Lección 15: Programación concurrente mediante paso síncrono de mensajes

- Limitaciones del paso asíncrono de mensajes
- Paso síncrono de mensajes:
  - notación simplificada
  - ejemplo: proceso filtro “copiar”
  - notación general
  - ejemplo: proceso servidor “mcd”
- Comunicación síncrona con guardas
- Ejemplos
  - un proceso-filtro
  - la criba de Eratóstenes
  - producto matriz por vector

Limitaciones del paso asíncrono de mensajes

- "***send***" asíncrono es no bloqueante, por lo que el número de mensajes en un canal puede ser no limitado
  - el que envía puede avanzar mucho antes de que nadie lo reciba, lo que puede obligarle a esperar
  - en caso de pérdida de mensajes, el que envía no se entera, lo que puede bloquearle en el futuro
  - los mensajes requieren un buffer, lo que:
    - puede generar desbordamiento
    - puede hacer que *send* sea bloqueante, lo que no respeta su semántica
- la comunicación síncrona evita estos problemas haciendo que ***send*** y ***receive*** sean bloqueantes

Notación para el paso síncrono de mensajes

- Concepto y notación usados por Hoare (1978)
  - Communicating Sequential Processes (CSP)
- La comunicación entre procesos se va a llevar a cabo mediante *puertos*

```
origen::
   ...
   destino!p(e1...en)
   ...
```

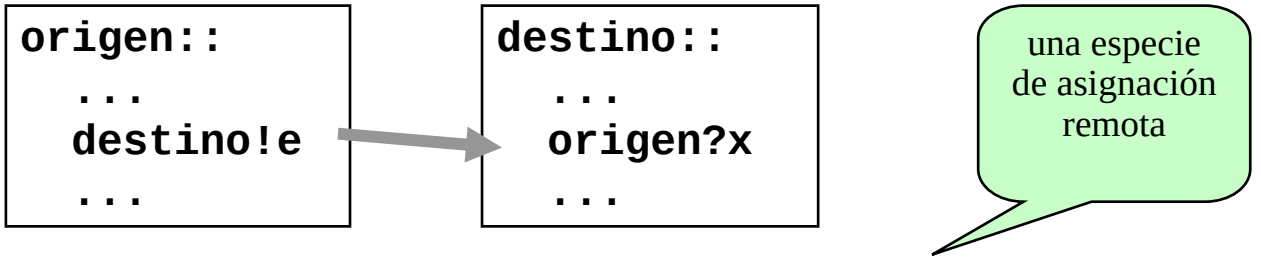
el proceso "**origen**" usa el puerto "**p**" para pasar las expresiones "**e_1...e_n**" al proceso "**destino**". Sentencia de *salida*

```
destino::
   ...
   origen?p(x1...xn)
   ...
```

el proceso "**destino**" usa el puerto "p" para asignar a "**x_1...x_n**" las expresiones "**e_1...e_n**" enviadas por "**origen**". Sentencia de *entrada*

