Practica 4: temas 10-12

Alberto Trigueros Postigo

1 Ejercicio 1:

Crea el programa WHILE mas simple posible que compute la función "diverge" , que tiene cero argumentos y computa la codificación.

El programa While más simple posible quiere decir el programa que diverja con cero argumentos que para la función code2N se tenga el menor natural posible. Podemos observar que la función trata de una composición por cada una de las sentencias a la función sent2N, que asigna a una sentencia un natural. Es decir, cerciorandonos que obtengamos el menor natural posible en cada línea de la función sent2N sin olvidar que debe diverger, hemos terminado.

"Minimicemos sent2N" para ello veamos la definición (figura2):

Definition 11.1.6. Encoding of While codes
Given $c = s_1; \ldots; s_m \in \text{CODE}$, where each s_i is a sentence,

$$code2N(c) = \Gamma\big(sent2N(s_1), \ldots, sent2N(s_m)\big) - 1$$

Figure 1:

Visualmente apreciamos que las sentencias con imagen menor serán las más altas pues las más bajas serán una cantorización más un natural cada vez más alto. Por ello escogemos sentencias lo más altas posibles. Como es una función sin argumentos, las variables que usemos están inicializadas a cero. Obligatoriamente debemos sumar un natural a X1 por ejemplo, hemos aplicado la tercera sentencia. Como debe diverger, aplicamos un bucle forzadamente (será una sentencia 5, muy alta pero necesaria) Por último, el bucle no puede estar vacío por definición por lo que usaremos la segunda sentencia para no hacer nada dentro del mismo pero que no esté vacío. Así obtenemos el programa:

X1 := X1 + 1
WHILE (X1 ! = 0) do
X1 := X1
od

Efectivamente diverge y será el menor programa while que lo haga.

Definition 11.1.5. Encoding of While sentences

$$sent2N:\ \text{CODE}^1 \to \mathbb{N} \text{ with } \text{CODE}^1 = \mathcal{L}\big((N, T, P, < \text{sentence} >)\big)$$

$$sent2N(c) = \begin{cases} 5(i-1) & \text{if } c = \texttt{Xi} \coloneqq \texttt{0} \\ 5\sigma_1^2(i-1, j-1) + 1 & \text{if } c = \texttt{Xi} \coloneqq \texttt{Xj} \\ 5\sigma_1^2(i-1, j-1) + 2 & \text{if } c = \texttt{Xi} \coloneqq \texttt{Xj + 1} \\ 5\sigma_1^2(i-1, j-1) + 3 & \text{if } c = \texttt{Xi} \coloneqq \texttt{Xj - 1} \\ 5\sigma_1^2\big(i-1, code2N(b)\big)+4 & \text{if } c = \texttt{while Xi} \neq \texttt{0 do}b\texttt{od} \end{cases}$$

Figure 2:

2 Ejercicio 2:

Crea un script de Octave que enumere todos los vectores.
Pensado en frío parece algo sumamente complicado pero tenemos la función ya definida en el repositorio llamada "godeldecoding.m" que es una biyección tal y como hemos visto en teoría que asocia un natural a un vector de cualquier longitud. Así tenemos todos los vectores enumerados.

He creado el script "ejercicio2.m" que se incluye en la práctica que en un bucle va aplicando esta función para cada natural de forma:

```
    t = 1;
while (t > 0)
disp(godeldecoding(t));
t = t + 1;
endwhile
```

al ejecutar de consola tenemos un bucle infinito representando vectores como queríamos:

3 Ejercicio 3:

Crea un script que enumere todos los lenguajes while. Lo hacemos análogamente al ejercicio 2, pero con la función N2WHILE que está definida en el repositorio de la asignatura. Esta función establece una biyección entre los naturales y el conjunto de los programas while, justo lo buscado. El programa consta

