Práctica 3 TALF: Temas 7-9

Alberto Trigueros Postigo

1 Ejercicio 1

Define una Máquina de Turing solución del ejercicio 3.4 de la relación y prueba su correcto funcionamiento.

El ejercicio 3.4 propone lo siguiente:
Probar que la función add(x,y) = x+y es turing-computable usando la notación unaria {|}. Debes crear una TM con dos argumentos separados por un símbolo en blanco que empieza y termina detrás de ambas cadenas.

Tal y como vimos en clase, una Máquina de Turing que resuelve nuestro problema puede ser la que se muestra a continuación: Su idea principal es rellenar el símbolo en blanco que separa las cadenas y restar un "|" para que el número que represente sea nuestro buscado x+y y no x+y+1 por haber añadido un "|".

$$\begin{bmatrix} 0 & * & l & 1 \\ 0 & | & | & 0 \\ 1 & * & | & 2 \\ 1 & | & l & 1 \\ 2 & * & l & 3 \\ 2 & | & r & 2 \\ 3 & * & l & 4 \\ 3 & | & * & 3 \\ 4 & * & h & 4 \\ 4 & | & * & 4 \end{bmatrix}$$

Se ha escogido así ya que inicialmente (estado cero) siempre vamos a estar en el espacio en blanco (iniciamos desde detrás de ambas cadenas). Pasamos al estado uno, el cual se moverá hacia la izquierda hasta encontrar el espacio en blanco, que se rellenará cuando se lea. Así pasamos al estado dos, que trata de ir a la derecha hasta llegar al principio desde dónde se ha comenzado, el cual vuelve al último "|". Pasamos en este último paso al estado 3, consiste en cambiar el "|" por el símbolo en blanco para asi terminar nuestra operación. No olvidemos que debemos terminar a la derecha de la cadena por lo que el último estado, 4 se dedica a ello.

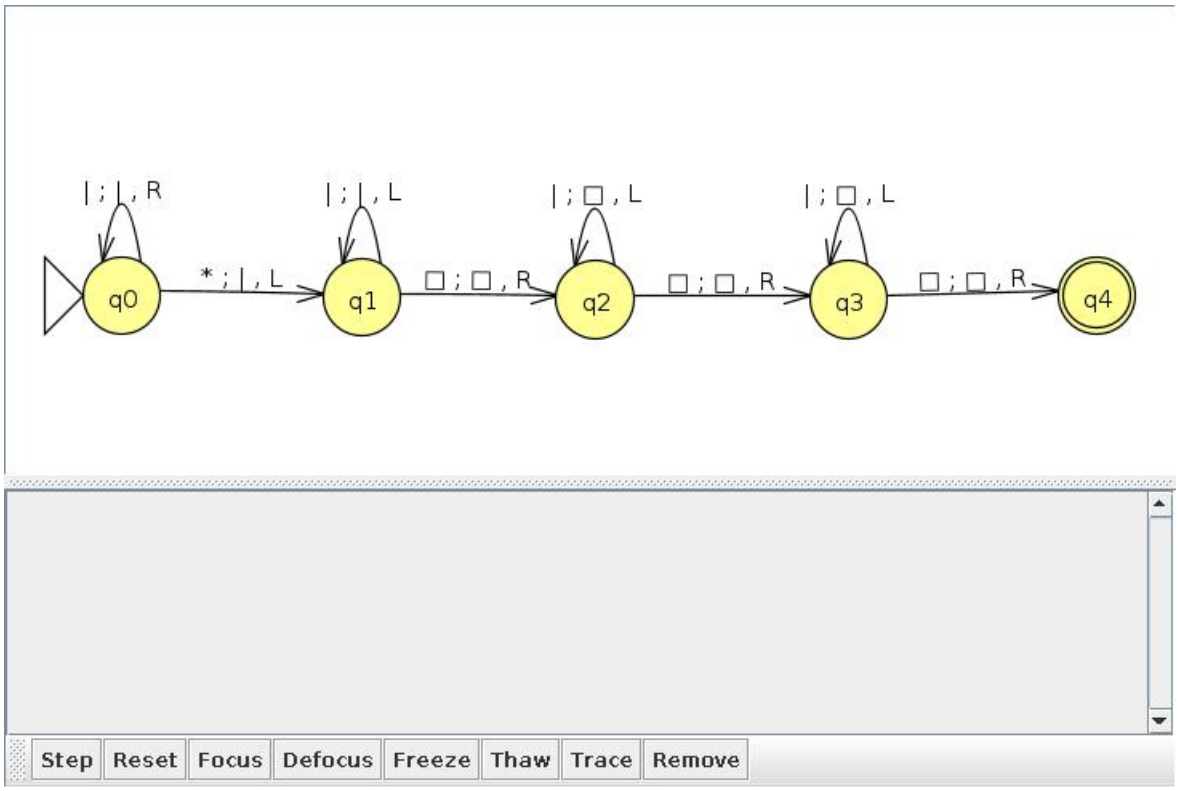
| ; | , R
| ; | , L
| ; □ , L
| ; □ , L
q0
* ; | , L
q1
□ ; □ , R
q2
□ ; □ , R
q3
□ ; □ , R
q4
Step
Reset
Focus
Defocus
Freeze
Thaw
Trace
Remove

