Interfoliación de sentencias atómicas

Sea C un programa concurrente formado por dos procesos P_A y P_B.

Al ejecutar C, se pueden dar las siguientes secuencias:

P_A formado por $A_1 \dots A_5$

P_B formado por $B_1 \dots B_5$

Secuencias:
- $A_1 A_2 A_3 A_4 A_5 B_1 B_2 B_3 B_4 B_5$
- $A_1 B_1 A_2 B_2 \dots$
- NUNCA $A_2 A_1 B_2 B_1$

El orden debe respetarse siempre (de cada proceso)

$P \equiv P_1 \,||\, P_2$ int x = 0

P1

x = x + 1

↓

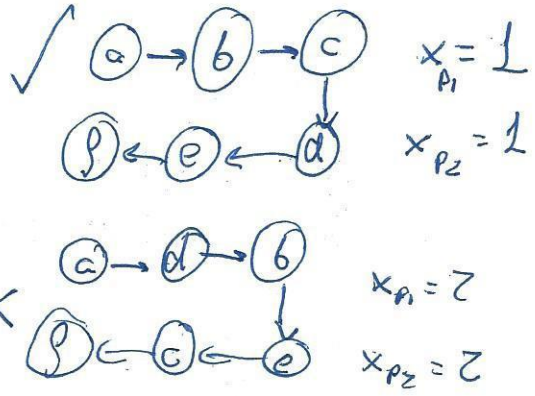
ⓐ load x, r
ⓑ add r, 1
ⓒ store r, x

P2

x = x + 1

↓

ⓓ load x, r
ⓔ add r, 1
ⓕ store r, x



- Abstracción: nos quedamos con los detalles importantes.

- El enlazamiento preserva la consistencia: el resultado de una instrucción individual no debe depender de las circunstancias de la ejecución.

- No se deben hacer suposiciones temporales, ya que dificulta la corrección del programa. Solo podemos asumir que la velocidad de ejecución del proceso es mayor a 0 (las instrucciones se ejecutarán en algún momento).

$V_{ej} > 0 \rightarrow$
- Punto de vista global: se permitirá la ejecución de al menos un proceso.
- Punto de vista local: al ejecutar una sentencia, esta finalizará en algún momento

- Traza de un programa concurrente: secuencia de estados producida en una secuencia concreta de interfoliación

$$s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_3 \ldots \rightarrow s_n$$

Ejecución concurrente

- Propuestas iniciales: no separan la definición de los procesos de su sincronización.
- Propuestas posteriores: separan conceptos y dan estructura.
- Declaración de procesos: rutinas de programación concurrente → Estructura de programa concurrente más clara.

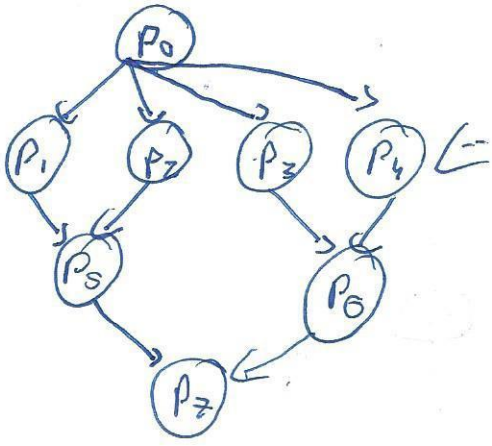
- Sistemas estáticos: los procesos se activan al lanzar el programa.
  Ejemplo: MP1-1
- Sistemas dinámicos: los procesos pueden activarse en tiempo de ejecución. Ejemplo: MP1-2

Grafo de sincronización

- Cada nodo representa una secuencia de sentencias.

→ Una flecha de A a B significa que B no se ejecuta hasta que termine A

→ Grado: máximo número de actividades que se ejecutan a la vez

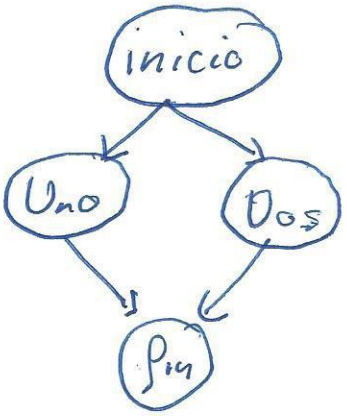


Definición estática de procesos

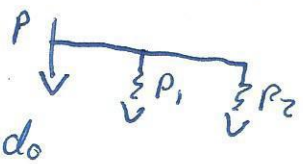
Palabra clave process

.....

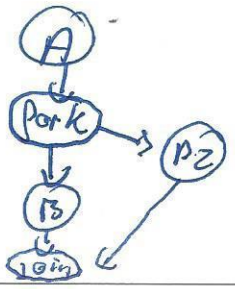
```
var ....   //Variables compartidas

process Uno;
 var ....   //Variables locales
begin
 ...
 ...
end

process Dos;
 var ...   //Locales
begin
 ...
 ...
end
```



- fork...: lanza un flujo que ejecuta el proceso indicado.
- join ...: espera a que termine el proceso indicado




```
procedure P1;
begin
   A:
   fork P2;
   B:
   join P2
end
```

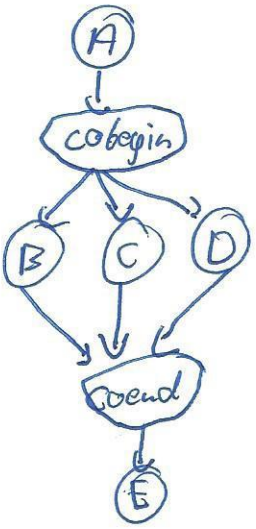


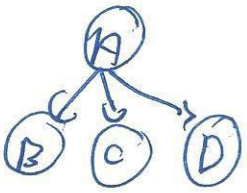
Fork y join son muy potentes, pero dificultan la comprensión de los programas.

Solución: cobegin y coend.

- cobegin: inicia todas las sentencias a la vez.
- coend: espera a que terminen todas las sentencias

```
begin
   A;
   cobegin
      B; C; D;
   coend
   E;
end
```





Ejercicio 3)

a) cobegin - coend

```
procedure P
begin
   P0;
   cobegin
      begin
         P1;
         P3;
         cobegin
            P4, P5
         coend
      end
      P2;
   coend
P6
```

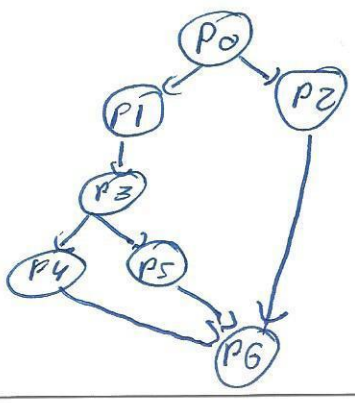
Posibles secuencias

P0 → P1 → P3 → P4 → P5 → P2 → P6

P0 → P1 → P3 → P5 → P4 → P2 → P6

P0 → P1 → P2 → P3 → P4 → P5 → P6

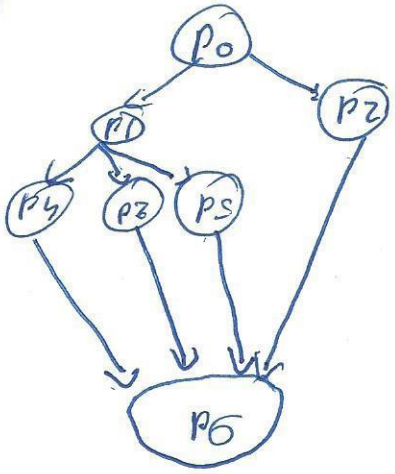
...



fork-join

```
procedure P;
begin
    P0;
    fork P2
     P1;
     P3;
     fork P4;
      P5;
     join P4;
      join P2;
      P6;
end
```

b)



