones. Por esto se tienen que definir de una nueva forma. Además, para recuperar las variables globales que se han redeclarado necesitamos la siguiente función:

$$\left(s'\left[X \mapsto s\right] x\right) := \begin{cases} s\ x, & \text{si } x \in X \\ s'\ x, & \text{si } x \notin X \end{cases}$$

  Intuitivamente, s' es el estado tras abandonar el bloque, s es el de antes de entrar y X es el conjunto de variables globales redeclaradas.

- Definimos ahora la semántica para las declaraciones de variables:

  - Caso base:

$$\left[\text{none}_{\text{ns}}\right] := \langle \varepsilon, s\rangle \to_D s$$

  - Caso recursivo:

$$\left[\text{var}_{\text{ns}}\right] := \frac{\langle D_V, s\left[x \mapsto \mathcal{A}[\![a]\!]s\right]\rangle \to_D s'}{\langle \textbf{var}\ x := a; D_V, s\rangle \to_D s'}$$

- Y para los bloques:

$$[\text{block}_{\text{ns}}] := \frac{\langle D_V, s\rangle \to_D s', \langle S, s'\rangle \to s''}{\langle \texttt{begin } D_V\ S \texttt{ end}, s\rangle \to s''\,[\text{DV}\,(D_V) \mapsto s]}.$$

Es decir, realizamos las declaraciones de las variables locales desde s lo que nos da s', ejecutamos S con s' y obtenemos s'', pero al final tenemos que devolver las variables globales a su estado original con la función auxiliar que hemos visto antes.

Procedimientos

- Presentamos ahora los *procedimientos*, otra nueva construcción que acompaña a los bloques (que no los sustituye, es decir, son los bloques los que generan ámbitos, no los procedimientos).
- Tenemos que cambiar la sintaxis añadiendo una nueva instrucción `call p` y modificando los bloques `begin Dv Dp S end`. Además, tenemos que añadir las declaraciones de los procedimientos: `Dp ::= proc p is S; Dp |  `. Con esto tenemos que $D_P \in \mathbf{Dec}_P$ que es la categoría sintáctica de *declaraciones de procedimientos* y $p \in \mathbf{Pname}$ que es el conjunto de nombres de procedimientos.
- Para definir la semántica extendida con los procedimientos tenemos tres posibilidades distintas

Variables y procedimientos dinámicos

- **Entorno de procedimientos:**
  Función que asocia los nombres de los procedimientos a sus cuerpos:

  $$\mathbf{Env}_P = \mathbf{Pname} \hookrightarrow \mathbf{Stm}$$

  La función no es total para poder hacer llamadas a procedimientos todavía no declarados (por ejemplo, para permitir la recursividad mutua).
- Si ahora tenemos un entorno $env_P \in \mathbf{Env}_P$ tenemos que las transiciones tendrán en cuenta el entorno de la siguiente manera:

  $$env_P \vdash \langle S, s\rangle \to s'$$
- Antes de definir la nueva semántica de largo paso, necesitamos una nueva función auxiliar que actualice $\mathbf{Env}_P$ cuando entramos en un bloque para añadir los nuevos procedimientos que se definan:

  $$\text{upd}_P : \mathbf{Dec}_P \times \mathbf{Env}_P \to \mathbf{Env}_P$$

  Lo hacemos recursivamente:
  - Caso base:

    $$\text{upd}_P\,(\varepsilon, env_P) = env_P$$
  - Caso recursivo:

    $$\text{udp}_P\,(\texttt{proc}\ p\ \texttt{is}\ S; D_P, env_P) = \text{udp}_P\,(D_P, env_P\,[p \mapsto S])$$
- Con todo esto, ya podemos actualizar la semántica de **WHILE**:

- Asignación:

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

- *Skip*:

$$[\text{skip}_{\text{ns}}] := env_P \vdash \langle \texttt{skip}, s\rangle \to s$$

- Composición:

$$[\text{comp}_{\text{ns}}] := \frac{env_P \vdash \langle S_1, s\rangle \to s',\ env_P \vdash \langle S_2, s'\rangle \to s''}{env_P \vdash \langle S_1; S_2, s\rangle \to s''}$$

