# Sesión 1: Programación concurrente

- (Apt and Olderog [3]) Assume that for the function ***f***, there is some integer value ***i*** for which ***f(i)=0***. Here are five concurrent algorithms that search for ***i***. An algorithm is correct if for all scenarios, ***both*** processes terminate after one of them has found the zero. For each algorithm, show that it is correct or find a scenario that is a counterexample.

M. Ben-Ari
**Principles of Concurrent and Distributed Programming**
Addison-Wesley, 2006

| Algorithm 2.11: Zero A | |
|---|---|
| boolean found | |
| **p** | **q** |
| integer i ← 0 | integer j ← 1 |
| p1: found ← false | q1: found ← false |
| p2: while not found | q2: while not found |
| p3: i ← i + 1 | q3: j ← j − 1 |
| p4: found ← f(i) = 0 | q4: found ← f(j) = 0 |

| Algorithm 2.12: Zero B | |
|---|---|
| boolean found ← false | |
| **p** | **q** |
| integer i ← 0 | integer j ← 1 |
| p1: while not found | q1: while not found |
| p2: i ← i + 1 | q2: j ← j − 1 |
| p3: found ← f(i) = 0 | q3: found ← f(j) = 0 |

**Algorithm 2.13: Zero C**

boolean found ← false

| p | | q | |
|---|---|---|---|
| | integer i ← 0 | | integer j ← 1 |
| p1: | while not found | q1: | while not found |
| p2: | i ← i + 1 | q2: | j ← j − 1 |
| p3: | if f(i) = 0 | q3: | if f(j) = 0 |
| p4: | found ← true | q4: | found ← true |

**Algorithm 2.14: Zero D**

```
boolean found ← false
integer turn ← 1
```

| p | q |
|---|---|
| integer i ← 0 | integer j ← 1 |
| p1: while not found | q1: while not found |
| p2: await turn = 1 | q2: await turn = 2 |
| turn ← 2 | turn ← 1 |
| p3: i ← i + 1 | q3: j ← j − 1 |
| p4: if f(i) = 0 | q4: if f(j) = 0 |
| p5: found ← true | q5: found ← true |

**Algorithm 2.15: Zero E**

```
boolean found ← false
integer turn ← 1
```

| p | q |
|---|---|
| integer i ← 0 | integer j ← 1 |
| p1: while not found | q1: while not found |
| p2: await turn = 1 | q2: await turn = 2 |
| turn ← 2 | turn ← 1 |
| p3: i ← i + 1 | q3: j ← j − 1 |
| p4: if f(i) = 0 | q4: if f(j) = 0 |
| p5: found ← true | q5: found ← true |
| p6: turn ← 2 | q6: turn ← 1 |

- Consider the following algorithm where each of ten processes executes the statements with ***i*** set to a different number 1..10.
- What does the algorithm do?
- What would happen if D in line *p3* were replaced by C?
- What would happen if the array C were initialized with values that are not all distinct?

M. Ben-Ari
**Principles of Concurrent and Distributed Programming**
Addison-Wesley, 2006

| **Algorithm 2.16: Concurrent algorithm A** | |
|---|---|
| | integer array[1..10] C ← ten *distinct* initial values<br>integer array[1..10] D |
| | integer myNumber, count |
| p1: | myNumber ← C[i] |
| p2: | count ← number of elements of C less than myNumber |
| p3: | D[count + 1] ← myNumber |

Completad el programa de manera que, una vez terminados todos los procesos, se muestre por la salida estándar el vector D

```
int array[1..10] C := (1..10,valores distintos)
int array[1..10] D
Process P(i:1..10)::
   int myNumber, count

   myNumber := C[i]
   count := number of elements of C
            less than myNumber
   D[count+1] := myNumber
end
```

Considérese el código que se muestra a continuación, en el que se lanzan M procesos concurrentes con el objetivo de que entre todos encuentren el valor máximo de un vector. Una vez todos han acabado, el proceso **informador** muestra el valor máximo del vector. Se pide completar el código y resolver las cuestiones de sincronización de acuerdo a la especificación. No se permite definir nuevos procesos. Es un objetivo que el programa se ejecute de la manera más eficiente posible.

```
constant integer N := ... //lo que sea
                 M := ... //lo que sea
                 NM := N*M
integer array[1..NM] val := ... //lo que sea, ya inicializado
integer max

// código a completar

process P(i: 1..M)::
    // código a completar

    operation maxTrozo(integer array[1..NM] v, int i1,i2): integer
    //Pre: 1<=i1<=NM, 1<=i2<=NM, i1<=i2
    //Post: devuelve el valor máximo de las componentes
    //      v[i1],v[i1+1],...,v[i2]

    // código a completar

    end operation

   // código a completar
   // al terminar todos los procesos, "max" contiene el
   // valor máximo del vector "val"
end process

process informador::
    // código a completar

    write('Max=␣')
    write(max)
end process
```

# Sesión 2: máximo de un vector

Considérese un programa concurrente compuesto por 10 procesos: 5 procesos **lectores** y 5 procesos **calculadores**, que comparten una matriz 5x50 de números enteros. El ejercicio pide completar el esquema de programa que se muestra a continuación, de manera que:

- Cada proceso lector lee de la entrada estándar una secuencia de enteros compuesta por 50 números y rellena una fila completa de la matriz
- Cada proceso calculador busca el máximo parcial de una fila de la matriz
- Una vez que todos los procesos calculadores han hecho su trabajo, el proceso con identificador 1 muestra por la salida estándar el máximo global de la matriz

# Sesión 2: máximo de un

```
integer array[1..5,1..50] D := ...
. . .
Process Lector(i:1..5)::
   . . .
end
Process Calculador(i:1..5)::
   . . .
end
```

Considérese un programa
procesos: 5 procesos **lectores**
5 procesos **calculadores**, q
números enteros. El ejerci
programa que se muestra a continuación, de manera que:

- Cada proceso lector lee de la entrada estándar una secuencia de enteros compuesta por 50 números y rellena una fila completa de la matriz
- Cada proceso calculador busca el máximo parcial de una fila de la matriz
- Una vez que todos los procesos calculadores han hecho su trabajo, el proceso con identificador 1 muestra por la salida estándar el máximo global de la matriz

Modelar mediante Redes de Petri el siguiente programa concurrente y construir su grafo de estados alcanzables

¿Cuál sería el valor final de las variables globales *x* e *y*?

```
integer x:= 1
integer y:= 7

process P
  integer z:= 2

  <z:=2x+1>
  <y:=z+x>
end process

process Q
  integer z:=9

  <z:=y+x>
  <x:=y>
end process
```

Modelar mediante Redes de Petri el siguiente programa concurrente y construir su grafo de estados alcanzables

```
                    integer s := 3

process UNO::       process DOS::       process TRES::
   loop forever        loop forever        loop forever
      <await s>0          <await s>1          <await s>2
         s := s-1            s := s-2            s := s-3
      >                   >                   >
      //SC1               //SC2               //SC3
      <s := s+1>          <s := s+2>          <s := s+1>
   end loop            end loop            end loop
end process         end process         end process
```

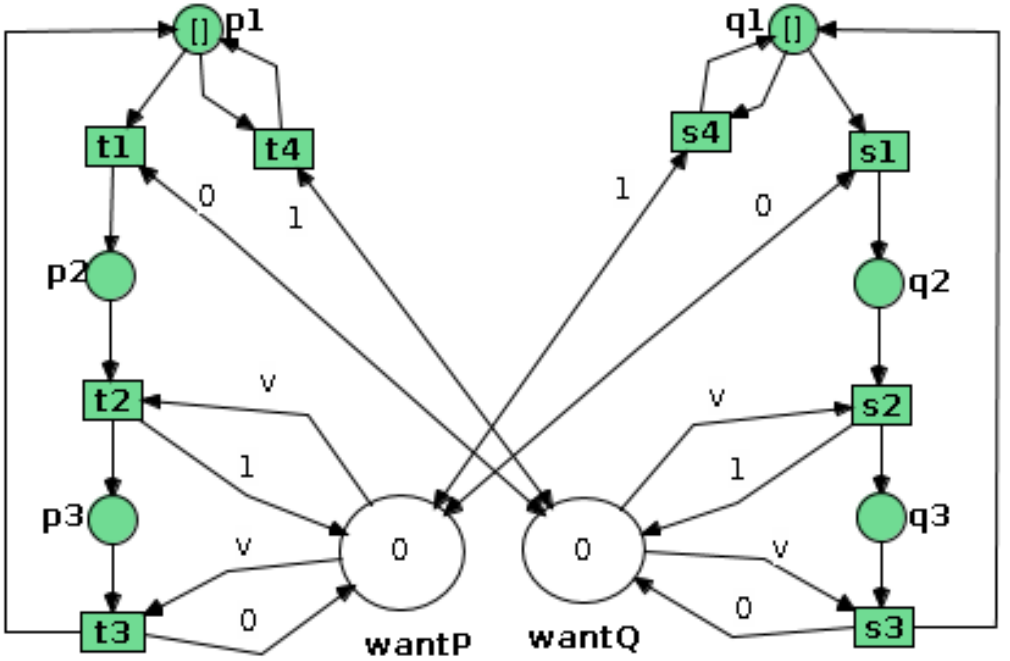
Modelar mediante Redes de Petri el siguiente programa concurrente y construir su grafo de estados alcanzables

```
integer v1 = 4, v2 = 0

process P1
   while(true)
      < await v1 >= 3     //transición t1
          v1 = v1-3
      >
      < v2 = v2+1 >       //transición t2
   end while
end process

process P2
   while(true)
      < await v1 >= 1     //transición t1P
          v1 = v1-1
      >
      < await v2 >= 1     //transición t2P
          v2 = v2-1
      >
      < v1 = v1+4 >       //transición t3P
   end while
end process
```

# Sesión 4: Programación concurrente

Obtener el grafo de estados alcanzables del siguiente modelo y responder a las siguientes cuestiones razonando sobre el grafo:

- ¿Tiene el programa un comportamiento equitativo? En caso negativo, evaluar el número de posibles historias no equitativas y mostrar al menos una.
- ¿Hay problemas de bloqueo?
- ¿Se cumple la exclusión mutua entre los estados de los procesos modelados, respectivamente, con los lugares p3 y q3?
- Probar que los estados modelados por los lugares p1, p2 y p3, están en exclusión mutua.


[]
p1
t1
t4
0
1
p2
t2
v
1
p3
v
t3
0
0
wantP
0
wantQ
q1
[]
s4
s1
1
0
q2
v
s2
1
q3
v
0
s3

# ncurrente

```
integer s := 2

Process P1
    while true
        <await s >= 1
           s := s-1
        >
        //en la zona crítica
        < s := s+1 >
    end
end

Process P2
    while true
        <await s >= 1
           s := s-1
        >
        //en la zona crítica
        < s := s+1 >
    end
end
```

```
Process P3
    while true
        <await s >= 2
           s := s-2
        >
        //en la zona crítica
        < s := s+2 >
    end
end

Process P4
    while true
        <await s >= 2
           s := s-2
        >
    end
end
```

# Sesión 4: Programación concurrente

Obtener el grafo de estados alcanzables del siguiente modelo y responder a las siguientes cuestiones razonando sobre el grafo:

- ¿Puede el programa llegar a bloquearse totalmente (es decir, llegar a un estado en el que ninguna instrucción se pueda ejecutar)? En caso afirmativo, hay que dar una secuencia de ejecución que lleve a bloqueo total. En caso negativo, hay que razonar por qué no puede bloquearse.
- ¿Existe alguna historia no equitativa (entendiendo que es una ejecución infinita en que a partir de un momento determinado algún proceso que desea intervenir puede no llegar a hacerlo)?
- ...

# Sesión 4: Programación concurrente

Obtener el grafo de estados alcanzables del siguiente modelo y responder a las siguientes cuestiones razonando sobre el grafo:

- ...
- Razónese sobre la corrección del invariante siguiente: "*Cuando P3 está en su zona crítica, P1 no puede estar en su zona crítica*".
- Razónese sobre la corrección de la siguiente propiedad: "*Exceptuando los posibles estados de bloqueo (si los hubiera) el estado inicial del sistema es recuperable desde cualquier estado posible del programa*".