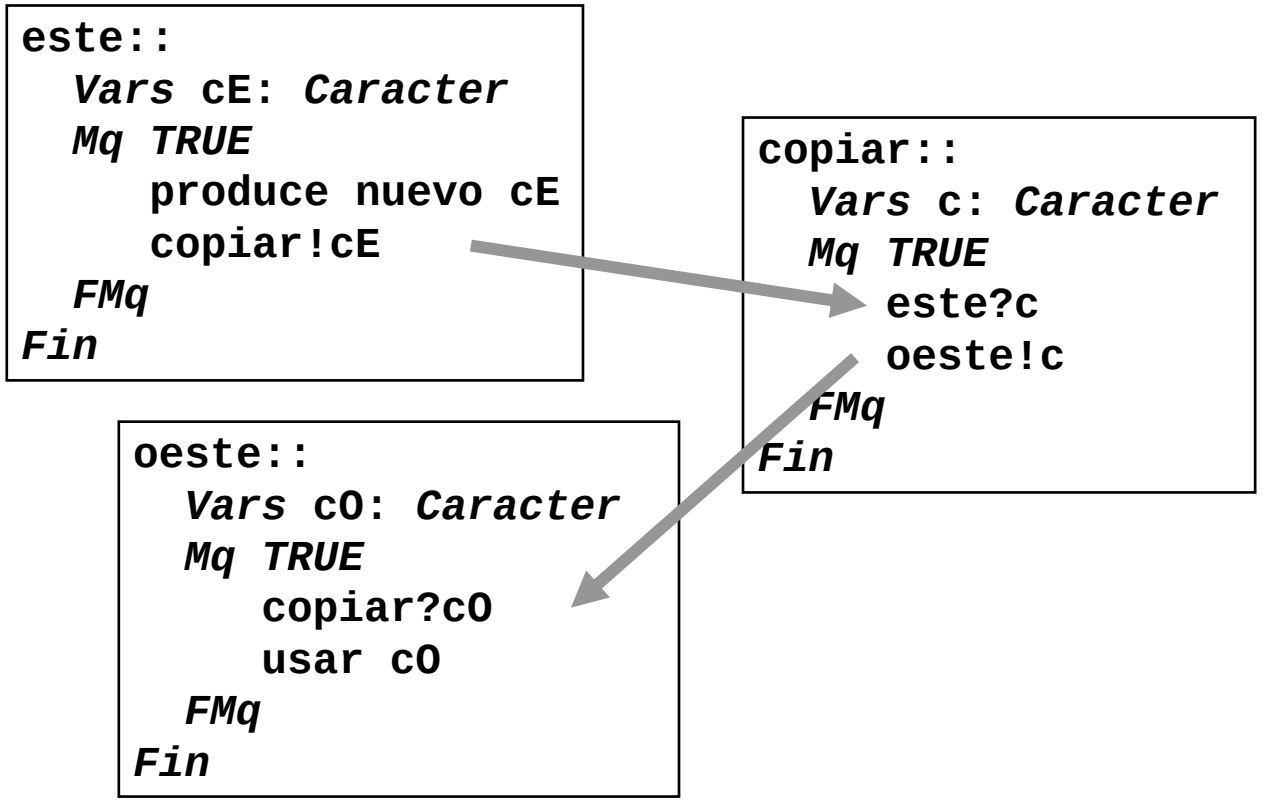
Notación para el paso síncrono de mensajes

- Los puertos se usan para distinguir tipos de mensajes que un proceso puede enviar/recibir
- Si no se quieren diferenciar los tipos de mensaje, no es necesario indicarlo:



- Por simplicidad, ni siquiera los declararemos explícitamente
- Cuando un proceso llega a una salida y otro a una entrada, se dice que las sentencias de comunicación "***se emparejan***"
  - es necesario que las "x_i" y las "e_i" sean de tipos adecuados

Un ejemplo muy sencillo



Otro ejemplo muy sencillo

```
mcd::
  Vars x,y: Ent
       r: Ent := 1
  Mq TRUE
    cliente?args(x,y)
    Mq r<>0
      ...
    FMq
    cliente!sol(x)
  FMq
Fin
```

```
cliente::
  Vars a,b,m: Ent

  Mq ...
    obtener a,b
    mcd!args(a,b)
    mcd?sol(m)
    ...
  FMq
Fin
```

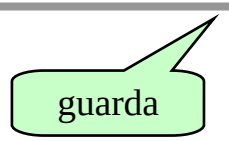
- Alternativamente

```
...
cliente?(x,y)
...
```

```
...
mcd!(a,b)
...
```

Comunición síncrona con guardas

- Sintaxis: `condición;comunicación: instrucción`



- Ejemplo:

```
x<0;destino!e: x:=y
```

- La guarda puede:
  - *estar abierta*: "condición" es True y "comunicación" lista
  - *estar cerrada*: "condición" es False
  - *estar bloqueada*: guarda satisfecha pero "comunicación" tiene que esperar

