Semántica de ejecución concurrente:
Los comandos **`await`** y **`co`**

Un programa concurrente consiste en múltiples procesos ejecutando concurrentemente

La ejecución concurrente de *n* (listas de) comandos S_i se especifica mediante el comando **`co`**:

$$\texttt{co } S_1\texttt{; || } S_2\texttt{; || … || } S_n\texttt{; oc;}$$

… en que cada (lista de) comando(s) S_i satisface el triple

$$\{ P_i \}\ S_i\ \{ Q_i \}$$

Semántica de ejecución concurrente: Los comandos **await** y **co**

Un programa concurrente consiste en múltiples procesos ejecutando concurrentemente

La ejecución concurrente de *n* (listas de) comandos S_i se especifica mediante el comando **co**:

```
co S1; || S2; || … || Sn; oc;
```

… en que cada (lista de) comando(s) S_i satisface el triple

$$\{ P_i \}\ S_i\ \{ Q_i \}$$

Empiece a ejecutar cada comando S_i en paralelo … luego, espere a que terminen

co termina una vez que todos los comandos han sido ejecutados

Operador de paralelismo

Si S_i empieza en un estado que satisface P_i y termina, entonces el estado satisfará Q_i

Para que la interpretación anterior (“Si S_i empieza en … ”) sea válida cuando los procesos son ejecutados concurrentemente

… los procesos deben partir en un estado que satisfaga la conjunción de los P_i

Si todos los procesos terminan, entonces el estado final satisfará la conjunción de los Q_i

Regla del comando **co**

$$\frac{\{P_i\}\ S_i\ \{Q_i\}\ \text{are interference free}}{\begin{array}{l}\{P_1 \wedge \ldots \wedge P_n\} \\ \textbf{co}\ S_1; \ // \ \ldots \ // \ S_n; \ \textbf{oc} \\ \{Q_1 \wedge \ldots \wedge Q_n\}\end{array}}$$

Para que la interpretación anterior ("Si S_i empieza en ... ") sea válida cuando los procesos son ejecutados concurrentemente

... los procesos deben partir en un estado que satisfaga la conjunción de los P_i

Si todos los procesos terminan, entonces el estado final satisfará la conjunción de los Q_i

Regla del comando **co**

$$\frac{\{P_i\}\ S_i\ \{Q_i\}\ \text{are interference free}}{\{P_1 \wedge \ldots \wedge P_n\}\ \texttt{co}\ S_1; \ //\ \ldots\ //\ S_n;\ \texttt{oc}\ \{Q_1 \wedge \ldots \wedge Q_n\}}$$

Los procesos y sus triples *no deben interferir* entre ellos

Un proceso *interfiere* con otro si ejecuta una asignación que invalida una aseveración en el otro proceso

Regla del comando **co**

$$\frac{\{P_i\}\ S_i\ \{Q_i\}\ \text{are interference free}}{\{P_1 \wedge \ldots \wedge P_n\}\ \textbf{co}\ S_1; \ //\ \ldots\ //\ S_n;\ \textbf{oc}\ \{Q_1 \wedge \ldots \wedge Q_n\}}$$

Los procesos y sus triples *no deben interferir* entre ellos

Un proceso *interfiere* con otro si ejecuta una asignación que invalida una aseveración en el otro proceso:

- las aseveraciones caracterizan lo que un proceso supone que es verdadero antes y después de cada comando
- si un proceso asigna a una variable compartida y consecuentemente invalida una suposición de otro proceso, entonces el triple del otro proceso no es válido

Regla del comando **co**

$$\frac{\{P_i\}\ S_i\ \{Q_i\}\ \text{are interference free}}{\{P_1 \wedge \ldots \wedge P_n\}\ \textbf{co}\ S_1; \ //\ \ldots\ //\ S_n;\ \textbf{oc}\ \{Q_1 \wedge \ldots \wedge Q_n\}}$$

Los procesos y sus triples *no deben interferir* entre ellos

Un proceso *interfiere* con otro si ejecuta una asignación que invalida una aseveración en el otro proceso:

- las aseveraciones caracterizan lo que un proceso supone que es verdadero antes y después de cada comando
- si un proceso asigna a una variable compartida y consecuentemente invalida una suposición de otro proceso, entonces el triple del otro proceso no es válido

```
{x == 0}
co ⟨x = x+1;⟩ // ⟨x = x+2;⟩ oc
{x == 3}
```

Un proceso | Otro proceso

E.g., ningún proceso puede suponer que x es 0 cuando empieza a ejeuctar
… si lo hace, esa aseveración se verá interferida si el otro proceso ejecuta primero

Regla del comando **co**

$$\frac{\{P_i\}\ S_i\ \{Q_i\}\ \text{are interference free}}{\{P_1 \wedge \ldots \wedge P_n\}\ \textbf{co}\ S_1; \ //\ \ldots\ //\ S_n;\ \textbf{oc}\ \{Q_1 \wedge \ldots \wedge Q_n\}}$$

Los procesos y sus triples *no deben interferir* entre ellos

```
{x == 0}
co {x == 0 ∨ x == 2}
   ⟨x = x+1;⟩
   {x == 1 ∨ x == 3}
// {x == 0 ∨ x == 1}
   ⟨x = x+2;⟩
   {x == 2 ∨ x == 3}
oc
{x == 3}
```

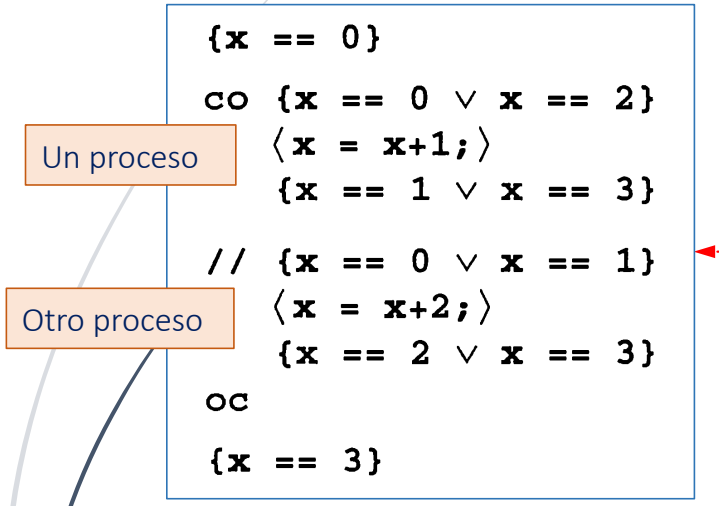
Un proceso

Otro proceso

Las aseveraciones en cada proceso toman en cuenta los dos posibles órdenes de ejecución

La conjunción de las precondiciones es **x** = 0

… y la de las postcondiciones, **x** = 3



Ej. de un *esbozo de demostración*:

- hay una aseveración antes y después de cada comando
- todos los triples reultantes son verdaderos
- hay tres triples: uno para cada proceso y uno para el **co**

Definición de **No interferencia**:

- Sea ***a*** una *acción de asignación* en un proceso

... y ***pre***(***a***) su precondición

- Sea ***C*** una *aseveración crítica* en otro proceso

Entonces ***a*** **no interfiere** con ***C*** si

$$\{ C \wedge pre(a) \}\ a\ \{ C \}$$

... es un teorema

Definición de **No interferencia**:

- Sea ***a*** una *acción de asignación* en un proceso

... y ***pre***(***a***) su precondición

- Sea ***C*** una *aseveración crítica* en otro proceso

Entonces ***a*** **no interfiere** con ***C*** si

$$\{ C \wedge pre(a) \}\ a\ \{ C \}$$

... es un teorema

Acción de asignación:

- comando de asignación
- ... o comando **`await`** que contiene una o más asignaciones

Aseveración crítica:

- precondición o poscondición que no está dentro de un **`await`**

Definición de **No interferencia**:

- Sea ***a*** una *acción de asignación* en un proceso

... y ***pre***(***a***) su precondición

- Sea ***C*** una *aseveración crítica* en otro proceso

Entonces ***a*** **no interfiere** con ***C*** si

$$\{ C \wedge pre(a) \}\ a\ \{ C \}$$

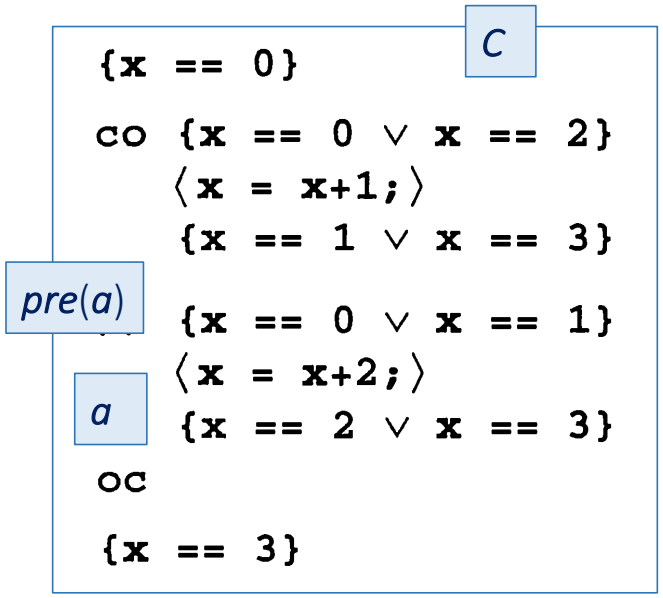
... es un teorema

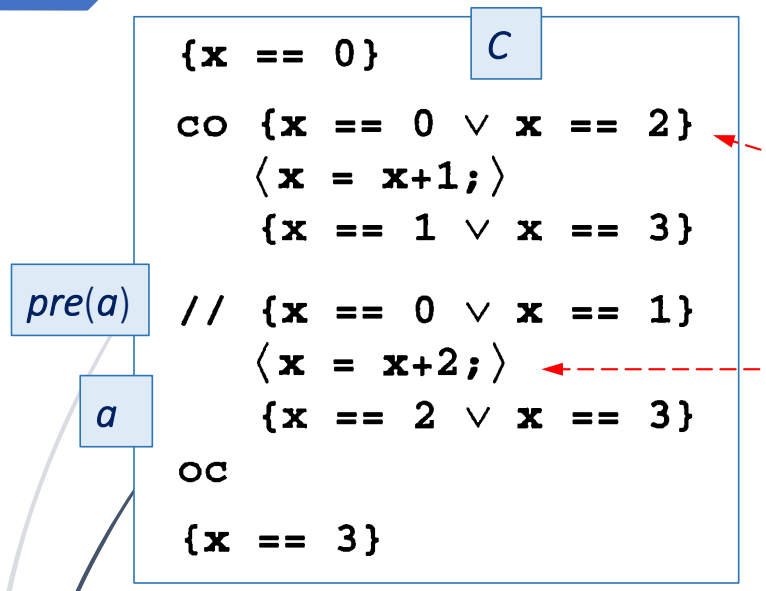
Acción de asignación:
- comando de asignación
- ... o comando `await` que contiene una o más asignaciones

Aseveración crítica:
- precondición o poscondición que no está dentro de un `await`

C es un invariante con respecto a la ejecución de ***a***

a puede ser ejecutada sólo si el proceso está en un estado que satisface ***pre***(***a***)

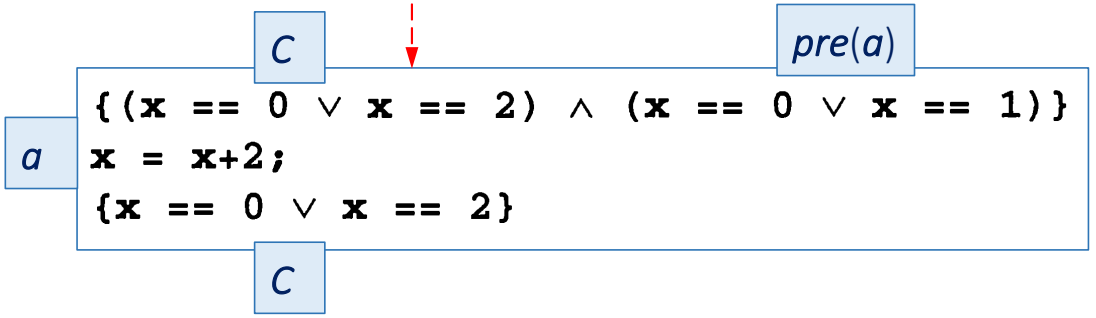

C
{x == 0}
co {x == 0 ∨ x == 2}
⟨x = x+1;⟩
{x == 1 ∨ x == 3}
pre(a)
{x == 0 ∨ x == 1}
⟨x = x+2;⟩
a
{x == 2 ∨ x == 3}
oc
{x == 3}



P.ej.,

La precondición del primer proceso es una aseveración crítica

... y no es interferida por el comando de asignación en el segundo proceso porque el siguiente triple es verdadero:



a no interfiere con *C* si

$\{ C \wedge pre(a) \}\ a\ \{ C \}$

... es un teorema

```
{x == 0}
co {x == 0 ∨ x == 2}
   ⟨x = x+1;⟩
   {x == 1 ∨ x == 3}
// {x == 0 ∨ x == 1}
   ⟨x = x+2;⟩
   {x == 2 ∨ x == 3}
oc
{x == 3}
```

P.ej.,

La precondición del primer proceso es una aseveración crítica

… y no es interferida por el comando de asignación en el segundo proceso porque el siguiente triple es verdadero:

```
{(x == 0 ∨ x == 2) ∧ (x == 0 ∨ x == 1)}
x = x+2;
{x == 0 ∨ x == 2}
```

Simplifica a { x = 0 }

… de manera que después de sumar 2 a x, el valor de x es ya sea 0 o 2

Hay otras tres aseveraciones críticas

Las demostraciones de no interferencia son similares

Si el segundo proceso se ejecuta antes que el primero, entonces el valor de x será 2 cuando el primer proceso comience a ejecutarse

Técnicas para evitar interferencia:

- Variables disjuntas
- Aseveraciones debilitadas
- Invariantes globales
- Sincronización

Técnicas para evitar interferencia:

- Variables disjuntas
- Aseveraciones debilitadas
- Invariantes globales
- Sincronización

Si el *conjunto de escritura* de un proceso es disjunto c/r al *conjunto de referencia* de un segundo proceso, y vice versa, entonces los dos procesos no pueden interferir

Las variables a las que el proceso asigna valores

Las variables que aparecen en las aseveraciones en una demostración sobre ese proceso

Técnicas para evitar interferencia:

- Variables disjuntas
- Aseveraciones debilitadas
- Invariantes globales
- Sincronización

… que capturan las relaciones entre variables globales

Si *I* es un predicado que hace referencia a variables globales

… entonces *I* es un **invariante global** c/r a un conjunto de procesos si

- *I* es verdadero cuando los procesos comienzan a ejecutarse
- *I* es preservado por cada acción de asignación

Para cada acción de asignación ***a***, *I* es verdadero después de ejecutar ***a*** suponiendo que *I* es verdadero antes de ejecutar ***a***

I es verdadero en el estado inicial de c/proceso

Técnicas para evitar interferencia:

- Variables disjuntas
- Aseveraciones debilitadas
- Invariantes globales
- Sincronización
  - Exclusión mutua
  - Sincronización por condición

Técnicas para evitar interferencia:

- Variables disjuntas
- Aseveraciones debilitadas
- Invariantes globales
- Sincronización

- Exclusión mutua
- Sincronización por condición

Consideremos

```
co P1: … a; …
|| P2: … S1; { C } S2 …
oc
```

… en que ***a*** interfiere con ***C***

Podemos "esconder" ***C*** de ***a*** combinando **S1** y **S2** en una única acción atómica

… de manera que el estado ***C*** es invisible para otros procesos

Asignación

Comandos

Aseveración crítica

Técnicas para evitar interferencia:

- Variables disjuntas
- Aseveraciones debilitadas
- Invariantes globales
- Sincronización

- Exclusión mutua
- Sincronización por condición

Consideremos

```
co P1: … a; …
|| P2: … S1; { C } S2 …
oc
```

… en que ***a*** interfiere con ***C***

Podemos "esconder" ***C*** de ***a*** combinando **S1** y **S2** en una única acción atómica

… de manera que el estado ***C*** es invisible para otros procesos

Asignación

Comandos

Aseveración crítica

Consideremos

```
co P1: … a; …
|| P2: … S1; { C } S2 …
oc
```

… en que ***a*** interfiere con ***C***

Podemos fortalecer la precondición de a :

⟨ **await** (¬***C*** o ***B***) ***a***; ⟩

B caracteriza el conjunto de estados tal que la ejecución de ***a*** vuelve verdadero a ***C***

Evita interferencia porque ya sea

- espera hasta que ***C*** sea falso, y por lo tanto **S2** no puede ejecutarse; o
- asegura que la ejecución de ***a*** hará verdadero a ***C***, y por lo tanto está OK ejecutar **S2**