Tema 4. Implementación correcta

La máquina correcta

La máquina abstracta está compuesta de configuraciones que siguen el siguiente patrón:

$$\langle c, e, s\rangle \in \mathbf{Code} \times \mathbf{Stack} \times \mathbf{State}$$

donde c es una serie de instrucciones ($push, add \ldots$), e es la pila de ejecución y s es el estado del programa. La configuración final es $\langle \varepsilon, e, s\rangle$

Las distintas instrucciones alteran la configuración de la máquina de la siguiente manera:

- “Empujar” un número:

$$\langle \texttt{PUSH}{-}n : c, e, s\rangle \triangleright \langle c, \mathcal{N}[\![n]\!] : e, s\rangle$$

- “Empujar” un booleano:

$$\begin{aligned}\langle \texttt{TRUE} : c, e, s\rangle &\triangleright \langle c, \mathbf{tt} : e, s\rangle \\ \langle \texttt{FALSE} : c, e, s\rangle &\triangleright \langle c, \mathbf{ff} : e, s\rangle\end{aligned}$$

- Operaciones con enteros, $z_1, z_2 \in \mathbb{Z}$:

$$\begin{aligned}\langle \texttt{ADD} : c, z_1 : z_2 : e, s\rangle &\triangleright \langle c, (z_1 + z_2) : e, s\rangle \\ \langle \texttt{SUB} : c, z_1 : z_2 : e, s\rangle &\triangleright \langle c, (z_1 - z_2) : e, s\rangle \\ \langle \texttt{MULT} : c, z_1 : z_2 : e, s\rangle &\triangleright \langle c, (z_1 * z_2) : e, s\rangle \\ \langle \texttt{EQ} : c, z_1 : z_2 : e, s\rangle &\triangleright \langle c, (z_1 = z_2) : e, s\rangle \\ \langle \texttt{LE} : c, z_1 : z_2 : e, s\rangle &\triangleright \langle c, (z_1 \leq z_2) : e, s\rangle\end{aligned}$$

- Operaciones con booleanos, $t, t_1, t_2 \in \mathbf{T}$:

$$\langle \texttt{AND} : c, t_1 : t_2 : e, s\rangle \triangleright \begin{cases} \langle c, \mathbf{tt} : e, s\rangle, & \text{si } t_1 = \mathbf{tt} = t_2 \\ \langle c, \mathbf{ff} : e, s\rangle & \text{c.c} \end{cases}$$

$$\langle \texttt{NEG} : c, t : e, s\rangle \triangleright \begin{cases} \langle c, \mathbf{ff} : e, s\rangle & \text{si } t = \mathbf{tt} \\ \langle c, \mathbf{tt} : e, s\rangle & \text{c.c} \end{cases}$$

- Acceso a memoria:

$$\langle \texttt{FETCH}-x : c, e, s\rangle \triangleright \langle c, (s\ x) : e, s\rangle$$
$$\langle \texttt{STORE}-x : c, z : e, s\rangle \triangleright \langle c, e, s\,[x \mapsto z]\rangle$$

- Control de “flujo”:

$$\langle \texttt{NOOP} : c, e, s\rangle \triangleright \langle c, e, s\rangle$$
$$\langle \texttt{BRACH}\,(c_1, c_2) : c, t : e, s\rangle \triangleright \begin{cases} \langle c_1 : c, e, s\rangle & \text{si } t = \mathbf{tt} \\ \langle c_2 : c, e, s\rangle & \text{c.c} \end{cases}$$
$$\langle \texttt{LOOP}\,(c_1, c_2) : c, e, s\rangle \triangleright \langle c_1 : \texttt{BRANCH}\,(c_2 : \texttt{LOOP}\,(c_1, c_2)\,, \texttt{NOOP}) : c, e, s\rangle$$

  Para la última instrucción tenemos que c_1 sería algo así como la “condición” y c_2 el cuerpo del bucle.

Un programa en esta máquina abstracta puede alcanzar dos tipos de configuraciones: con la cola de instrucciones vacía (con lo que la ejecución ha sido exitosa) o con la cola no vacía (con lo que se ha alcanzado una configuración en la que la máquina no ha podido ejecutar la siguiente instrucción).

Propiedades

Esta máquina abstracta se rige, como acabamos de ver, por unas reglas muy similares a las de la semántica operacional de paso corto. Por esta razón, la mayoría de propiedades de esta última tienen equivalencia aquí (Ver ejercicios).

