

# Tema 1

## Sistemas Concurrentes y Distribuidos

### Introducción

Asignatura *Sistemas Concurrentes y Distribuidos*

Fecha 16 septiembre 2021

#### Conceptos básicos y motivación

Conceptos básicos relacionados con la Concurrencia

Motivación de la Programación Concurrente

#### Modelo abstracto de la PC y consideraciones sobre el hardware

Modelo abstracto de la PC  
Consideraciones sobre el hardware

#### Notaciones para PC

#### Exclusión mutua y sincronización

Exclusión mutua

#### Propiedades de los sistemas concurrentes

#### Verificación de programas concurrentes

Introducción

Enfoque axiomático

Departamento de Lenguajes y Sistemas Informáticos  
Universidad de Granada

# Concurrencia: programa y programación concurrente

## Concepto de *programa*

- **programa secuencial**: declaraciones de datos + conjunto de instrucciones ejecutables en secuencia
- **programa concurrente**: conjunto de “programas” secuenciales que se pueden ejecutar *lógicamente* en paralelo
- **proceso**: ejecución de 1 programa secuencial
- **Concurrencia**: potencial para la ejecución *paralela* de código: entremezclamiento de instrucciones de varios programas (real o virtual)
- **Programación Concurrente**: conjunto de notaciones y técnicas de programación utilizadas para expresar el paralelismo potencial y resolver problemas de sincronización y comunicación



Conceptos básicos y  
motivación

Conceptos básicos  
relacionados con la  
Concurrencia

Motivación de la  
Programación Concurrente

Modelo abstracto de la  
PC y consideraciones  
sobre el hardware

Modelo abstracto de la PC  
Consideraciones sobre el  
hardware

Notaciones para PC

Exclusión mutua y  
sincronización

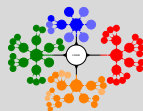
Exclusión mutua

Propiedades de los  
sistemas concurrentes

Verificación de  
programas  
concurrentes

Introducción

Enfoque axiomático



Conceptos básicos y  
motivación

Conceptos básicos  
relacionados con la  
Concurrencia

Motivación de la  
Programación Concurrente

Modelo abstracto de la  
PC y consideraciones  
sobre el hardware

Modelo abstracto de la PC  
Consideraciones sobre el  
hardware

Notaciones para PC

Exclusión mutua y  
sincronización

Exclusión mutua

Propiedades de los  
sistemas concurrentes

Verificación de  
programas  
concurrentes

Introducción

Enfoque axiomático

- **programación paralela:** enfocada a acelerar la resolución de problemas de eficiencia de los cálculos mediante el mejor aprovechamiento de la capacidad de proceso paralelo del hardware
- **programación distribuida:** su objetivo es hacer que varios componentes software localizados en distintos ordenadores trabajen juntos
- **programación de tiempo real:** resuelve la programación de sistemas que funcionan continuamente, reciben entradas/salidas desde componentes hardware y *reaccionan* frente a éstas, se han de cumplir plazos estrictos de tiempo para relizar las tareas del programa

# Beneficios de la Programación Concurrente

La PC es más compleja que la programación secuencial, entonces nos preguntamos:

*¿Por qué es necesario conocer la Programación Concurrente?*

Motivación del desarrollo de la PC como una disciplina diferenciada dentro del estudio de la Programación:

- Mejora la **eficiencia** de los programas
- Mejora la **calidad** de los programas



## Conceptos básicos y motivación

Conceptos básicos relacionados con la Concurrencia

## Motivación de la Programación Concurrente

## Modelo abstracto de la PC y consideraciones sobre el hardware

Modelo abstracto de la PC  
Consideraciones sobre el hardware

## Notaciones para PC

## Exclusión mutua y sincronización

Exclusión mutua

## Propiedades de los sistemas concurrentes

## Verificación de programas concurrentes

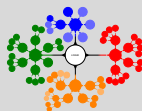
Introducción

Enfoque axiomático

# Beneficios de la Programación Concurrente: eficiencia

La PC permite aprovechar mejor los recursos hardware existentes en nuestro ordenador

- **En sistemas con un solo procesador:**
  - Al existir ahora varias tareas en los programas que avanzan “simultáneamente”, se evita la espera ociosa del procesador cuando el programa realiza E/S
  - Utilización del sistema por varios usuarios de forma interactiva (similar al funcionamiento de un SO multiusuario)
- **En sistemas con varios procesadores:**
  - Es posible repartir la ejecución de las tareas entre los distintos *núcleos* de un procesador
  - Se pueden asignar tareas muy complejas a otros procesadores
  - Para acelerar la ejecución de cálculos muy complejos



