TEORÍA DE LA PROGRAMACIÓN

Facultad de Ciencias Matemáticas, UCM - curso 2021-22

Ejercicio del 3 de abril de 2022

Ejercicio 1 Se desea optimizar la generación de código. Para ello se va a desarrollar un Máquina Abstracta con saltos sin etiquetas.

1. Genera las instrucciones JUMP-z(que siempre realiza el salto) y JUMPFALSE-z(que realizará en caso de que el booleano de la cima de la pila sea falso) siendo z un valor entero, positivo para realizar un salto hacia delante de z-instrucciones y un valor negativo para saltos hacia atrás de z-instrucciones. Modifica la máquina abstracta para que ahora las instrucciones sean una lista fija de la que se ejecutará en cada paso la instrucción apuntada desde la variable nueva pc, que has de añadir a la configuración de la máquina. Es decir, la configuración de la máquina es ahora $\langle pc, c, e, s\rangle \in \mathbb{Z} \times \mathbf{Code} \times \mathbf{Stack} \times \mathbf{State}$

2. Genera los esquemas de traducción para la nueva máquina sin utilizar las instrucciones *BRANCH* y *LOOP*.

3. Demuestra la corrección de la nueva máquina con respecto a alguna de las semánticas vista en clase, también se puede realizar con respcto a la antigua máquina.

B

UNIVERSIDAD COMPLUTENSE MADRID
FACULTAD DE CIENCIAS MATEMÁTICAS

APELLIDOS Llamas Núñez		NOMBRE Juan Carlos	HOJA Nº
ASIGNATURA TPROG		DNI	
CURSO	GRUPO	FECHA	

1.-

Tenemos que dar una nueva semántica operacional para la máquina abstracta. La definimos haciendo uso de la anterior:

$\langle pc, c, e, s \rangle \triangleright_N \langle pc+1, c, e', s' \rangle$ si $\langle c[pc], e, s \rangle \triangleright \langle \varepsilon, e', s' \rangle$

Nótese que en la máquina abstracta original, todas las "instrucciones" menos BRANCH y LOOP no generan nuevo código y, por construcción, c no tendrá BRANCH ni LOOP. Por tanto c[pc] siempre irá a ε.

Ahora:

$\langle pc, c, e, s \rangle \triangleright_N \langle pc+z, c, e, s \rangle$ si $c[pc] = JUMP\text{-}z$.

No necesario

y

$\langle pc, c, tt : e, s \rangle \triangleright_N \langle pc+1, c, e, s \rangle$ si $c[pc] = JUMPFALSE\text{-}z$.

y

$\langle pc, c, ff : e, s \rangle \triangleright_N \langle pc+z, e, s \rangle$ si $c[pc] = JUMP\text{-}FALSE\text{-}z$.

No necesario.

En todos estos casos asumimos que $0 \leq p \leq c.size - 1$. ya que en caso contrario c[pc] no estaría definido y la máquina se bloquearía.

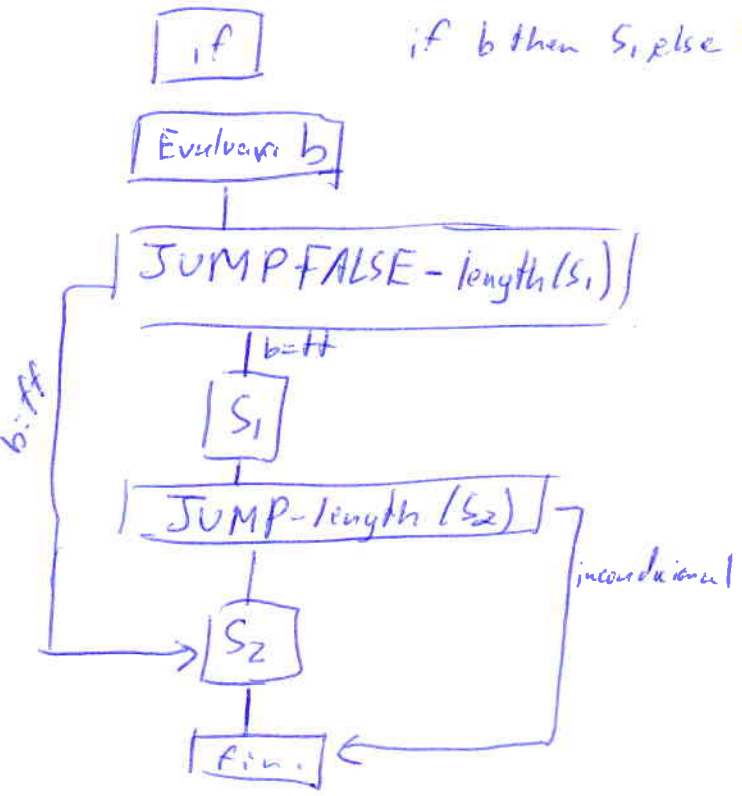
2.- Definimos la nueva traducción.