Figure 1:

Para ello hemos creado una máquina de turing en JFLAP como se muestra:

Es la misma idea pero en JFLAP se comienza desde antes de la última cadena y no en el espacio en blanco, por lo que nos movemos a la derecha y para la aplicación, el espacio en blanco es un cuadrado.

Para comprobar que funciona, probamos la entrada $|| * ||$ que se transforma efectivamente en ||||:

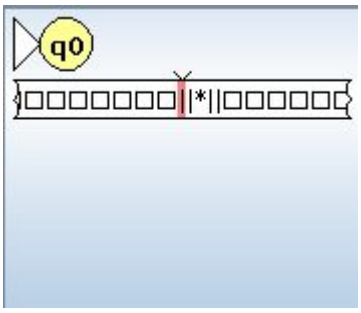

Figure 2:

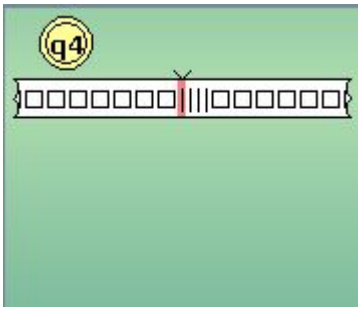

Figure 3:

2 Ejercicio 2:

Define una función recursiva para la suma de tres valores:

Como se expone en clase, la suma de dos valores se define con recursión primitiva como sigue: $suma(x, y) =< \pi_1^1 | \sigma(\pi_3^3) > (x, y)$

Para que se pueda añadir un valor más a dicha recursión debemos modificarla, con un caso base y caso general (las g y h de la definición)

Una opción es, para el caso base usar la suma de dos valores que hemos explicado anteriormente y el caso general será al último valor (cuatro componentes pues es una más que la de la función estudiada) al cual se le suma uno para sumar uno hasta que se acabe el contador (como es el valor se sumará uno las veces que queremos) quedaría de forma:

$suma(x,y,z) =<< \pi_1^1|\sigma(\pi_3^3) > |\sigma(\pi_4^4) > (x,y,z)$

Lo podemos ver en octave:
$evalrecfunction('<< \pi_1^1|\sigma(\pi_3^3) > |\sigma(\pi_4^4) >', 1, 2, 3)$
Se encuentra disponible en un programa octave en el repositorio. Da una salida por consola como la vista en la figura 4 (al final del documento).