# Beneficios de la Programación Concurrente: calidad de los programas

Muchos problemas son más fáciles de resolver programando varios procesos secuenciales ejecutándose “virtualmente” o realmente en paralelo que con un único programa secuencial

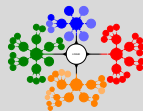
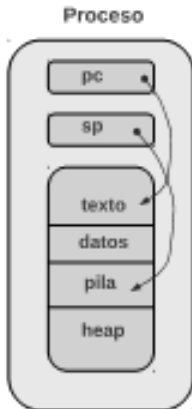
## Ejemplos:

- **Servidor Web para reserva de vuelos:** es más natural considerar cada petición de un usuario como un proceso y programar políticas para evitar situaciones conflictivas (evitar que el número de reservas supere el de asientos disponibles de un vuelo)
- **Simulador del funcionamiento de una gasolinera:** es más sencillo modelar los surtidores, clientes vehículos y empleados como procesos que cambian de estado, al participar en varias actividades comunes, que considerar a todos estos elementos como entidades de un único programa secuencial



# Abstracción de la PC

- *La Programación Concurrente es independiente de la implementación del paralelismo a bajo nivel (1 procesador, procesador multinúcleo, GPU, etc.)*
- *Se trata de una abstracción que nos ayuda a programar según un modelo “paralelo”*



## Conceptos básicos y motivación

Conceptos básicos  
relacionados con la  
Concurrencia

Motivación de la  
Programación Concurrente

## Modelo abstracto de la PC y consideraciones sobre el hardware

### Modelo abstracto de la PC

Consideraciones sobre el  
hardware

## Notaciones para PC

## Exclusión mutua y sincronización

Exclusión mutua

## Propiedades de los sistemas concurrentes

## Verificación de programas concurrentes

Introducción

Enfoque axiomático

- a poder razonar la solución de los problemas que se nos planteen, al nivel de detalle adecuado, sin entrar en consideraciones de demasiado bajo nivel que nos impidan obtener soluciones a los problemas de comunicación entre procesos de una forma sencilla;
- b facilitar la notación de programación, ya que propicia la definición de lenguajes de programación, que incluyen las instrucciones de comunicación y sincronización necesarias, sin tener que recurrir a llamadas al sistema o instrucciones de nivel máquina, para resolver los problemas que nos interesan en este ámbito;
- c los programas que desarrollemos serán transportables ya que son independientes de una máquina o plataforma de ejecución concreta, sólo con disponer de un compilador del lenguaje de programación concurrente.



### Conceptos básicos y motivación

Conceptos básicos relacionados con la Concurrency

Motivación de la Programación Concurrente

### Modelo abstracto de la PC y consideraciones sobre el hardware

#### Modelo abstracto de la PC

Consideraciones sobre el hardware

### Notaciones para PC

### Exclusión mutua y sincronización

Exclusión mutua

### Propiedades de los sistemas concurrentes

### Verificación de programas concurrentes

Introducción

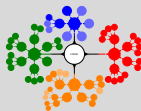
Enfoque axiomático



# Modelo abstracto de la Programación Concurrente

Se define a partir de 5 hipótesis, o axiomas si lo consideramos como un sistema lógico formal de base matemática para razonar sobre la corrección de los programas :

- i atomicidad y entrelazamiento de las instrucciones,
- ii coherencia en el acceso concurrente a los datos del programa,
- iii irrepetitibilidad de la secuencia de instrucciones atómicas,
- iv independencia de la velocidad relativa de ejecución de los procesos y
- v progreso de todos los procesos en tiempo finito.



## Conceptos básicos y motivación

Conceptos básicos  
relacionados con la  
Concurrencia

Motivación de la  
Programación Concurrente

## Modelo abstracto de la PC y consideraciones sobre el hardware

### Modelo abstracto de la PC

Consideraciones sobre el  
hardware

## Notaciones para PC

## Exclusión mutua y sincronización

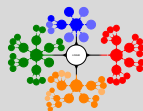
Exclusión mutua

## Propiedades de los sistemas concurrentes

## Verificación de programas concurrentes

Introducción

Enfoque axiomático



Conceptos básicos y  
motivación

Conceptos básicos  
relacionados con la  
Concurrencia

Motivación de la  
Programación Concurrente

Modelo abstracto de la  
PC y consideraciones  
sobre el hardware

Modelo abstracto de la PC

Consideraciones sobre el  
hardware

Notaciones para PC

Exclusión mutua y  
sincronización

Exclusión mutua

Propiedades de los  
sistemas concurrentes

Verificación de  
programas  
concurrentes

Introducción

Enfoque axiomático

## Sentencia atómica(indivisible)

Una sentencia o instrucción de un proceso en un programa concurrente es **atómica** si siempre se ejecuta de principio a fin sin verse *afectada* (durante su ejecución) por las sentencias en ejecución de otros procesos del programa

