TEORÍA DE AUTÓMATAS Y LENGUAJES FORMALES

PRÁCTICA 4

EJERCICIO 1. Calculate the smallest number of a WHILE code that computes the diverge function (with zero arguments).

Para crear la función WHILE más pequeña que diverja, simplemente tendremos que crear el bucle más sencillo del que la función no pueda salir. Por ello, en primer lugar, nos aseguramos de que la única variable (X_1) tenga valor positivo mayor que cero. Ahora sólo tendremos que entrar en un bucle donde X_1 se actualice a su propio valor (es decir, no cambie en ninguna de las iteraciones), y que se ejecutará mientras X_1 no valga cero. Es evidente que esta condición nunca se cumplirá, por tanto, será imposible salir del bucle, lo que lleva a que la función diverge. El código sería el siguiente:

Q=(0,div)

div:

$X_1 := X_1 + 1;$

while $X_1 \neq 0$ **do**

$X_1 := X_1;$

od

EJERCICIO 2. Create an Octave script to print all the vectors.

La solución del problema se reduce a crear un bucle que imprima en cada iteración uno de los vectores. Por ello, si queremos imprimir los primeros n vectores, teniendo en cuenta que se puede establecer una biyección entre éstos y los números naturales, el código sería el siguiente:

```
function imprVectores(n)
        for j=1:n
                disp(['(' num2str(godeldecoding(j-1)) ')'])
        end
end
```

Ahora probamos su funcionamiento en Octave. Así obtenemos los primeros 30 vectores:

```
>> imprVectores(30)
()
(0)
(0  0)
(1)
(0  0  0)
(1  0)
(2)
(0  0  0  0)
(1  0  0)
(0  1)
(3)
(0  0  0  0  0)
(1  0  0  0)
(0  0  1)
(2  0)
(4)
(0  0  0  0  0  0)
(1  0  0  0  0)
(0  0  0  1)
(0  1  0)
(1  1)
(5)
(0  0  0  0  0  0  0)
(1  0  0  0  0  0)
(0  0  0  0  1)
(0  0  1  0)
(1  0  1)
(0  2)
(6)
(0  0  0  0  0  0  0  0)
```

EJERCICIO 3. Create an Octave script to print all the WHILE programs.

La solución a este problema no difiere demasiado con respecto a la del ejercicio anterior. De hecho, el razonamiento es el mismo, puesto que también existe una biyección entre los números naturales y los programas WHILE. De esta forma, el programa que muestra los n primeros programas WHILE es el siguiente:

```
function imprProgramasWhile(n)
        for j=1:n
                disp(N2WHILE(j-1))
        end
end
```

Probemos su funcionamiento en Octave. Así obtenemos los primeros 30 programas While:

```
>> imprProgramasWhile(30)
(0, X1=0)
(1, X1=0)
(0, X1=0; X1=0)
(2, X1=0)
(1, X1=0; X1=0)
(0, X1=X1)
(3, X1=0)
(2, X1=0; X1=0)
(1, X1=X1)
(0, X1=0; X1=0; X1=0)
(4, X1=0)
(3, X1=0; X1=0)
(2, X1=X1)
(1, X1=0; X1=0; X1=0)
(0, X1=X1; X1=0)
(5, X1=0)
(4, X1=0; X1=0)
(3, X1=X1)
(2, X1=0; X1=0; X1=0)
(1, X1=X1; X1=0)
(0, X1=X1+1)
(6, X1=0)
(5, X1=0; X1=0)
(4, X1=X1)
(3, X1=0; X1=0; X1=0)
(2, X1=X1; X1=0)
(1, X1=X1+1)
(0, X1=0; X1=0; X1=0; X1=0)
(7, X1=0)
(6, X1=0; X1=0)
```