- Condicional:
  - Si se cumple:

$$\left[\text{if}^{\text{tt}}_{\text{ns}}\right] := \frac{env_P \vdash \langle S_1, s\rangle \to s'}{env_P \vdash \langle \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2, s\rangle \to s'},\ \text{ si } \mathcal{B}[\![b]\!]s = \mathbf{tt}$$

  - Si no se cumple:

$$\left[\text{if}^{\text{ff}}_{\text{ns}}\right] := \frac{env_P \vdash \langle S_2, s\rangle \to s'}{env_P \vdash \langle \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2, s\rangle \to s'},\ \text{ si } \mathcal{B}[\![b]\!]s = \mathbf{ff}$$

- Bucle:
  - Si se cumple:

$$\left[\text{while}^{\text{tt}}_{\text{ns}}\right] := \frac{env_P \vdash \langle S, s\rangle \to s', env_P \vdash \langle \texttt{while}\ b\ \texttt{do}\ S, s'\rangle \to s''}{env_P \vdash \langle \texttt{while}\ b\ \texttt{do}\ S, s\rangle \to s''},\ \text{si } \mathcal{B}[\![b]\!]s = \mathbf{tt}$$

  - Si no se cumple:

$$\left[\text{while}^{\text{ff}}_{\text{ns}}\right] := env_P \vdash \langle \texttt{while}\ b\ \texttt{do}\ S, s\rangle \to s,\ \text{ si } \mathcal{B}[\![b]\!]s = \mathbf{ff}$$

- Bloque:

$$[\text{block}_{\text{ns}}] := \frac{\langle D_V, s\rangle \to_D s',\ \text{udp}_P\,(D_P, env_P) \vdash \langle S, s'\rangle \to s''}{env_P \vdash \langle \texttt{begin}\ D_V\ D_P\ S\ \texttt{end}, s\rangle \to s''\,[\text{DV}\,(D_V) \mapsto s]}$$

- Llamada:

$$[\text{call}^{\text{rec}}_{\text{ns}}] := \frac{env_P \vdash \langle S, s\rangle \to s'}{env_P \vdash \langle \texttt{call}\ p, s\rangle \to s'},\ \text{ si } env_P\ p = S$$

Variables dinámicas y procedimientos estáticos

- **Entorno de procedimientos:**
  Función que asocia los nombres de los procedimientos a sus cuerpos:

$$\mathbf{Env}_P = \mathbf{Pname} \hookrightarrow \mathbf{Stm} \times \mathbf{Env}_P$$

Necesitamos recordar también los procedimientos *ya declarados* en el momento que se declara uno nuevo (puesto que las llamadas son estáticas). Ahora necesitamos una nueva definición para udp_P que será:

$$\begin{cases} \mathrm{udp}_P\left(\varepsilon, env_P\right) & = env_P \\ \mathrm{udp}_P\left(\texttt{proc}\ p\ \texttt{is}\ S; D_P, env_P\right) & = \mathrm{udp}_P\left(D_P, env_P\left[p \mapsto (S, env_P)\right]\right) \end{cases}$$

Lo que quiere decir que, cuando actualizamos el entorno para añadir p, guardamos no solo su cuerpo sino también el entorno *en ese momento*.

- Con esto ya podemos actualizar la definición semántica. En general será igual a la vista en la anterior sección a excepción de las llamadas:

$$[\mathrm{call}_{\mathrm{ns}}] := \frac{env'_P \vdash \langle S, s\rangle \to s'}{env_P \vdash \langle \texttt{call}\ p, s\rangle \to s'}, \ \text{ si } env_P\ p = (S, env'_P)$$

Sin embargo, en este caso las llamadas recursivas no las tenemos directamente y tenemos que añadir una nueva derivación para el caso de que una función se llame a sí misma (observar que cuando actualizamos el entorno, no p no tiene asociado el entorno en el que ha sido declarada sino *el anterior*).

$$[\mathrm{call}^{\mathrm{rec}}_{\mathrm{ns}}] := \frac{env'_P\left[p \mapsto (S, env'_P)\right] \vdash \langle S, s\rangle \to s'}{env_P \vdash \langle \texttt{call}\ p, s\rangle \to s'}, \ \text{ si } env_P\ p = (S, env'_P)$$

En esta nueva definición estamos añadiendo el valor de p al entorno anterior al que se declaró con lo que podemos realizar la llamada recursiva.