La función *ejecución*

El significado de una secuencia de instrucciones será una función *parcial* de estado a estado definida de la siguiente manera:

$$\mathcal{M} : \mathbf{Code} \to (\mathbf{State} \hookrightarrow \mathbf{State})$$
$$\mathcal{M}[\![c]\!]s = \begin{cases} s', & \text{si } \langle c, \varepsilon, s\rangle \triangleright^* \langle \varepsilon, e, s'\rangle \\ \texttt{undefinied}, & \text{c.c} \end{cases}.$$

Es decir, que un programa tendrá significado si se pueden ejecutar todas sus instrucciones.

Especificación de la traducción

En esta sección daremos la función que permite «compilar» del lenguaje **While** a las instrucciones de esta máquina abstracta.

Expresiones

Usaremos las siguientes funciones totales:

$$\begin{aligned} \mathcal{CA} &: \mathbf{Aexp} \to \mathbf{Code} \\ \mathcal{CB} &: \mathbf{Bexp} \to \mathbf{Code} \end{aligned}$$

que se definen de manera composicional de la forma natural. Solo destacaré el uso de variables:

$$\begin{aligned} \mathcal{CA}[\![n]\!] &= \texttt{PUSH}{-}n \\ \mathcal{CA}[\![x]\!] &= \texttt{FETCH}{-}x \end{aligned}$$

En los operadores, el operador de la derecha será el que está en la cima, seguido del de la izquierda y del operador en sí, en ese orden.

Instrucciones

Usaremos la siguiente función:

$$\mathcal{CS} : \mathbf{Stm} \to \mathbf{Code}$$

definida composicionalmente de la siguiente manera:

$$\begin{aligned} \mathcal{CS}[\![x := a]\!] &= \mathcal{CA}[\![a]\!] : \texttt{STORE}{-}x \\ \mathcal{CS}[\![\texttt{skip}]\!] &= \texttt{NOOP} \\ \mathcal{CS}[\![S_1; S_2]\!] &= \mathcal{CS}[\![S_1]\!] : \mathcal{CS}[\![S_2]\!] \\ \mathcal{CS}[\![\texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2]\!] &= \mathcal{CB}[\![b]\!] : \texttt{BRANCH}\,(\mathcal{CS}[\![S_1]\!], \mathcal{CS}[\![S_2]\!]) \\ \mathcal{CS}[\![\texttt{while}\ b\ \texttt{do}\ S]\!] &= \texttt{LOOP}\,(\mathcal{CB}[\![b]\!], \mathcal{CS}[\![S]\!]) \end{aligned}$$

Función semántica

Con esta función de compilación ya definida, podemos dar el significado de una sentencia S a través de la siguiente función:

$$\mathcal{S}_{\text{am}} : \mathbf{Stm} \to (\mathbf{State} \hookrightarrow \mathbf{State})$$

definida por composición entre la compilación a sentencias de la máquina abstracta y el significado de las instrucciones de esta misma máquina:

$$\mathcal{S}_{\text{am}}[\![S]\!] = (\mathcal{M} \circ \mathcal{CS})\,[\![S]\!]$$

Corrección

En esta sección buscamos demostrar la equivalencia entre la función semántica para las sentencias de **While** que hemos definido en la anterior sección con la semántica operacional que vimos en los anteriores capítulos.

Expresiones

Lema (Corrección expresiones aritméticas)
Para cualquier expresión aritmética a se cumple que

$$\langle \mathcal{CA}[\![a]\!], \varepsilon, s\rangle \rhd^* \langle \varepsilon, \mathcal{A}[\![a]\!]s, s\rangle.$$

Además, todas las configuraciones intermedias tendrán una pila de ejecución no vacía.

Claramente, este lema también se da con las expresiones booleanas.

Instrucciones

Se podría ver la equivalencia con la semántica operacional de paso largo o paso corto. Como en este lenguaje ambas son equivalentes, daría lo mismo. En este caso, siguiendo el libro, lo veremos con el paso largo. La demostración será similar a la de la equivalencia entre las dos semánticas operacionales ya que al definir esta nueva función semántica hemos utilizado un significado muy parecido al paso corto.

Teorema (Equivalencia entre semánticas)
Para toda sentencia S del lenguaje **While**, se da la siguiente igualdad:

$$\mathcal{S}_{\text{ns}}[\![S]\!] = \mathcal{S}_{\text{am}}[\![S]\!]$$