Implementación correcta. Ejercicios

Ejercicio 4.4

Enunciado

Demostrar que si $\langle c_1, e_1, s\rangle \triangleright^k \langle c', e', s'\rangle$, entonces se cumple que $\langle c_1 : c_2, e_1 : e_2, s\rangle \triangleright^k \langle c' : c_2, e' : e_2, s'\rangle$.

Resolución

Razonaremos por inducción sobre el tamaño de k. Para los casos base observamos que ninguna de las instrucciones (por definición) modifica c_2 ni e_2, por lo que se cumple.

Sea ahora una computación de longitud $k+1$ $\left(\langle c_1, e_1, s\rangle \triangleright^{k+1} \langle c', e', s'\rangle\right)$ y supongamos, por hipótesis de inducción, que la propiedad se cumple para las secuencias de longitud k. Si ejecutamos la primera instrucción de c_1 tendremos $\langle c_1 : c_2, e_1 : e_2, s\rangle \triangleright \langle c_1' : c_2, e_1' : e_2, s_1'\rangle$ puesto que aplicamos el caso base. Como ahora tenemos que $\langle c_1', e_1', s\rangle \triangleright^k \langle c', e', s'\rangle$, podemos aplicar la hipótesis de inducción y obtener el resultado que buscábamos.

Ejercicio 4.5

Enunciado

Demostrar que si $\langle c_1 : c_2, e, s\rangle \triangleright^k \langle \varepsilon, e'', s''\rangle$, entonces existe una configuración $\langle \varepsilon, e', s'\rangle$ y $k_1, k_2 \in \mathbb{N}. k_1 + k_2 = k$ tal que:

$$\langle c_1, e, s\rangle \triangleright^{k_1} \langle \varepsilon, e', s'\rangle \wedge \langle c_2, e', s'\rangle \triangleright^{k_2} \langle \varepsilon, e'', s''\rangle$$

Resolución

Razonaremos por inducción sobre la longitud de una computación k. Sea $k = 0$, entonces tenemos directamente el resultado. Supongamos entonces que se cumple el resultado para $k \leq k_0$ y lo demostraremos para $k_0 + 1$. Con esto asumimos que se cumple:

$$\langle c_1 : c_2, e, s\rangle \triangleright^{k_0+1} \langle \varepsilon, e'', s''\rangle$$

o lo que es lo mismo:

$$\langle c_1 : c_2, e, s\rangle \triangleright \langle c_1' : c_2, e_1, s_1\rangle \triangleright^{k_0} \langle \varepsilon, e'', s''\rangle$$

donde c_1' puede ser ε. Esto es así si consideramos que

$$\langle c_1, e, s\rangle \triangleright \langle c_1', e_1, s_1\rangle,$$

aplicamos el anterior ejercicio y tenemos en cuenta el determinismo de este lenguaje.

Si es el caso de que $c_1' = \varepsilon$, entonces ya tenemos el resultado porque $k_1 = 1$ y $k_2 = k_0$ y podemos aplicar la hipótesis de inducción al ser $k_2 \leq k_0$.

Supongamos entonces que $c_1' \neq \varepsilon$. Podemos aplicar la hipótesis de inducción sobre la segunda derivación, de longitud k_0, y obtenemos k_1' y k_2 tales que $k_1' + k_2 = k_0$ cumpliendo que:

$$\langle c_1', e_1, s_1\rangle \triangleright^{k_1'} \langle \varepsilon, e', s'\rangle \wedge \langle c_2, e', s'\rangle \triangleright \langle \varepsilon, e'', s''\rangle$$

Pero entonces con simplemente sumar un paso más a k_1', obtenemos k_1 que cumplirá el resultado que buscábamos:

$$\langle c_1, e, s\rangle \triangleright \langle c_1', e_1, s_1\rangle \triangleright^{k_1'} \langle \varepsilon, e', s'\rangle.$$

Ejercicio 4.6

Enunciado

Demostrar que la semántica dada de la MA es determinista.

$$\gamma \triangleright \gamma' \wedge \gamma \triangleright \gamma'' \Rightarrow \gamma' = \gamma''$$

Resolución

No es posible que γ' sea distinto de γ'' puesto que solo hay una derivación para cada instrucción.

Ejercicio 4.7

Enunciado

Modificar la MA para referirse a las variables por su dirección:

- Las configuraciones serán $\langle c, e, m\rangle$ con $m \in \mathbb{Z}^*$ tal que $m\,[n]$ toma el n-ésimo valor de la lista m.
- Sustituir $\texttt{FETCH}-x$ y $\texttt{STORE}-x$ por $\texttt{GET}-n$ y $\texttt{PUT} - n$ donde $n \in \mathbb{N}$ representa una dirección.