Variables y procedimientos estáticos

- Para tener variables estáticas necesitamos modificar el estado y «separarlo» en dos nuevas funciones distintas. Una guardará la *localización* de la variable en un memoria abstracta mientras que la otra *asignará* un valor a esa dirección. Con esto el estado será la composición de estas dos funciones.

- **Funciones $\mathbf{Env}_V$:**
  Esta funciones mapean variables de $\mathbf{Var}$ a posiciones de una memoria abstracta, $\mathbf{Loc}$:

$$\mathbf{Env}_V := \mathbf{Var} \to \mathbf{Loc}$$

- Ahora teniendo un “memoria” $\mathbf{Loc}$, necesitamos una forma de obtener la siguiente dirección libre. Para ello usamos new que es una función que dada una dirección, nos devuelve la siguiente libre:

$$\mathrm{new} : \mathbf{Loc} \to \mathbf{Loc}$$

- **Funciones $\mathbf{Store}$:**
  Estas funciones asignan a las direcciones de $\mathbf{Loc}$ valores que pueden tener las variables (en el caso de **WHILE**, enteros):

$$\mathbf{Store} = \mathbf{Loc} \cup \{\texttt{next}\} \to \mathbb{Z}$$

El elemento **next** representa la siguiente dirección de memoria libre.

- Con esto la semántica de las declaraciones de variables tendrá que ser actualizada de la siguiente forma:

$$\langle D_V, env_V, sto\rangle \to_D (env'_V, sto')$$

- Con todo esto ya sí podemos dar las definiciones de la semántica de las declaraciones de variables. De nuevo lo haremos de forma recursiva:
  - Caso base:

$$[\text{none}_{\text{ns}}] := \langle \varepsilon, env_V, sto\rangle \to_D (env_V, sto)$$

  - Caso recursivo: $[\text{var}_{\text{ns}}] :=$

$$\frac{\langle D_V, env_V\,[x \mapsto l]\,, sto\,[l \mapsto v]\,[\texttt{next} \mapsto \text{new}\ l]\rangle \to_D (env'_V, sto')}{\langle \texttt{var}\ x := a;\ D_V, env_V, sto\rangle \to_D (env'_V, sto')}, \\ \text{si } v = \mathcal{A}[\![a]\!]\,(sto \circ env_V)\ \text{ y } l = sto\,\texttt{next}$$

- De forma intuitiva, la anterior definición nos indica varias cosas:
  - El elemento v nos da el valor de la nueva variable x en el estado anterior a su declaración mientras que l nos indica la posición que ocupará.
  - En la premisa estamos actualizando recursivamente el entorno de las variables añadiendo la posición que ocupará x y el *store* con el valor de x (que se asociará a su posición l) así como con el nuevo **next** que lo aportará la función new, que hemos definido antes.
- Con todas estas definiciones ya estamos en disposición de definir el *entorno de procedimientos* para este caso. Claramente será necesario el entorno de variables. Por tanto:

$$\mathbf{Env}_P = \mathbf{Pname} \hookrightarrow \mathbf{Stm} \times \mathbf{Env}_V \times \mathbf{Env}_P$$

  Y, claro, la actualización tendrá que "recordar" también las variables declaradas en ese momento: $\text{udp}_P : \mathbf{Dec}_p \times \mathbf{Env}_V \times \mathbf{Env}_P \to \mathbf{Env}_P$.

$$\begin{cases} \text{udp}_P\,(\varepsilon, env_V, env_P) & = env_P \\ \text{udp}_P\,(\texttt{proc}\ p\ \texttt{is}\ S; D_P, env_V, env_P) & = \text{udp}_P\,(D_P, env_V, env_P\,[p \mapsto (S, env_V, env_P)]) \end{cases}$$