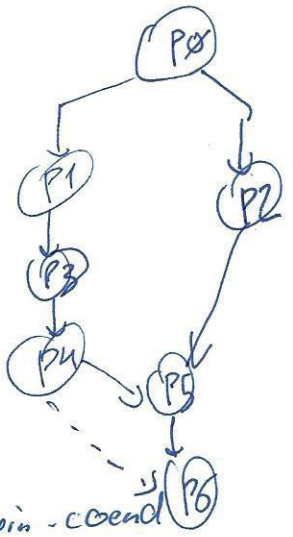
cobegin-coend

```
Procedure P;
  begin
    P0;
    cobegin;
      begin
        P1;
        cobegin
          P4, P3, P5;
        coend;
      end
        P2;
    coend
    P6;
  end;
```

fork - join

```
Procedure P.
begin
    P0;
    fork P2;
    P1;
    fork P4, P5;
    P3;
    join P4, P5;
    join P2;
    P6;
end;
```

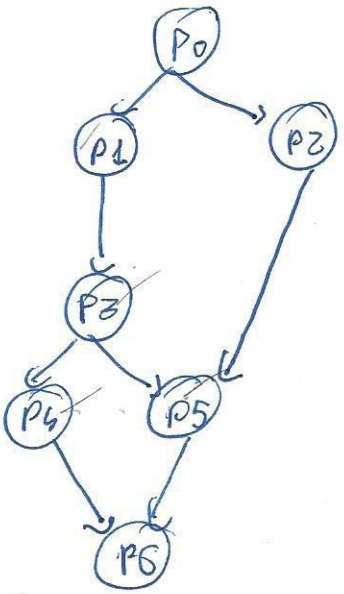
* Ir por un camino secuencial hasta el final y abrir ramas cuando sea necesario *



cobegin - coend

```
Procedure P;
begin
    P0;
    cobegin
        begin
            P1;
            P3;
            P4;
        end
        P2
    coend
    P5;
    P6;
end
```

c)



~~cobegin - coend~~

fork - join

```
Procedure P;
begin
    P0;
    P1;
    fork P2;
    P3;
    fork P4;
    join P2;
```

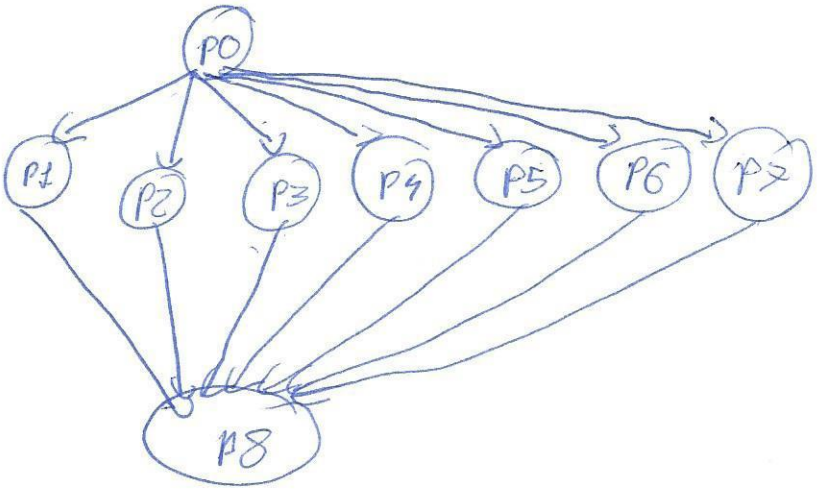
```
P5;
join P4;
P6;
end;
```

En programas con procesos cooperativos (no independientes entre sí) hay secuencias de interfoliación no válidas.

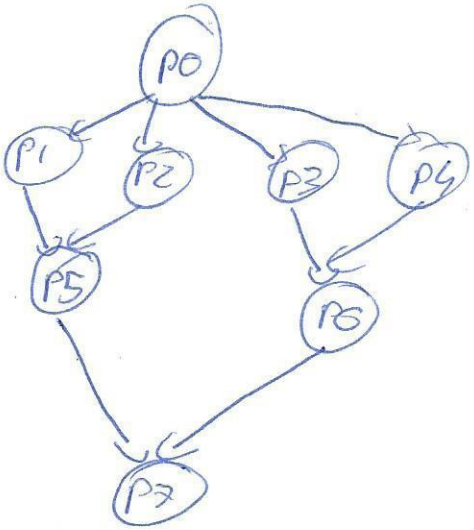
- Condición de sincronización: se impone una restricción en la secuencia de interfoliación
- Exclusión mutua: secuencias finitas que deben ejecutarse completamente por un único proceso, sin que otro proceso pueda ejecutarlas también.

4) Obtener grafos

```
begin
    P0;
    cobegin
        P1;
        P2;
        cobegin
            P3; P4; P5; P6;
        coend
        P7;
    coend;
    P8;
end
```




```
begin
    P0;
    cobegin
        begin
            cobegin
                P1; P2;
            coend
            P5;
        end
        begin
            cobegin P3; P4;
            coend
            P6;
        end
        .
        .
        .
        coend
        P7;
    end
```



1)

{variables compartidas}

var x: integer := 0

```
process P1;
  var i: integer;
begin
  for i:=1 to 2 do begin
    x := x+1;
  end
end
```

```
process P2;
  var j: integer;
begin
  for j:=1 to 2 do begin
    x := x+1;
  end
end
```

{x, r, r1}

P1

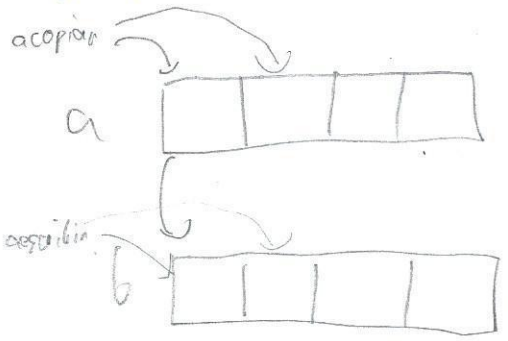
(1)	load	x, r	{0,0,0}
(2)	add	r, 1	{0,1,0}
(5)	store	r, x	{1,1,1}
(7)	load	x, r	{1,1,1} {1,1,1}
(8)	add	r, 1	{1,2,1} {1,2,1}
(9)	store	r, x	{2,2,2} {2,2,2}

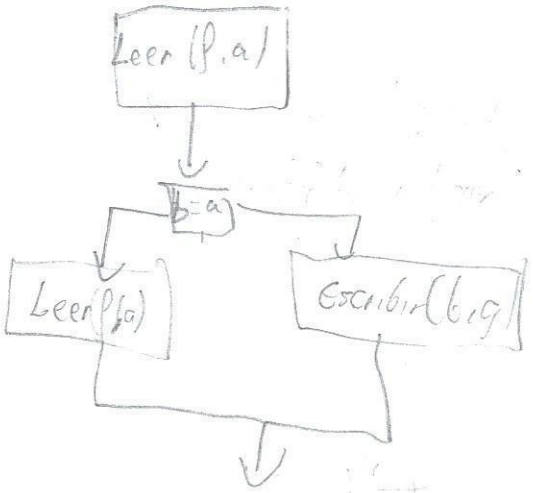
P2

(3)	load	x, r1	{0,1,0}
(4)	add	r1, 1	{0,1,1}
(6)	store	r1, x	{1,1,1}
(10)	load	x, r1	{1,1,1} {2,2,2}
(11)	add	r1, 1	{1,2,2} {2,2,3}
(12)	store	r1, x	{2,2,2} {3,2,3}

Posibles valores de x = {2, 3, 4}

2) Copia de archivo concurrente






```
var a, b
begin
  a = leer (f)
  while ( ! fin(f) )
    begin
      b = a
      cobegin
        escribir (b, q)
        a = leer (f)
      coend
    end
end
```

5)

① 1, 2, 3

Estado inicial
n=k out=?

1:- n=k+1, out=?

2:- n=k+1, out=k+1 ✓

3:- n=0, out=k+1

② 2, 1, 3

1:- n=k, out=k

2:- n=k+1, out=k ✗

3:- n=0, out=k

Se pierde uno

③

1:- n=k, out=k

2:- n=0, out=k

3:- n=1, out=k ✓

8)

var acabado [9]: boolean. [false]

```
procedure EsperarPor(int i)
  begin
    while ! acabado [i] do
    end
  end
```

```
procedure Acaba (int i)
  begin
    acabado [i] := true
    if (i == 9) then
      for (i=0; i<9; i++)
        acabado [i] = false
    end
  end
```