Dar la semántica operacional de esta nueva MA.

Resolución

No daremos todas las instrucciones por ser muchas equivalentes, de hecho, todas las que no utilicen o modifiquen el estado en la anterior MA serán iguales en la nueva (cambiando las s por m simplemente). Por tanto, solo tenemos que modificar las dos que nos indica el enunciado:

$$\langle \texttt{GET}{-}x : c, e, m\rangle \triangleright \langle c, m\left[x\right] : e, m\rangle$$

$$\langle \texttt{STORE}{-}x : c, z : e, m\rangle \triangleright \langle c, e, m'\rangle, \text{ donde } m'\left[n\right] = \begin{cases} m[n], & \text{si } n \neq x \\ z, & \text{si } n = x \end{cases}$$

Ejercicio 4.8

Enunciado

Modificar la MA del anterior ejercicio para que tenga instrucciones de salto no estructuradas:

- Las configuraciones serán $\langle pc, e, m\rangle$ donde $pc \in \mathbb{N}$ indica el contador de programa que apunta a la siguiente instrucción a ejecutar en c ($c\left[pc\right]$).
- Sustituir `BRANCH` y `LOOP` por $\texttt{LABEL}{-}l$, $\texttt{JUMP}{-}l$ y $\texttt{JUMPFALSE} - l$, representado $\texttt{LABEL}{-}l$ con $l \in \mathbb{N}$ una posición en c.

Dar la semántica operacional de esta nueva MA.

Resolución

La mayoría de instrucciones (a parte de las mencionadas en el enunciado) no tendrán un cambio sustancial respecto a la anterior MA. Por ello, solo mostraré, a modo de ejemplo, un par de ellas:

- Si $c\left[pc\right] = \texttt{PUSH}{-}n$:

$$\langle pc, c, e, m\rangle \triangleright \langle pc + 1, c, \mathcal{N}[\![n]\!] : e, m\rangle$$

- Si $c\left[pc\right] = \texttt{GET}{-}n$:

$$\langle pc, c, e, m\rangle \triangleright \langle pc + 1, c, m\left[x\right] : e, m\rangle$$

Y el resto de instrucciones serán igual.

Veamos ahora las instrucciones que nos pide cambiar el enunciado:

- Si $c\left[pc\right] = \texttt{LABEL}{-}l$:

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

Simplemente queremos esta instrucción para que luego cuando la busquemos con el `JUMP` podamos saltar a ella.

- Si $c\,[pc] = \texttt{JUMP}{-}l$:

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

donde $pc' = \min\{pc : c\,[pc] = \texttt{LABEL}{-}l\}$, es decir, saltará a la primera instrucción que sea $\texttt{LABEL}{-}l$.

- Si $c\,[pc] = \texttt{JUMPFALSE}{-}l$ tenemos dos casos:

$$\langle pc, c, \mathbf{ff} : e, m\rangle \triangleright \langle pc', c, e, m\rangle$$
$$\langle pc, c, \mathbf{tt} : e, m\rangle \triangleright \langle pc + 1, c, e, m\rangle$$

donde $pc' = \min\{pc : c\,[pc] = \texttt{LABEL}{-}l\}$. Es decir, que si en la cima de la pila de ejecución hay un símbolo de *falso*, se realiza el salto y, si no es así, se pasa a la siguiente instrucción.

Ejercicio 4.11

Enunciado

Demostrar que $\mathcal{CA}[\![(a_1 + a_2) + a_3]\!] \neq \mathcal{CA}[\![a_1 + (a_2 + a_3)]\!]$, pero que se comportan de forma *suficientemente similar*.

Resolución

Por definición:

$$\begin{aligned}\mathcal{SA}[\![(a_1 + a_2) + a_3]\!] &= \mathcal{SA}[\![a_3]\!] : \mathcal{SA}[\![a_1 + a_2]\!] : \texttt{ADD} \\ &= \mathcal{SA}[\![a_3]\!] : \mathcal{SA}[\![a_2]\!] : \mathcal{SA}[\![a_1]\!] : \texttt{ADD} : \texttt{ADD}\end{aligned}$$

Mientras que:

$$\begin{aligned}\mathcal{SA}[\![a_1 + (a_2 + a_3)]\!] &= \mathcal{SA}[\![a_2 + a_3]\!] : \mathcal{SA}[\![a_1]\!] : \texttt{ADD} \\ &= \mathcal{SA}[\![a_3]\!] : \mathcal{SA}[\![a_2]\!] : \texttt{ADD} : \mathcal{SA}[\![a_1]\!] : \texttt{ADD}\end{aligned}$$

con lo que estrictamente no son iguales. Sin embargo, son *suficientemente similares* puesto que la suma es asociativa y el orden en que se hagan las operaciones no importa.