Comunición síncrona con guardas

- La comunicación síncrona con guardas aparecerá en instrucciones
  - "*Sel ... FSel*"
    - si al menos una guarda está abierta, de manera indeterminista, se ejecuta primero la comunicación y a continuación la instrucción
    - si todas cerradas, la selección termina
    - si ninguna abierta pero alguna bloqueada, el proceso queda bloqueado hasta que alguna guarda esté abierta
      - procesos no comparten variables, por lo que el proceso queda bloqueado hasta que algún otro proceso llegue a emparejar alguna comunicación con éste
  - "*Mq ... FMq*"
    - la versión iterativa de lo anterior

Comunición síncrona con guardas

OjO: sólo es una notación compacta

- No es lo mismo

```
Sel
    ...
    Bi;Ci: Si
    ...
FSel
```

```
Sel
    ...
    Bi: Ci;Si
    ...
FSel
```

- Si es lo mismo

```
Mq
    ...
    Bi;Ci: Si
    ...
FMq
```

```
Mq B1 ∨ ... ∨ Bn
    Sel
        ...
        Bi;Ci: Si
        ...
    FSel
FMq
```

- Si es lo mismo

```
Para
  (i:1..n)
       P(i)?d(v): ...
   ...
FPara
```

```
Sel
    True;P(1)?d(v) ...
    True;P(2)?d(v) ...
    ...
    True;P(n)?d(v) ...
    ...
FSel
```

Ejemplo: un proceso-filtro

- Proceso-filtro que copia datos recibidos desde un proceso ”fuente" enviándolos a un proceso “destino”, manteniendo un almacén de “n” caracteres

```
copiar::
  Consts  n=10
  Vars  alm: Vector(1..n) De Car
        nD: Ent := 0; sig: Ent := 1
        pri: Ent := 1
  Mq (nD < n)∨(nD > 0)
    Sel
      nD<10;fuente?alm(sig):
        nD := nD+1
        sig := (sig mod n)+1
      nD>0;destino!alm(pri):
        nD := nD-1
        pri := (pri mod n)+1
    FSel
  FMq
```

Ejemplo: un proceso-filtro

- Proceso-filtro que copia datos recibidos desde un proceso ”fuente" enviándolos a un proceso ”destino", manteniendo un almacén de “n” caracteres

```
copiar::
  Consts  n=10
  Vars  alm: Vector(1..n) De Car
        nD: Ent := 0; sig: Ent := 1
        pri: Ent := 1
  Mq
      nD<10;fuente?alm(sig):
        nD := nD+1
        sig := (sig mod n)+1
      nD>0;destino!alm(pri):
        nD := nD-1
        pri := (pri mod n)+1
  FMq
```

Semántica formal

- **Considerar el entorno:**

$--Q_1$ destino!p($e_1 \ldots e_n$) $--R_1$	$--Q_2$ fuente?p($x_1 \ldots x_n$) $--R_2$

- **Regla de satisfacción para la *comunicacion síncrona***: para cada posible emparejamiento, probar

$$Q_1 \wedge Q_2 \rightarrow (R_1 \wedge R_2)^{e_1 \ldots e_n}_{x_1 \ldots x_n}$$

Semántica formal

- Regla para la comunicación en selección:

```
--Q
Sel
   B1;C1:S1
   ...
   Bn;Cn:Sn
FSel
--R
```

$$\frac{Q \wedge \neg(B_1 \vee \ldots \vee B_n) \rightarrow R \qquad \forall i \in \{1..n\}.\{Q \wedge B_i\}\ C_i;S_i\ \{R\}}{\{Q\}\ Sel...FSel\ \{R\}}$$