CA y CB siguen siendo las mismas.

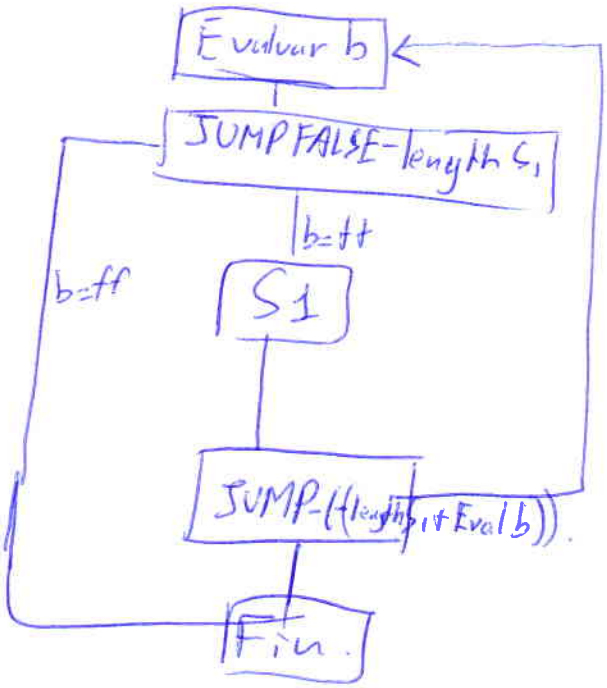
CS[[x:=a]], CS[[skip]], CS[[S1;S2]] siguen siendo las mismas.

Tenemos que definir el if y el while.

Basándonos en la siguiente idea:



if b then S_1 else S_2.



while b do S.

La traducción sería:

$$CS[\![\text{if } b \text{ then } S_1 \text{ else } S_2]\!] = C_{bool}[\![b]\!] : \text{JUMPFALSE-length}(S_1) \overset{+2}{:} CS[\![S_1]\!] : \text{JUMP-length}(S_2) \overset{+1}{:} CS[\![S_2]\!]$$

$$CS[\![\text{while } b \text{ do } S]\!] = C_{bool}[\![b]\!] : \text{JUMPFALSE-length}(S_1) \overset{+2}{:} CS[\![S_1]\!] : \text{JUMP}(-\text{length}(b,S_1)) \overset{-2}{}$$

donde length (S_1) ≡ número de instrucciones de código en las que se traduce S_1 (length $(CS[\![S_1]\!])$.

length (b, S_1) ≡ número de instrucciones de código en las que se traduce la evaluación de b y S_1 (length $(C_{bool}[\![b]\!] : CS[\![S_1]\!])$.

La definición de length es estándar, por casos sobre b y S_1 y la dejo para el final si me da tiempo. Obs: length devuelve un Num, no un Z.

UNIVERSIDAD COMPLUTENSE MADRID
FACULTAD DE CIENCIAS MATEMÁTICAS

APELLIDOS Llamas Núñez		NOMBRE Juan Carlos	HOJA Nº
ASIGNATURA TPROG		DNI	
CURSO	GRUPO	FECHA	

3.- Vamos a probar que s.

(1) $\langle S, s\rangle \longrightarrow s' \Longrightarrow \langle 0; CS[\![S]\!], \varepsilon, s\rangle \; D_N^* \; \langle length(S); CS[\![S]\!], \varepsilon, s'\rangle$

y que

(2) $\langle 0, CS[\![S]\!], \varepsilon, s\rangle \; D_N^* \; \langle length(CS[\![S]\!]), CS[\![S]\!], \varepsilon, s'\rangle \Rightarrow \langle S, s\rangle \longrightarrow s'$.

(1) Demostramos por casos sobre la regla de derivación aplicada:

[ass] Sabemos $\langle x:=a, s\rangle \longrightarrow s'$ luego $s' = s[x \mapsto \mathcal{A}[\![a]\!]s]$.

Ahora

Adaptación del 4.4 y 4.13 para la nueva máquina

$\langle 0; CS[\![x:=a]\!], \varepsilon, s\rangle = \langle 0, CA[\![a]\!] : STORE\text{-}x, \varepsilon, s\rangle \; D_N^*$ Reducción!

$D_N^* \langle length(CA[\![a]\!]), STORE\text{-}x, \mathcal{A}[\![a]\!]s, s\rangle \; D_N \; \langle length(CA[\![a]\!])+1, \varepsilon, \varepsilon, s[x \mapsto \mathcal{A}[\![a]\!]s]\rangle$ ✓

↑ Abuso notación. No creo que es abuso de notación y se tiene el resultado.

Habría que definir recursivamente length(c) con c lista de instrucciones. length : Code → ℕ.

Pasamos a if y while que son más interesantes:

$[if_{ns}^{tt}]$ $\langle$if b then S_1 else $S_2, s\rangle \longrightarrow s'$ porque.

$\langle S_1, s\rangle \longrightarrow s'$ y $\mathcal{B}[\![b]\!]s = tt$.