Ejercicio 4.14

Enunciado

Extender **WHILE** con la instrucción `repeat S until b` y dar la compilación de esta a AM.

Resolución

La compilación será:

$$\mathcal{CS}[\![\texttt{repeat}\ S\ \texttt{until}\ b]\!] = \mathcal{CS}[\![S]\!] : \texttt{LOOP}\left(\mathcal{CB}[\![\neg b]\!], \mathcal{CS}[\![S]\!]\right).$$

Ejercicio 4.16

Enunciado

Dar la función de compilación para las instrucciones de **WHILE** a la MA del ejercicio 4.7. Utilizar la función $env : \mathbf{Var} \to \mathbb{N}$ que, dada una variable, devuelve su dirección de memoria.

Resolución

Tan solo tendremos que cambiar las asignaciones y el *fetch* de las variables:

$$\begin{aligned}\mathcal{CA}[\![x]\!] &= \texttt{GET}-(env\ x)\\ \mathcal{CA}[\![x := a]\!] &= \mathcal{CA}[\![a]\!] : \texttt{PUT}-(env\ x)\end{aligned}$$

El resto de traducciones se mantendrán igual.

Ejercicio 4.17

Enunciado

Dar la función de compilación para las instrucciones de **WHILE** a la MA del ejercicio 4.8. Garantizar la unicidad de las etiquetas incorporando el parámetro adicional *siguiente etiqueta sin usar* (sig).

Resolución

De nuevo, las únicas instrucciones que cambiarán son los `if` y los `while`. Diremos que:

- Condicional:

  $$\begin{aligned}\mathcal{CA}[\![\texttt{if}\ b\ \texttt{then}\ S_1\ \texttt{else}\ S_2, l]\!] = \mathcal{CB}[\![b]\!] &: \texttt{JUMPFALSE}-l : \mathcal{CS}[\![S_1, l+2]\!] : \texttt{JUMP}-(l+1)\\ &: \texttt{LABEL}-l : \mathcal{CS}[\![S_2, l+2]\!] : \texttt{LABEL}-(l+1).\end{aligned}$$

  Es decir, evaluamos b si es falso saltamos al *label* que hemos puesto justo después del bloque S_1. Si se ejecuta S_1, tenemos que saltar al final del condicional para que no se

ejecute S_2 también.

- Bucle:

$$\begin{aligned}\mathcal{CA}[\![\texttt{while}\ b\ \texttt{do}\ S]\!] = \texttt{LABEL}{-}l : \mathcal{CB}[\![b]\!] : \texttt{NEG} : \texttt{JUMPFALSE} - (l+1) \\ : \mathcal{CS}[\![S, l+2]\!] : \texttt{JUMP}{-}l : \texttt{LABEL}{-}(l+1)\,.\end{aligned}$$

  Es decir, creamos un *label* al que saltaremos una vez iteremos sobre el bucle y evaluamos la condición del `while`. Si es cierta, saltamos al final de bucle y si no, ejecutamos S.

Ejercicio 4.23

Enunciado

Supongamos que la traducción de **`skip`** es no generar código. ¿Complica esto la demostración de la equivalencia entre la semántica operacional de paso largo y la semántica derivada de la compilación a AM?

Resolución

Lo veremos en dos partes. En primer lugar, la prueba de que si $\langle S, s\rangle \to s'$, entonces $\langle \mathcal{CS}[\![S]\!], \varepsilon, s\rangle \triangleright^* \langle \varepsilon, \varepsilon, s'\rangle$, no cambia en absoluto. Sin embargo, la demostración de la inversa tiene que modificarse ligeramente. Ahora el caso base de $k = 0$ no es trivial. Si denotamos por $S_n :=$ **skip** ; **skip** ; . . . **skip** tenemos que $\langle \mathcal{CS}[\![S_n]\!] = \varepsilon, \varepsilon, s\rangle \triangleright^0 \langle \varepsilon, \varepsilon, s'\rangle$. En este caso, es obvio que $s = s'$. Veamos ahora por inducción sobre n que se cumple lo que buscamos:

- $n = 1$. En este caso $S_1 =$ **`skip`** por lo que solo podemos aplicar $[\mathrm{skip}_{\mathrm{ns}}]$ y obtenemos:

$$\langle S_1, s\rangle \to s.$$

- Caso inductivo. Supongamos cierta la hipótesis para n. Como $S_{n+1} = S_n$; **`skip`** y $\langle$**skip**$, s\rangle \to s$ y, por HI, $\langle S_n, s\rangle \to s$, podemos aplicar $[\mathrm{comp}_{\mathrm{ns}}]$ y obtenemos que

$$\langle S_{n+1}, s\rangle \to s.$$