- De esta forma las transiciones de la semántica de paso largo serán de la siguiente forma:

$$env_V, env_P \vdash \langle S, sto\rangle \to sto'$$

- Con todo esto, ya podemos actualizar la semántica de **WHILE**:
  - Asignación:

$$[\text{ass}_{\text{ns}}] := env_V, env_P \vdash \langle x := a, sto\rangle \to sto\,[l \mapsto v]\,, \\ \text{si } l = env_V\ x \text{ y } v = \mathcal{A}[\![a]\!]\,(sto \circ env_V)$$

  - *Skip*:

$$[\text{skip}_{\text{ns}}] := env_V, env_P \vdash \langle \texttt{skip}, sto\rangle \to sto$$

- Composición:

$$[\text{comp}_{\text{ns}}] := \frac{env_V, env_P \vdash \langle S_1, sto\rangle \to sto',\ env_V, env_P \vdash \langle S_2, sto'\rangle \to sto''}{env_V, env_P \vdash \langle S_1; S_2, sto\rangle \to sto''}$$

- Condicional:
  - Si se cumple:

$$\left[\text{if}^{\text{tt}}_{\text{ns}}\right] := \frac{env_V, env_P \vdash \langle S_1, sto\rangle \to sto'}{env_V, env_P \vdash \langle \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2, sto\rangle \to sto'}, \quad \text{si } \mathcal{B}[\![b]\!]\,(sto \circ env_V) = \texttt{tt}$$

  - Si no se cumple:

$$\left[\text{if}^{\text{ff}}_{\text{ns}}\right] := \frac{env_V, env_P \vdash \langle S_2, sto\rangle \to sto'}{env_V, env_P \vdash \langle \texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2, sto\rangle \to sto'}, \quad \text{si } \mathcal{B}[\![b]\!]\,(sto \circ env_V) = \texttt{ff}$$

- Bucle:
  - Si se cumple:

$$\left[\text{while}^{\text{tt}}_{\text{ns}}\right] := \frac{\begin{array}{c} env_V, env_P \vdash \langle S, sto\rangle \to sto', \\ env_V, env_P \vdash \langle \texttt{while}\ b\ \texttt{do}\ S, sto'\rangle \to sto'' \end{array}}{env_P \vdash \langle \texttt{while}\ b\ \texttt{do}\ S, sto\rangle \to sto''}, \quad \text{si } \mathcal{B}[\![b]\!]\,(sto \circ env_V) = \texttt{tt}$$

  - Si no se cumple:

$$\left[\text{while}^{\text{ff}}_{\text{ns}}\right] := env_V, env_P \vdash \langle \texttt{while}\ b\ \texttt{do}\ S, sto\rangle \to s, \ \text{ si } \mathcal{B}[\![b]\!]\,(sto \circ env_V) = \texttt{ff}$$

- Bloque:

$$[\text{block}_{\text{ns}}] := \frac{\begin{array}{c} \langle D_V, env_V, sto\rangle \to_D (env'_V, sto'), \\ env'_V, env'_P \vdash \langle S, sto'\rangle \to sto'' \end{array}}{env_V, env_P \vdash \langle \texttt{begin}\ D_V\ D_P\ S\ \texttt{end}, sto\rangle \to sto''}, \quad \text{si } env'_P = \text{udp}_P\,(D_P, env'_V, env_P)$$

  Es decir, actualizamos el entorno de las variables para obtener env'_V y sto'. Tras esto, actualizamos el entorno de procedimientos para obtener env'_P. Con todo esto, ya sí ejecutamos S.

- Llamada:
  - No recursiva:

$$[\text{call}_{\text{ns}}] := \frac{env'_V, env_P \vdash \langle S, sto\rangle \to sto'}{env_V, env_P \vdash \langle \texttt{call}\ p, sto\rangle \to sto'}, \ \text{ si } env_P\ p = (S, env'_V, env'_P)$$

  - Recursiva:

$$[\text{call}^{\text{rec}}_{\text{ns}}] := \frac{env'_V, env'_P\,[p \mapsto (S, env'_V, env'_P)] \vdash \langle S, sto\rangle \to sto'}{env_V, env_P \vdash \langle \texttt{call}\ p, sto\rangle \to sto'} \\ \text{si } env_P\ p = (S, env'_V, env'_P),$$