3 Ejercicio 3:

Implementar un programa while que compute la suma de tres valores:

-NOTA-: me da problemas el editor a la hora de poner el programa while en el estilo proporcionado en el campus

while X2 ! = 0 do
X1 := X1 + 1;
X2 := X2 - 1;
od
while X3 ! = 0 do
X1 := X1 + 1;
X3 := X3 - 1;
od

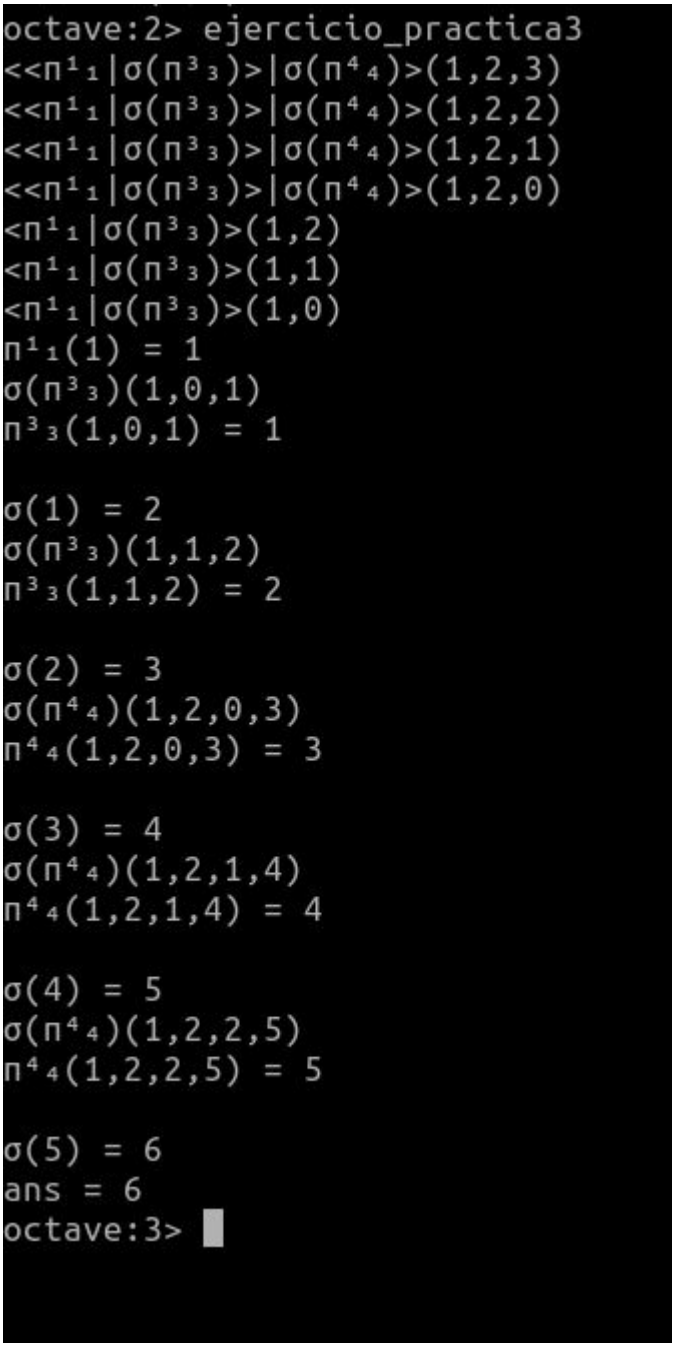
octave:2> ejercicio_practica3
<<π¹₁|σ(π³₃)>|σ(π⁴₄)>(1,2,3)
<<π¹₁|σ(π³₃)>|σ(π⁴₄)>(1,2,2)
<<π¹₁|σ(π³₃)>|σ(π⁴₄)>(1,2,1)
<<π¹₁|σ(π³₃)>|σ(π⁴₄)>(1,2,0)
<π¹₁|σ(π³₃)>(1,2)
<π¹₁|σ(π³₃)>(1,1)
<π¹₁|σ(π³₃)>(1,0)
π¹₁(1) = 1
σ(π³₃)(1,0,1)
π³₃(1,0,1) = 1
σ(1) = 2
σ(π³₃)(1,1,2)
π³₃(1,1,2) = 2
σ(2) = 3
σ(π⁴₄)(1,2,0,3)
π⁴₄(1,2,0,3) = 3
σ(3) = 4
σ(π⁴₄)(1,2,1,4)
π⁴₄(1,2,1,4) = 4
σ(4) = 5
σ(π⁴₄)(1,2,2,5)
π⁴₄(1,2,2,5) = 5
σ(5) = 6
ans = 6
octave:3>

Figure 4: