Lección 5:
Técnicas para asegurar ausencia de interferencias

- Uso de variables disjuntas
- Debilitamiento de aserciones
- Uso de invariantes globales
- Sobre el uso de variables auxiliares
- Cómo imponer ausencia de interferencias

Uso de variables disjuntas

- Considerar las "$\{Q_i\}\ S_i\ \{R_i\}$", $i \in \{1..n\}$
  - para cada $i \in \{1..n\}$ definimos

CA_i={variables compartidas a las que se asigna en S_i}

CC_i={variables compartidas en alguna a.c. de $\{Q_i\}S\{R_i\}$}

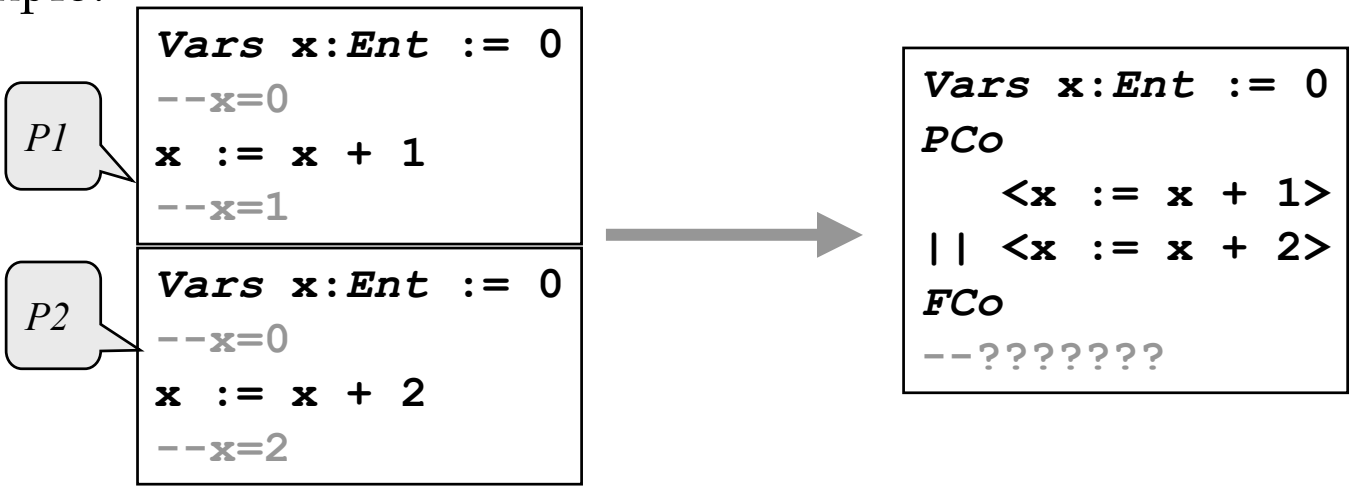
- Si, $\forall i,j \in \{1..n\}.\ i \neq j \rightarrow CA_i \cap CC_j = \varnothing$ entonces L.I.

- Probar la corrección de

```
Vars x:Ent := 0
     y:Ent := 0
-- x=0 ∧ y=0
PCo <x:=x+1> || <y:=y+1> FCo
-- ¿x=1 ∧ y=1?
```

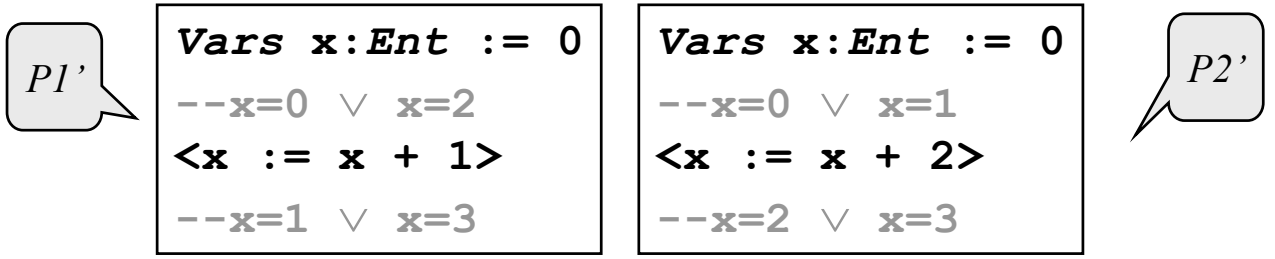
Debilitamiento de aserciones

- Trata de hacer que cada proceso, en su verificación, tenga en cuenta las posibles interferencias de los otros procesos
- Para ello, debilitamos aserciones
  - forma natural: añadir disjunciones
- Ejemplo:



Debilitamiento de aserciones

- P1 puede tratar de prever que, al empezar, posiblemente P2 ya haya hecho la asignación, y viceversa



- Fácil comprobar que sus verificaciones no interfieren
- Cuestión:
  - debilitar lo suficiente como para asegurar no interferencias
  - debilitar lo mínimo como para poder probar algo "interesante"

Uso de invariantes globales

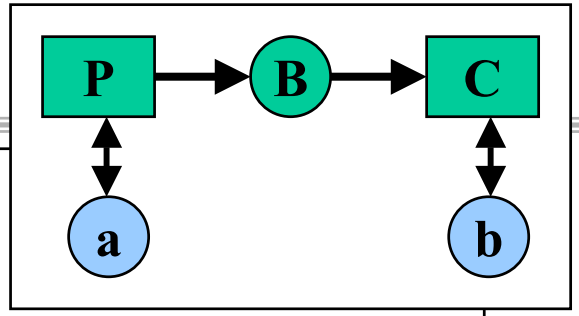
- **Invariante global**: una aserción I relativa a las variables globales lo es cuando
  - I se cumple antes de que los procesos empiecen a ejecutarse
  - I es invariante a cada ejecución de asignación de cada proceso
    - asignación directa
    - instrucción "await" con una asignación en su interior
- Para verificar que una aserción I es inv. global:
  - probar que se cumple antes de cada proceso
  - cada asignación A de cada proceso cumple

$$\{pre(A) \wedge I\}\ \mathbf{A}\ \{I\}$$

Uso de invariantes globales

- Uso de un invariante para probar la ausencia de interferencias
  - si cada aserción crítica C se puede escribir como $C = I \wedge L_c$
    - I: invariante global
    - L_c: se refiere a variables locales o globales sólo modificadas por este proceso
  - entonces C no puede ser interferida
- ¿Cómo usarlo en la verificación del programa concurrente?

Un ejemplo




```
Consts n=.....
Vars buf:Ent
     p:Ent := 0
     c:Ent := 0
     a:vector(1..n)de Ent:=A(1..n)
     b:vector(1..n)de Ent
--a(1..n)=A(1..n)
```

```
Productor::
  Mq p<n
    <await (p=c)>
    buf:=a(p+1)
    p:=p+1
  FMq
```

```
Consumidor::
  Mq c<n
    <await (p>c)>
    b(c+1):=buf
    c:=c+1
  FMq
```

```
--b(1..n)=A(1..n)
```

Un ejemplo

```
--I: (0<=p<=n)∧(0<=c<=n)∧(c<=p<=c+1)∧
     (p=c+1→buf=A(p))∧(b(1..c)=A(1..c))∧a=A
```

```
Productor::
--I
Mq p<n
  --P1:I∧(p<n)
  <await (p=c)>
  --P2: I∧(p<n)∧(p=c)
  buf := a(p+1)
  --P3: I∧(p<n)∧(p=c)∧
  --      (buf=a(p+1))
  p := p+1
  --I
FMq
--I∧(p=n)
```

```
Consumidor::
--I
Mq c<n
    --C1: I∧(c<n)
    <await (p>c)>
    --C2: I∧(c<n)∧(p>c)
    b(c+1) := buf
    --C3: I∧(c<n)∧(p>c)∧
    --      (b(c+1)=buf)
    c := c+1
    --I
FMq
--I∧(c=n)
```

Sobre el uso de variables auxiliares

- La regla de concurrencia no logra que nuestra lógica sea completa
  - algunos teoremas válidos no pueden ser probados
- Ejemplo: lo siguiente no puede ser verificado

```
{x=X} PCo <x:=x+1> || <x:=x+1> FCo {x=X+2}
```

- Razón: necesitamos saber dónde se encuentra el otro proceso
  - "valor de su contador de programa"
- Solución: introducir **variables auxiliares**
  - "espias", "monitores", ...

Sobre el uso de variables auxiliares

- Una solución:

```
--x=X
t1:=0;t2:=0
    --I:x=t1+t2+X
PCo
    --I∧t1=0
    <x:=x+1;t1:=1>
    --I∧t1=1
|| --I∧t2=0
    <x:=x+1;t2:=1>
    --I∧t2=1
FCo
--x=X+2
```

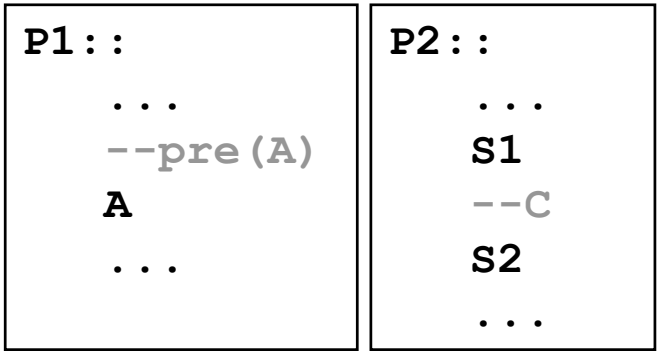
- Variables auxiliares: sólo pueden aparecer en asignaciones "**x:=E**", donde a su vez "**x**" es auxiliar
  - no aparecen en asignaciones a variables de programa
  - no aparecen en guardas
- Regla de las vars. auxiliares:

$$\frac{\{Q\}\ S'\ \{R\}}{\{Q\}\ S\ \{R\}}$$

• "Q", "R" no referencian auxiliares
• "S" es "S'" sin asignaciones a auxiliares

Cómo imponer ausencia de interferencias

- Hemos visto formas de asegurar la no interferencia
- Ahora vamos a imponerla
- Considerar el entorno con interferencia entre A y **C**.
- Dos maneras de evitar el problema:
  - establecer una exclusión mutua:
    - cambiar **S1;S2** por **<S1;S2>**
  - establecer una sinc. por condición
    - cambiar **A** por **<await (¬C ∨ pmd(A,C)) A>**



Un ejemplo (con una mala solución)

```
Consts n=.....
       TEOR=.....
Vars acc:vector(1..n)
                  de Ent
     x,y: 1..n
     emb:Bool:=FALSE
```

```
Transfer::
x := .... ; y := ....
--acc(x)=X ∧
--acc(y)=Y
acc(x):=acc(x)-100
acc(y):=acc(y)+100
--acc(x)=X-100 ∧
--acc(y)=Y+100
```

```
Auditor::
Vars total:Ent:=0
     i:Ent:=1
 --I:total=acc(1)+...+acc(i-1)∧
 --     1<= i<=n+1
Mq i<=n
 --I∧i<=n
 total:=total+acc(i)
 --total=acc(1)+...+acc(i) ∧
 --     1<= i<=n
 i := i+1
 --I
FMq
--I∧i=n+1
emb:= (total<>TEOR)
--emb=(TEOR<>acc(1)+...+acc(n))
```

Un ejemplo (con una buena solución)

```
Transfer::
x := .... ; y := ....
--acc(x)=X∧acc(y)=Y
<await (x<i ∧ y<i) ∨
       (x>i ∧ y>i)
  acc(x):=acc(x)-100
  acc(y):=acc(y)+100
>
--acc(x)=X-100∧
--acc(y)=Y+100
```

```
Auditor::
Vars total:Ent:=0
     i:Ent:=1
 --I:total=acc(1)+...+acc(i-1)∧
 --    1<= i<=n+1
Mq i<=n
 --I∧i<=n
 total:=total+acc(i)
 --total=acc(1)+...+acc(i) ∧
 --    1<= i<=n
 i := i+1
 --I
FMq
--I∧i=n+1
emb:= (total<>TEOR)
--emb=(TEOR<>acc(1)+...+acc(n))
```