Teoría de Autómatas y Lenguajes Formales

Práctica 4: Numeración de programas y lenguaje WHILE extendido

Lázaro, Vargas García

2 de diciembre de 2022

1. Define el programa WHILE más simple que compute una función divergente y codifícalo

Tenemos que intentar que el programa WHILE que usemos sea tal que al codificarlo, el número natural resultante sea lo más bajo posible. Para obtener una intuición de cómo hacerlo, es importante observar cómo está definida la función que codifica un programa WHILE en un número natural.
Tenemos que la función $WHILE2N$ está definida como sigue: Si $(n, c) \in WHILE$, entonces:

$$WHILE2N(n, c) = \sigma_1^2(n, CODE2N(c))$$

Siendo $\sigma_1^2(x, y) = \frac{(x+y)(x+y+1)}{2} + y$. Vemos que el valor de la cantorización es directamente proporcional a las coordenadas del vector argumento. Por tanto, que el programa tenga 0 argumentos es lo más óptimo para conseguir una entrada lo más pequeña posible.

Por otro lado, $CODE2N$, es la godelización del vector que obtenemos como resultado de codificar cada una de las sentencias de nuestro programa, que se hace con $SENT2WHILE$. Esta función está definida de la siguiente forma:

$$SENT2WHILE(s) =$$

$$\begin{cases} 5(i-1) & si \quad s = X_i := 0 \\ 5\sigma_1^2(i-1, j-1) + 1 & si \quad s = X_i := X_j \\ 5\sigma_1^2(i-1, j-1) + 2 & si \quad s = X_i := X_j + 1 \\ 5\sigma_1^2(i-1, j-1) + 3 & si \quad s = X_i := X_j - 1 \\ 5\sigma_1^2(i-1, code2N(b)) + 4 & si \quad s = while X_i ! = 0 \ \ dobod \end{cases}$$

Esto nos quiere decir que, como la godelización sigue un cierto orden, $CODE2N$ devolverá un valor que será proporcional a los números del vector, por lo que debemos intentar que en cada sentencia hagamos la operación más simple posible y utilizando los índices más bajos posibles, ya que $SENT2N$ depende principalmente de estos dos aspectos. Además, cuanto menos sentencias haya, mejor.

Todo esto nos lleva a elegir este programa:

Diverg = (0, s)

s:
while $X_1 = 0$ **do**
$X_1 := 0$;
od

```
octave:13> F("(0, while X1=0 do X1:=0;  od)", [])
complexity has reached 1000, press Ctrl-C to stop, or any other key to continue.
..
^[complexity has reached 2000, press Ctrl-C to stop, or any other key to continu
e...
```

Vemos que este programa diverge, ya que al ejecutarlo en Octave su ejecución no termina. Además, utiliza las variables de menor índice posible, y parece imposible que un programa con menos sentencias pueda diverger.

Además, para codificar $Diverg$, lo hacemos usando el fichero $WHILE2N.m$ El resultado es,

$$WHILE2N(Diverg) = 119$$

```
octave:5> WHILE2N(0, "while X1=0 do X1:=0;  od")
ans = 119
```

2. Crea un script de octave que enumere todos los vectores

He creado este script usando el script *GodelDecoding.m*, el cual devuelve el vector asociado al número natural que se introduce como argumento. He usado entoces un bucle infinito con un contador i que se va incrementando indefinidamente, y en cada iteración se hace una llamada a *godeldecoding* usando i como parámetro.

```
function res = vectores()
i=0
while(i>=0)
  res = godeldecoding(i)
  i = i + 1
  printf('\n')
end
```

La salida de el script es la siguiente (se encuentra en la siguiente página):

```
octave:1> vectores()
i = 0
res = [](0x0)
i = 1

res = 0
i = 2

res =

  0  0

i = 3

res = 1
i = 4

res =

   0   0   0

i = 5

res =

  1  0

i = 6

res = 2
i = 7

res =

   0   0   0   0

i = 8

res =

   1   0   0

i = 9

res =

  0  1

i = 10

res = 3
i = 11
```

3. Crea un script de octave que enumere todos los programas WHILE

La elaboración de este script es totalmente análoga al anterior, solo que se usa el fichero $N2WHILE.m$.

```
function res = programasWhile()
i=0
while(i>=0)
  res = N2WHILE(i)
  i = i + 1
  printf('\n')
end
```

La salida de el script es la siguiente (se encuentra en la siguiente página):

```
octave:1> programasWhile()
i = 0
res = (0, X1:=0)
i = 1

res = (1, X1:=0)
i = 2

res = (0, X1:=0; X1:=0)
i = 3

res = (2, X1:=0)
i = 4

res = (1, X1:=0; X1:=0)
i = 5

res = (0, X1:=X1)
i = 6

res = (3, X1:=0)
i = 7

res = (2, X1:=0; X1:=0)
i = 8

res = (1, X1:=X1)
i = 9

res = (0, X1:=0; X1:=0; X1:=0)
i = 10

res = (4, X1:=0)
i = 11

res = (3, X1:=0; X1:=0)
i = 12

res = (2, X1:=X1)
i = 13

res = (1, X1:=0; X1:=0; X1:=0)
i = 14

res = (0, X1:=X1; X1:=0)
i = 15

res = (5, X1:=0)
i = 16

res = (4, X1:=0; X1:=0)
i = 17
```