- No se verá afectada cuando el *funcionamiento* de dicha instrucción *no dependa nunca* de cómo se estén ejecutando otras instrucciones
- El funcionamiento de una instrucción se define por su efecto en el *estado de ejecución* del programa justo cuando ésta acaba
- El estado de ejecución de un programa concurrente está formado por los valores de las variables y de los registros de todos los procesos



Como ejemplo de **instrucciones atómicas** podemos citar muchas de las instrucciones máquina del repertorio de un procesador, por ejemplo las tres siguientes:

- Leer una celda de memoria (variable de tipo simple **x**) y cargar su valor en un registro (**r**) del procesador : `LOAD x`
- Incrementar el valor de un registro (**r**) u otras operaciones aritméticas sobre registros del procesador: `ADD r, a`
- Escribir el valor de un registro (**r**) en una celda de memoria: `STORE x`

Conceptos básicos y motivación

Conceptos básicos relacionados con la Concurrencia

Motivación de la Programación Concurrente

Modelo abstracto de la PC y consideraciones sobre el hardware

Modelo abstracto de la PC

Consideraciones sobre el hardware

Notaciones para PC

Exclusión mutua y sincronización

Exclusión mutua

Propiedades de los sistemas concurrentes

Verificación de programas concurrentes

Introducción

Enfoque axiomático

## Sentencias atómicas: ejemplo

El resultado de estas instrucciones no dependerá nunca de otras instrucciones que se estén ejecutando concurrentemente.

Al finalizar, la celda de memoria o el registro (donde se escribe) tomará un valor concreto predecible siempre a partir del estado de inicio, es decir, el estado del proceso justo al terminar la ejecución de la sentencia está *determinado*.

- Si en un estado, el registro  $r$  tiene el valor  $v$  y se ejecuta la instrucción de escritura de  $r$  en una variable  $x$ , entonces en el estado final de  $x$  sabemos que vale  $v$
- Lo anterior se puede afirmar incluso si existen otros procesos del programa accediendo a la variable  $x$
- El estado del proceso, al terminar la ejecución de una sentencia atómica, estará bien definido y se puede representar como una función matemática que sólo depende del estado al inicio de dicha ejecución



### Conceptos básicos y motivación

Conceptos básicos  
relacionados con la  
Concurrencia

Motivación de la  
Programación Concurrente

### Modelo abstracto de la PC y consideraciones sobre el hardware

#### Modelo abstracto de la PC

Consideraciones sobre el  
hardware

### Notaciones para PC

### Exclusión mutua y sincronización

Exclusión mutua

### Propiedades de los sistemas concurrentes

### Verificación de programas concurrentes

Introducción

Enfoque axiomático

## Sentencias no atómicas

La mayoría de las sentencias de los lenguajes de programación de alto nivel son típicamente no atómicas, por ejemplo, la sentencia:

`x := x + 1;` (incrementa el valor de la variable entera en 1 unidad)

El compilador utiliza una secuencia de tres sentencias:

- ❶ Leer el valor de **x** y cargarlo en un registro **r** del procesador
- ❷ Incrementar en una unidad el valor almacenado en el registro **r**
- ❸ Escribir el valor del registro **r** en la variable **x**

El valor que toma la variable **x** justo al terminar la ejecución de la sentencia dependerá de que haya o no otras sentencias ejecutándose a la vez y tratando de escribir simultáneamente en la variable **x**

Decimos en este caso que existe *indeterminación*: no se puede predecir el estado final del proceso a partir de su estado inicial



### Conceptos básicos y motivación

Conceptos básicos  
relacionados con la  
Concurrencia

Motivación de la  
Programación Concurrente

### Modelo abstracto de la PC y consideraciones sobre el hardware

#### Modelo abstracto de la PC

Consideraciones sobre el  
hardware

### Notaciones para PC

### Exclusión mutua y sincronización

Exclusión mutua

### Propiedades de los sistemas concurrentes

### Verificación de programas concurrentes

Introducción

Enfoque axiomático

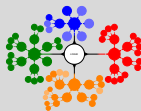
## (i) Entrelazamiento de instrucciones atómicas