$\langle 0, CS[\![$if b then S_1 else $S_2]\!], \varepsilon, s\rangle = \langle 0, CA[\![b]\!] : JUMPFALSE\text{-}length(CS[\![S_1]\!]) : CS[\![S_1]\!] :$

$JUMP\text{-}length(CS[\![S_2]\!]) : CS[\![S_2]\!], \varepsilon, s\rangle \; D_N^*$ ← Adaptación del 4.4 y 4.14 Reducción!

$D^* \langle length(CA[\![b]\!]) ; JUMPFALSE\text{-}length(CS[\![S_1]\!]) : CS[\![S_1]\!] : \ldots, \mathcal{B}[\![b]\!]s, s\rangle \; D_N$

↑ $CA[\![b]\!]$; $\mathcal{B}[\![b]\!]s = tt$

$C[\![b]\!]$; JUMPFALSE ...

$D_N < length(C[\![b]\!])+1, CS[\![S_1]\!]; JUMP\text{-}length(CS[\![S_2]\!]); \ldots, \varepsilon, s >$ ✓

~~Por HI estruct~~ Como $< S_1, s > \to s'$ por hipótesis de inducción sobre la "forma del árbol" (S_1 subparte del if)

se tiene que $< 0, CS[\![S_1]\!], \varepsilon, s > D_N^* < length(CS[\![S_1]\!]), CS[\![S_1]\!] \varepsilon, s' >$.

Adaptando el resultado de "pegar por detrás" para la nueva máquina.

y por delante! Reducción!

$< length(C[\![b]\!])+1+\ldots, C[\![b]\!]: JUMPFASE \ldots : CS[\![S_1]\!]: JUMP\text{-}length(CS[\![S_2]\!]) \ldots, \varepsilon, s >$

$D_N^* < length(C[\![b]\!])+1+length(CS[\![S_1]\!]), "code", \varepsilon, s' >$.

Ahora $code[pc] = JUMP\text{-}length(CS[\![S_2]\!])$

$D_N < length(C[\![b]\!])+1+length(CS[\![S_1]\!])+length(CS[\![S_2]\!]), code, \varepsilon, s' >$.

$length(CS[\![S_1; S_2]\!])$ ✓

[while^{tt}] Sabemos que $< while\ b\ do\ S, s > \to s''$ porque

$< S, s > \to s'$, $< while\ b\ do\ S, s' > \to s''$ y $A[\![b]\!]s = tt$.

Ahora llamamos $code = CS[\![while\ b\ do\ S]\!] = C[\![b]\!]: JUMPFALSE\text{-}length(S_1): CS[\![S_1]\!]: JUMP\text{-}(-length(C[\![b]\!]) - length(CS[\![S_1]\!]))$

Adaptación 4.1(a) y 4.4 Reducción! tt

$< 0, code, \varepsilon, s > D_N^* < length(C[\![b]\!]), code, A[\![b]\!]s, s > D_N$

$D_N < length(C[\![b]\!])+1, code, \varepsilon, s > D_N^* < length(C[\![b]\!])+1+length(CS[\![S]\!]), code, \varepsilon, s' > D_N$

↑ Lema de pegar por detrás

Por HI estruct. como $< S, s > \to s' \Rightarrow < 0, CS[\![S]\!], \varepsilon, s > D_N^* < length(CS[\![S]\!]), CS[\![S]\!], \varepsilon, s' >$

$D_N < 0, code, \varepsilon, s' > D_N^* < length(code), code, \varepsilon, s'' >$.

↑ Por HI como $< while\ b\ do\ S, s' > \to s''$

$\Rightarrow < 0, code, \varepsilon, s' > D^* < length(code), \varepsilon, s'' >$

code

Hay que ajustar la definición del salto con ± 1 OK!

UNIVERSIDAD COMPLUTENSE MADRID

FACULTAD DE CIENCIAS MATEMÁTICAS

APELLIDOS Llamas Núñez		NOMBRE Juan Carlos	HOJA Nº
ASIGNATURA		DNI	
CURSO	GRUPO	FECHA	

De la demo de (2) la haremos por inducción sobre k.

Creo que no sale! Mejor una demostración similar a la de

$(\langle 0, CS[\![S]\!], \varepsilon, s\rangle \triangleright_N^k K \text{length}(CS[\![S]\!]), CS[\![S]\!], \varepsilon, s'\rangle \Rightarrow \langle S, s\rangle \to s'$

Si $k=0$, como $\text{length}(CS[\![S]\!]) > 0$ $\forall S$ el resultado es trivialmente cierto.

Sea $k \geq 0$, suponemos el resultado cierto $\forall n = 0, 1, \ldots k$ y vemos que es cierto para $k+1$.

Lo demostramos por casos sobre S.

Si $S =$ if b then do S_1 else S_2 sabemos que

$\langle 0, CS[\![S]\!], \varepsilon, s\rangle \triangleright^{k+1}$