Semántica formal

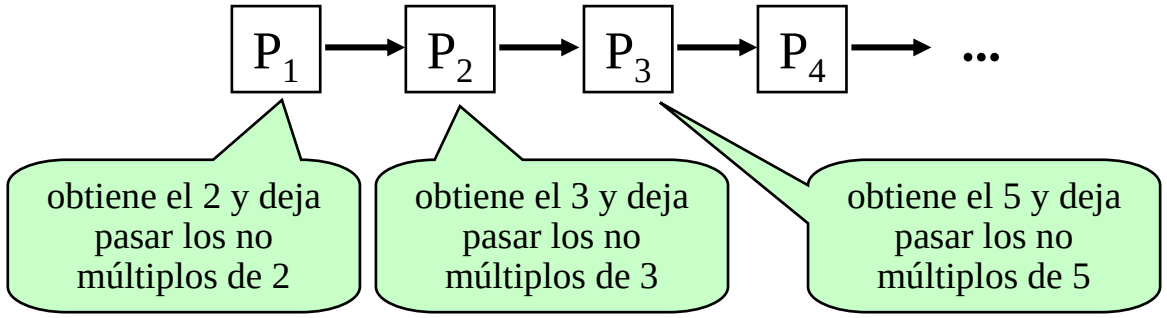
- Regla para la comunicación en iteración:

```
--Q
Mq
    B1;C1:S1
    ...
    Bn;Cn:Sn
FMq
--R
```

$$\frac{\forall i \in \{1..n\}.\{I \wedge B_i\}\ C_i;S_i\ \{I\}}{\{I\}\ Mq...FMq\ \{I \wedge \neg(B_1 \vee ... \vee B_n)}$$

Ejemplo: la criba de Eratóstenes

- **La criba de Eratóstenes**: encuentra los primos menores o iguales que un *n* dado
- Secuencial sencillo:
  - el 2 es primo; "tachar" su múltiplos.
  - el primero no "tachado" (el 3) es primo; tachar sus múltiplos
  - el primero no "tachado" (el 5) es primo; tachar sus múltiplos. etc.
- Una versión en "pipe-line":



Ejemplo: la criba de Eratóstenes

```
criba(1)::
  Vars p: Ent:=2
       j: Ent:=3

  escribe(p)
  Mq j<=n
    criba(2)!j
    j:=j+2
  FMq
```

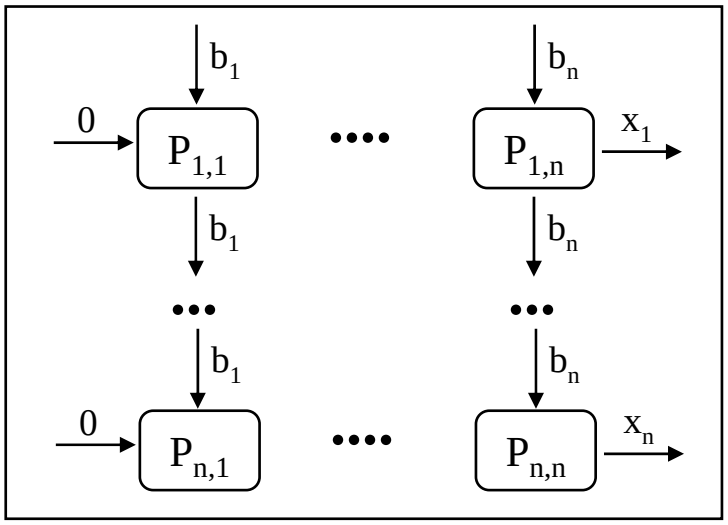
deja pasar los no múltiplos de 2

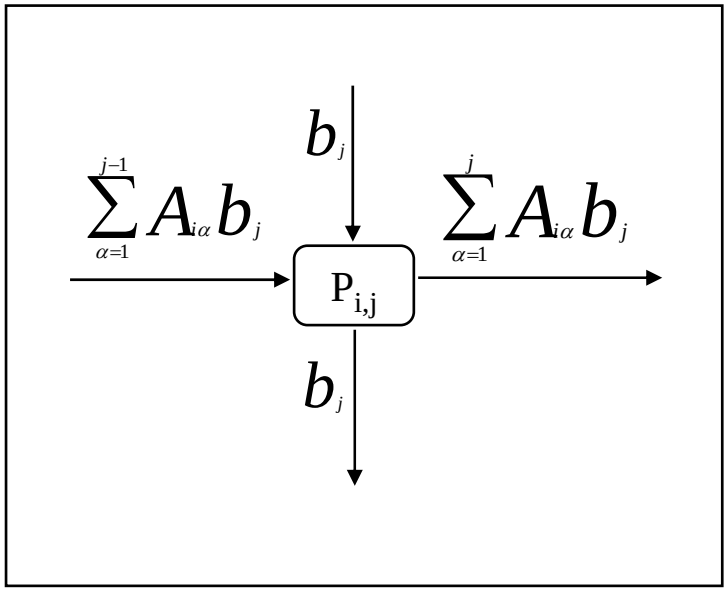
```
criba(i:2..L)::
  Vars p: Ent; sig: Ent

  criba(i-1)?p --p es primo
  escribe(p)
  Mq True
    criba(i-1)?sig
    Si (sig MOD p) <> 0 Ent
                --no múltiplo de p
      criba(i+1)!sig
    FSi
  FMq
```

deja pasar los no múltiplos de p

Ejemplo: producto matriz por vector

- **Problema**: calcular "**x=Ab**" siendo "**A**" una matriz nxn y "**x**" y "**b**" vectores nx1
- Una solución síncrona, usando nxn procesadores:





Ejemplo: producto matriz por vector

- Inconvenientes:
  - cada procesador hace poco cálculo y mucha comunicación
  - poco eficiente
- Alternativas:
  - partir A en bloques
  - cada bloque hace más operaciones
  - pero también envía/recibe más datos

```
P(i:1..n;j:1..n)::
  Vars suma: Real := 0
       b_j: Real := 0

  P(i-1,j)?b_j
  P(i+1,j)!b_j
  P(i,j-1)?suma

P(i,j+1)!(suma+a_i_j*b_j)
```

adaptar para los procesos de la frontera

Ejemplo: servidor de recursos compartidos

- **Problema**: implementar un servidor de recursos que atiende peticiones de sus clientes para adquirir y liberar recursos.
- Todos los recursos son idénticos, aunque cada uno tiene su propio identificador.

Ejemplo: servidor de recursos compartidos

```
Cliente(i:1..n)::
  Vars idRecurso: Ent
  ...
  acciones  previas
  servidor!pedir()        --petición de recurso
  servidor?concedido(idRecurso)  --esperando respuesta
  ...                              --uso del recurso
  servidor!dejar(idRecurso) --libera el recurso
  acciones  posteriores
```

Ejemplo: servidor de recursos compartidos

```
ServidorDeRecursos::
  Vars disponibles: Ent:=MAX
       recursos: Conjunto De Ent := {...}
       idRecurso: Ent
  acciones de inicialización
  Mq Verdad
    Para
       (i:1..n)
            disponibles>0;Cliente(i)?pedir():
                disponibles:=disponibles-1
                idRecurso:=tomar(recursos)   --uno
                Cliente(i)!concedido(idRecurso)
       (i:1..n)
            TRUE;Cliente(i)?dejar(idRecurso):
                disponibles:=disponibles+1
                añadir(recursos,idRecurso)
    FPara
  FMq
```

Mejoras:
* **generalizar**
* **atender por orden de petición**
* **hacerlo equitativo**

Ejercicio

- Anotar y verificar el siguiente programa:

```
Q1::
  Vars b: Ent
Principio
  P?e(b)
  P!e(b*b)
Fin
```

```
Q2::
  Vars b: Ent
Principio
  P?e(b)
  P!e(3*b)
Fin
```

```
P::
  Vars a: Ent := A
       r: Ent
Principio
       --a=A
  Q1!e(a)
  Q2!e(a)
  Sel
    a<0; Q1?e(r):  seguir
    a>=0; Q2?e(r): seguir
  FMq
       -- a<0 --> r=A² ∧
       -- a>=0 --> r=3A

Fin
```