Programa concurrente C compuesto por 2 procesos secuenciales:  $P_A$  y  $P_B$  que se ejecutan a la vez

La ejecución de C puede dar lugar a cualquiera de los posibles entrelazamientos de sentencias atómicas de  $P_A$  y  $P_B$ :

Pr.	Posibles secuencias de instrucciones atómicas
$P_A$	$A_1 A_2 A_3 A_4 A_5$
$P_B$	$B_1 B_2 B_3 B_4 B_5$
C	$A_1 A_2 A_3 A_4 A_5 B_1 B_2 B_3 B_4 B_5$
C	$B_1 B_2 B_3 B_4 B_5 A_1 A_2 A_3 A_4 A_5$
C	$A_1 B_1 A_2 B_2 \dots$
C	$B_1 B_2 A_1 B_3 B_4 A_2 \dots$
C	$\dots$

Las secuencia se va ordenando a medida que acaba cada una de las sentencias atómicas que la componen (que es cuando tienen efecto)



# Modelo de programación basado en el entrelazamiento de instrucciones atómicas



## Conceptos básicos y motivación

Conceptos básicos  
relacionados con la  
Concurrencia

Motivación de la  
Programación Concurrente

## Modelo abstracto de la PC y consideraciones sobre el hardware

### Modelo abstracto de la PC

Consideraciones sobre el  
hardware

## Notaciones para PC

## Exclusión mutua y sincronización

Exclusión mutua

## Propiedades de los sistemas concurrentes

## Verificación de programas concurrentes

Introducción

Enfoque axiomático

El modelo basado en el estudio de todas las posibles secuencias de ejecución entrelazadas de los procesos de un programa concurrente constituye una **abstracción**:

- Se consideran sólo las *características relevantes* que determinan el resultado final del programa
- Nos permite simplificar el análisis y estudio de los programas

Se *ignoran detalles* de la computación *no relevantes* para obtener el resultado, como por ejemplo:

- El estado de la memoria asignado a cada proceso
- Los registros particulares a los que accederá cada proceso
- El costo de los cambios de contexto entre procesos que realiza el sistema operativo
- La política concreta de planificación de los procesos
- Diferencias de velocidad entre entornos multiprocesador y monoprocesador

## (ii) coherencia en al acceso concurrente a los datos del programa

El entrelazamiento de instrucciones atómicas preserva la consistencia de los resultados

El resultado de una instrucción individual sobre un dato no depende de las circunstancias de la ejecución

Independencia total del entorno de ejecución de los procesos o del programa

Suponemos un programa  $P$  que se compone de 2 instrucciones atómicas,  $I_0$  e  $I_1$ , que se ejecutan concurrentemente, es decir,  $P \equiv I_0 \parallel I_1$ , entonces:

- Si  $I_0$  e  $I_1$  no acceden a la misma celda de memoria o registro, entonces el orden de ejecución de dichas instrucciones no afectará al resultado final
- Si no (acceden a la misma celda  $M$ ): Si  $I_0 \equiv M \leftarrow 1$  e  $I_1 \equiv M \leftarrow 2$ , la única suposición razonable es que el resultado sea consistente.
- Por tanto, al final  $M = 1$  ó  $M = 2$ , pero nunca podría alcanzar otro valor, por ejemplo:  $M = 3$

En caso de que no se cumpliera, sería imposible poder razonar sobre las propiedades de corrección de los programas concurrentes



### Conceptos básicos y motivación

Conceptos básicos  
relacionados con la  
Concurrencia

Motivación de la  
Programación Concurrente

### Modelo abstracto de la PC y consideraciones sobre el hardware

#### Modelo abstracto de la PC

Consideraciones sobre el  
hardware

### Notaciones para PC

#### Exclusión mutua y sincronización

Exclusión mutua

### Propiedades de los sistemas concurrentes

#### Verificación de programas concurrentes

Introducción

Enfoque axiomático



## (iii) Irrepetibilidad de la secuencia de instrucciones atómicas

### Estado de un programa concurrente

- Valores de las variables del programa en un momento dado. Incluyen variables declaradas explícitamente y variables con información de estado oculta (contador del programa, registros, ...)
- Un programa concurrente comienza su ejecución en un estado inicial y los procesos van modificando el estado conforme se van ejecutando sus sentencias atómicas (producen transiciones entre dos estados de forma indivisible)

**Historia o traza de un programa concurrente:** Secuencia de estados  $s_0 \rightarrow s_1 \rightarrow \dots \rightarrow s_n$ , producida por una secuencia concreta de entrelazamiento