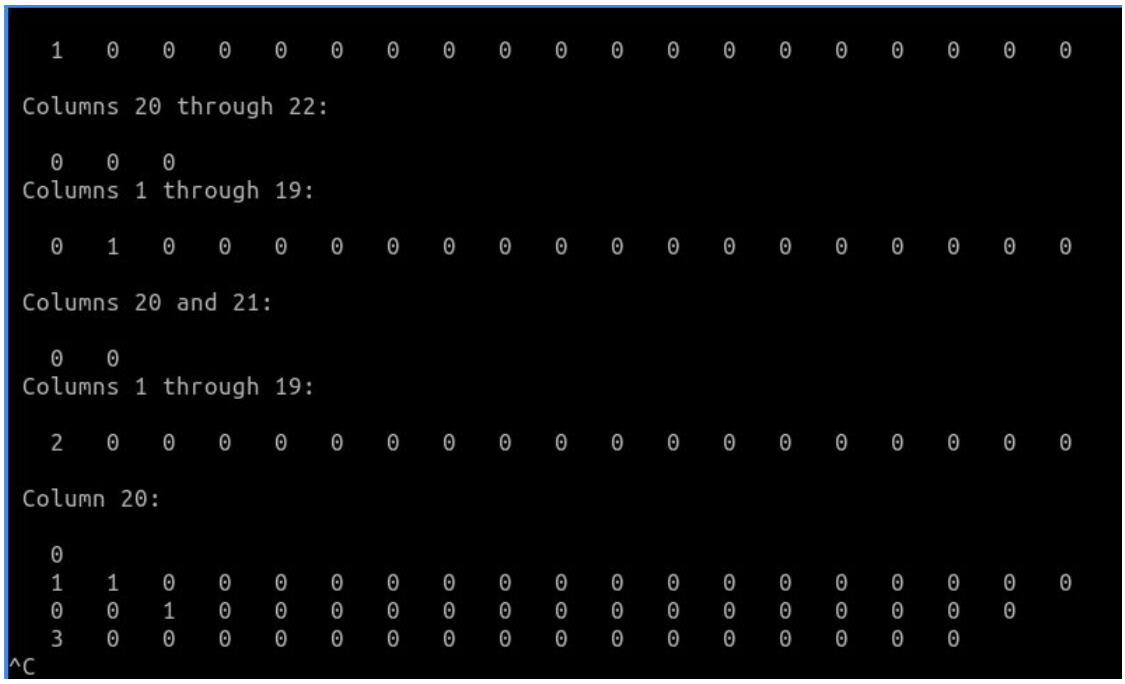

Figure 3:

de:

```
    t = 1;
while (t > 0)
disp(N2WHILE(t));
t = t + 1;
endwhile
```

Crea una ejecución por consola como podemos observar infinita que a cada natural le asocia un único programa while:

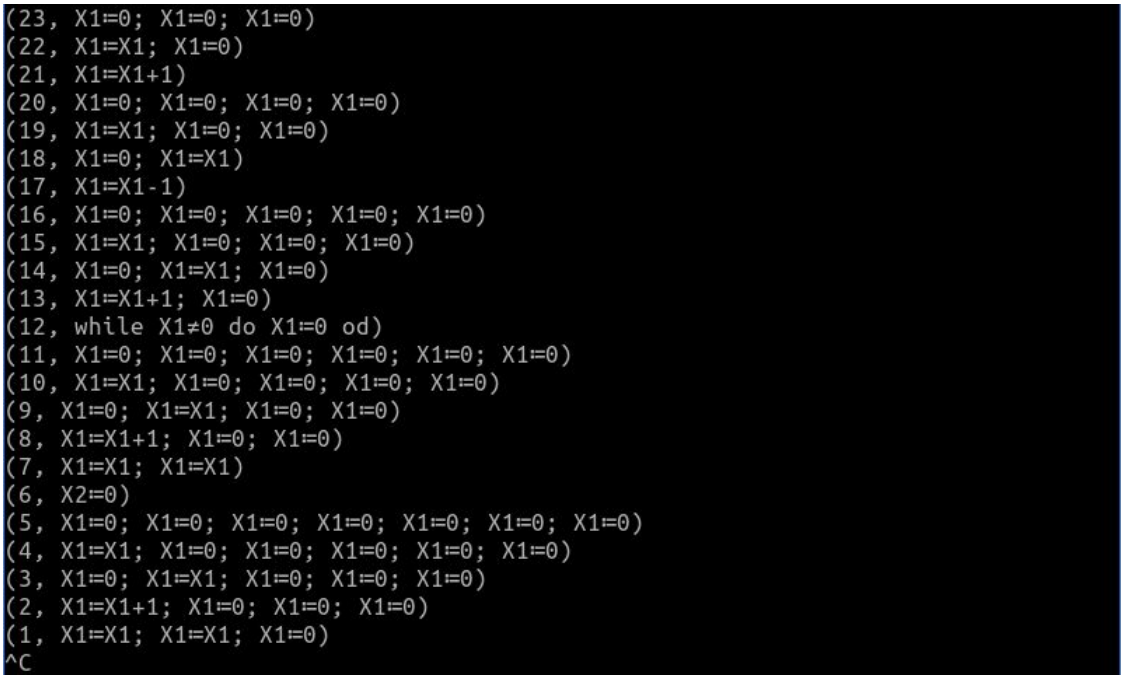
(23, X1:=0; X1:=0; X1:=0)
(22, X1:=X1; X1:=0)
(21, X1:=X1+1)
(20, X1:=0; X1:=0; X1:=0; X1:=0)
(19, X1:=X1; X1:=0; X1:=0)
(18, X1:=0; X1:=X1)
(17, X1:=X1-1)
(16, X1:=0; X1:=0; X1:=0; X1:=0; X1:=0)
(15, X1:=X1; X1:=0; X1:=0; X1:=0)
(14, X1:=0; X1:=X1; X1:=0)
(13, X1:=X1+1; X1:=0)
(12, while X1≠0 do X1:=0 od)
(11, X1:=0; X1:=0; X1:=0; X1:=0; X1:=0; X1:=0)
(10, X1:=X1; X1:=0; X1:=0; X1:=0; X1:=0)
(9, X1:=0; X1:=X1; X1:=0; X1:=0)
(8, X1:=X1+1; X1:=0; X1:=0)
(7, X1:=X1; X1:=X1)
(6, X2:=0)
(5, X1:=0; X1:=0; X1:=0; X1:=0; X1:=0; X1:=0; X1:=0)
(4, X1:=X1; X1:=0; X1:=0; X1:=0; X1:=0; X1:=0)
(3, X1:=0; X1:=X1; X1:=0; X1:=0; X1:=0)
(2, X1:=X1+1; X1:=0; X1:=0; X1:=0)
(1, X1:=X1; X1:=X1; X1:=0)
^C

Figure 4: