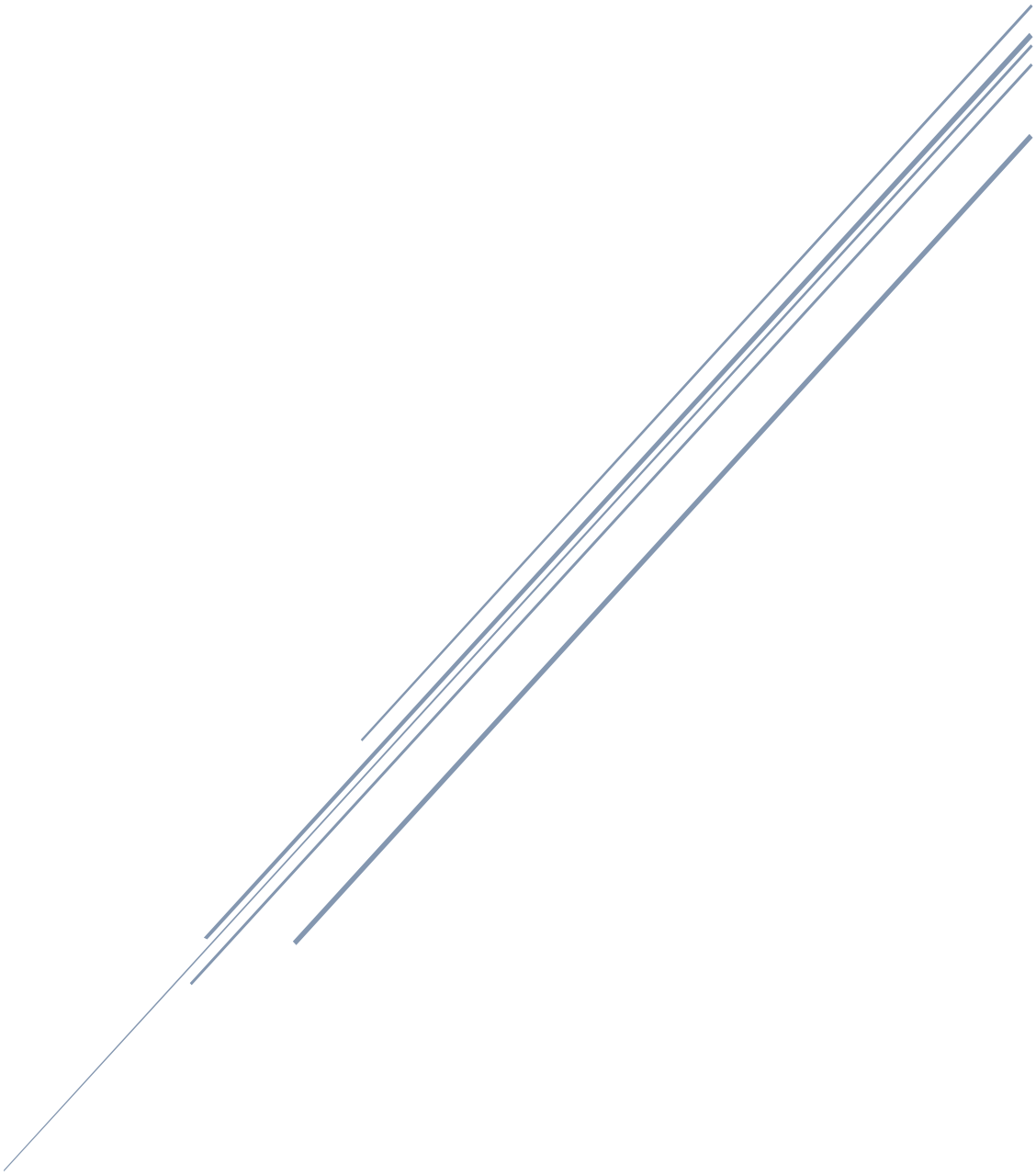
PROGRAMAR FUNCIÓN ACKERMANN - THREADS

MFC – Modelos Formales de Computación



MITSS
Sergi Sanz Carreres

Índice

Función de Ackermann

La función Ackermann es una función recursiva encontrada en 1926 por Wilhelm Ackermann, esta función toma dos números naturales como argumentos y devuelve un único número natural. Como norma general se define como sigue:

$$A(m,n) = \begin{cases} n+1, & \text{si } m = 0; \\ A(m-1,1), & \text{si } m > 0 \text{ y } n = 0; \\ A(m-1, A(m,n-1)), & \text{si } m > 0 \text{ y } n > 0 \end{cases}$$

Una de las particularidades de la función Ackermann es que no cae en las consideradas funciones recursivas primitivas, debido principalmente a su desmesurado crecimiento.

Este teorema se lleva a cabo mediante la técnica de reducción al absurdo, el cual es uno de los métodos lógicos de demostración más usado en matemáticas para demostrar la validez (o invalidez) de proposiciones categóricas.

Ejemplo Recursivo:

Antes de cumplir con las necesidades requeridas para el desarrollo de este ejercicio haciendo uso de la ejecución de hilos, se ha procedido a desarrollar de forma recursiva esta función con el siguiente fragmento de código en Maude:

```
search eval(
  let rec 'ack(m , n) =
  if m == 0 then (n + 1)
    else if n == 0 then 'ack((m - 1), 1)
    else 'ack((m - 1), 'ack(m, (n - 1)))
  in 'ack(1, 2)
) =>! X:[Value] .
```

Dando el siguiente resultado:

```
==========================================
search in CONCURRENT-FUN-SEMANTICS : eval (let rec 'ack (m,n) = if m == 0 then
    n + 1 else if n == 0 then 'ack (m - 1,1) else 'ack (m - 1,'ack (m,n - 1))
    in 'ack (1,2)) =>! X:[ValueList] .

Solution 1 (state 40)
states: 41  rewrites: 371 in 1ms cpu (1ms real) (362658 rewrites/second)
X:[ValueList] --> int(4)
```

Ejemplo Concurrencia:

Para la realización de este ejemplo se requiere crear un nuevo hilo de ejecución para cada llamada recursiva.

Por tanto, para evitar problemas con las variables, se ha modificado ligeramente el código anterior añadiendo una variable g que hará la función de la variable n pero de manera global, para que los distintos hilos sobrescriban g y no se pierdan los valores al hacer uso de la concurrencia.

Además de eso, para evitar condiciones de carrera y evitar que se obtengan resultados anómalos en las variables, se ha optado por la utilización de un bucle while, cuyo fin es el de solucionar estas situaciones.

De manera que el fragmento de código sería el siguiente:

```
search eval (
  let rec g = 0 and b = 0 and c = 0 and d = 0 and 'ack(m,n,z) =
    spawn (if m == 0 then if b == 0 then g := n + 1
                                    else { d := n + 1 ; c := c - 1 }
           else if m >= 1 and n == 0 then 'ack(m - 1,1,z)
           else if m >= 1 and n >= 1
               then {
                    c := c + 1 ;
                    b := b + 1 ;
                    'ack(m,n - 1,z + 1) ;
                    while (c >= z + 1) {} ;
                    b := b - 1 ;
                    'ack(m - 1,d,z)
                    }
           else [])
   in {
     'ack(1,2,0) ;
     while (g == 0 or b >= 1) {} ;
     g
   }
) =>! X:[Value] .
```

Resultados obtenidos:

Para la correcta comprobación de los resultados se ha optado por utilizar la siguiente tabla para comprobar que los resultados obtenidos son correctos

Values of $A(m, n)$

m \ n	0	1	2	3	4	n
0	1	2	3	4	5	$n+1$
1	2	3	4	5	6	$n+2 = 2+(n+3)-3$
2	3	5	7	9	11	$2n+3 = 2\cdot(n+3)-3$
3	5	13	29	61	125	$2^{(n+3)}-3$
4	13 $= 2^{2^2}-3$ $= 2 \uparrow\uparrow 3-3$	65533 $= 2^{2^{2^2}}-3$ $= 2 \uparrow\uparrow 4-3$	$2^{65536}-3$ $= 2^{2^{2^{2^2}}}-3$ $= 2 \uparrow\uparrow 5-3$	$2^{2^{65536}}-3$ $= 2^{2^{2^{2^{2^2}}}}-3$ $= 2 \uparrow\uparrow 6-3$	$2^{2^{2^{65536}}}-3$ $= 2^{2^{2^{2^{2^{2^2}}}}}-3$ $= 2 \uparrow\uparrow 7-3$	$\underbrace{2^{2^{\cdot^{\cdot^{2}}}}}_{n+3}-3$ $= 2 \uparrow\uparrow (n+3)-3$
5	65533 $= 2 \uparrow\uparrow (2 \uparrow\uparrow 2)-3$ $= 2 \uparrow\uparrow\uparrow 3-3$	$2 \uparrow\uparrow\uparrow 4-3$	$2 \uparrow\uparrow\uparrow 5-3$	$2 \uparrow\uparrow\uparrow 6-3$	$2 \uparrow\uparrow\uparrow 7-3$	$2 \uparrow\uparrow\uparrow (n+3)-3$
6	$2 \uparrow\uparrow\uparrow\uparrow 3-3$	$2 \uparrow\uparrow\uparrow\uparrow 4-3$	$2 \uparrow\uparrow\uparrow\uparrow 5-3$	$2 \uparrow\uparrow\uparrow\uparrow 6-3$	$2 \uparrow\uparrow\uparrow\uparrow 7-3$	$2 \uparrow\uparrow\uparrow\uparrow (n+3)-3$
m	$(2 \to (3) \to (m-2))-3$	$(2 \to (4) \to (m-2))-3$	$(2 \to (5) \to (m-2))-3$	$(2 \to (6) \to (m-2))-3$	$(2 \to (7) \to (m-2))-3$	$(2 \to (n+3) \to (m-2))-3$

Ejemplo 1:

Para esta ejecución se ha optado por la utilización de n= -2 y m = 1, de manera que al tratarse de una n con un valor negativo la ejecución no debe devolver ningún valor:

```
========================================
search in CONCURRENT-FUN-SEMANTICS : eval (let rec g = 0 and b = 0 and c = 0
    and d = 0 and 'ack (m,n,z) = spawn(if m == 0 then if b == 0 then g := n + 1
    else {d := n + 1 ; c := c - 1} else if (m >= 1) and (n == 0) then 'ack (m -
    1,1,z) else if (m >= 1) and (n >= 1) then {c := c + 1 ; b := b + 1 ; 'ack (
    m,n - 1,z + 1) ; while c >= z + 1 {} ; b := b - 1 ; 'ack (m - 1,d,z)} else
    []) in {'ack (1,-2,0) ; while (g == 0) or (b >= 1) {} ; g}) =>! X:[
    ValueList] .

No solution.
```

Ejemplo 2:

Para esta ejecución se ha optado por la utilización de n= 2 y m = 1, de manera que al tratarse de una n con un valor positivo y comprobando la tabla anterior la ejecución debería devolver el valor 4.

```
========================================
search in CONCURRENT-FUN-SEMANTICS : eval (let rec g = 0 and b = 0 and c = 0
    and d = 0 and 'ack (m,n,z) = spawn(if m == 0 then if b == 0 then g := n + 1
    else {d := n + 1 ; c := c - 1} else if (m >= 1) and (n == 0) then 'ack (m -
    1,1,z) else if (m >= 1) and (n >= 1) then {c := c + 1 ; b := b + 1 ; 'ack (
    m,n - 1,z + 1) ; while c >= z + 1 {} ; b := b - 1 ; 'ack (m - 1,d,z)} else
    []) in {'ack (1,2,0) ; while (g == 0) or (b >= 1) {} ; g}) =>! X:[
    ValueList] .

Solution 1 (state 451)
states: 452  rewrites: 20239 in 32ms cpu (32ms real) (622374 rewrites/second)
X:[ValueList] --> int(4)

No more solutions.
states: 452  rewrites: 20239 in 32ms cpu (32ms real) (621877 rewrites/second)
```

Ejemplo 3:

Para esta ejecución se ha optado por la utilización de n = 1 y m = 2, de manera que al tratarse de una n con un valor positivo y comprobando la tabla anterior la ejecución debería devolver el valor 5.

```
==========================================
search in CONCURRENT-FUN-SEMANTICS : eval (let rec g = 0 and b = 0 and c = 0
    and d = 0 and 'ack (m,n,z) = spawn(if m == 0 then if b == 0 then g := n + 1
    else {d := n + 1 ; c := c - 1} else if (m >= 1) and (n == 0) then 'ack (m -
    1,1,z) else if (m >= 1) and (n >= 1) then {c := c + 1 ; b := b + 1 ; 'ack (
    m,n - 1,z + 1) ; while c >= z + 1 {} ; b := b - 1 ; 'ack (m - 1,d,z)} else
    []) in {'ack (2,1,0) ; while (g == 0) or (b >= 1) {} ; g}) =>! X:[
    ValueList] .

Solution 1 (state 2233)
states: 2234  rewrites: 126102 in 184ms cpu (184ms real) (683587
    rewrites/second)
X:[ValueList] --> int(5)

No more solutions.
states: 2234  rewrites: 126102 in 184ms cpu (184ms real) (683501
    rewrites/second)
```

Ejemplo 4:

Para esta ejecución se ha optado por la utilización de n = 2 y m = 2, de manera que al tratarse de una n con un valor positivo y comprobando la tabla anterior la ejecución debería devolver el valor 7.

```
==========================================
search in CONCURRENT-FUN-SEMANTICS : eval (let rec g = 0 and b = 0 and c = 0
    and d = 0 and 'ack (m,n,z) = spawn(if m == 0 then if b == 0 then g := n + 1
    else {d := n + 1 ; c := c - 1} else if (m >= 1) and (n == 0) then 'ack (m -
    1,1,z) else if (m >= 1) and (n >= 1) then {c := c + 1 ; b := b + 1 ; 'ack (
    m,n - 1,z + 1) ; while c >= z + 1 {} ; b := b - 1 ; 'ack (m - 1,d,z)} else
    []) in {'ack (2,2,0) ; while (g == 0) or (b >= 1) {} ; g}) =>! X:[
    ValueList] .

Solution 1 (state 61017)
states: 61018  rewrites: 5364586 in 10616ms cpu (10809ms real) (505313
    rewrites/second)
X:[ValueList] --> int(7)

No more solutions.
states: 61018  rewrites: 5364586 in 10616ms cpu (10809ms real) (505311
    rewrites/second)
```