**Irrepetibilidad de las historias:** La probabilidad de que 2 trazas de ejecución seguidas del mismo programa coincidan es prácticamente "0"



#### Conceptos básicos y motivación

Conceptos básicos  
relacionados con la  
Concurrencia

Motivación de la  
Programación Concurrente

#### Modelo abstracto de la PC y consideraciones sobre el hardware

##### Modelo abstracto de la PC

Consideraciones sobre el  
hardware

#### Notaciones para PC

#### Exclusión mutua y sincronización

Exclusión mutua

#### Propiedades de los sistemas concurrentes

#### Verificación de programas concurrentes

Introducción

Enfoque axiomático

## (iv) Independencia de la velocidad realtiva de ejecución de los procesos

No se puede hacer ninguna suposición acerca de las velocidades absolutas/relativas de ejecución de los procesos, salvo que es mayor que cero

Un programa concurrente se entiende sólo en base a sus componentes (procesos) y sus interacciones, sin tener en cuenta el entorno de ejecución.

**Ejemplo:** un disco es más lento que el procesador, pero no se debe asumir este hecho en el diseño de un programa concurrente.

Si se hicieran suposiciones que dependiesen del tiempo o de la velocidad de ejecución de los procesos:

- Sería difícil detectar y corregir fallos
- La corrección dependería de la configuración de ejecución, que puede cambiar



### Conceptos básicos y motivación

Conceptos básicos relacionados con la Concurrency

Motivación de la Programación Concurrente

### Modelo abstracto de la PC y consideraciones sobre el hardware

#### Modelo abstracto de la PC

Consideraciones sobre el hardware

### Notaciones para PC

### Exclusión mutua y sincronización

Exclusión mutua

### Propiedades de los sistemas concurrentes

### Verificación de programas concurrentes

Introducción

Enfoque axiomático

## (v) Hipótesis de *progreso finito* de los procesos

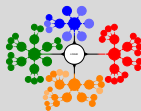
Si se cumple la hipótesis, la velocidad de ejecución de cada proceso será no nula, lo cual tiene estas dos consecuencias:

### Punto de vista global

- Durante la ejecución de un programa concurrente, en cualquier momento existirá al menos 1 proceso preparado, es decir, eventualmente el planificador de bajo nivel del sistema seleccionará a algún proceso para su ejecución

### Punto de vista local

- Cuando un proceso concreto de un programa concurrente comienza la ejecución de una sentencia, completará la ejecución de la sentencia en un intervalo de tiempo finito



#### Conceptos básicos y motivación

Conceptos básicos  
relacionados con la  
Concurrencia

Motivación de la  
Programación Concurrente

#### Modelo abstracto de la PC y consideraciones sobre el hardware

##### Modelo abstracto de la PC

Consideraciones sobre el  
hardware

#### Notaciones para PC

#### Exclusión mutua y sincronización

Exclusión mutua

#### Propiedades de los sistemas concurrentes

#### Verificación de programas concurrentes

Introducción

Enfoque axiomático

# Modelos de arquitecturas para PC

## Mecanismos de implementación de la concurrencia en los programas:

- Dependen de la arquitectura del procesador
- Consideran una *máquina virtual* (multiprocesador o multicomputador) como la base común para modelar la ejecución de los procesos concurrentes
- El tipo de paralelismo (monoprocesador, multiprocesador, sistema distribuido o multicomputador) afectará a la eficiencia de la ejecución pero nunca a la corrección de los programas



### Conceptos básicos y motivación

Conceptos básicos  
relacionados con la  
Concurrencia

Motivación de la  
Programación Concurrente

### Modelo abstracto de la PC y consideraciones sobre el hardware

Modelo abstracto de la PC

Consideraciones sobre el  
hardware

### Notaciones para PC

#### Exclusión mutua y sincronización

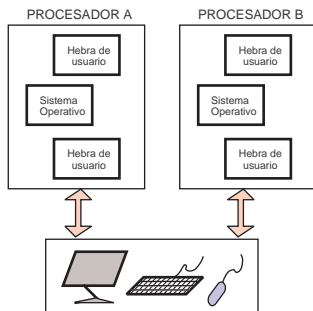
Exclusión mutua

#### Propiedades de los sistemas concurrentes

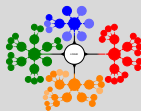
#### Verificación de programas concurrentes

Introducción

Enfoque axiomático



- **Multiprogramación:** el sistema operativo gestiona cómo múltiples procesos se reparten los ciclos del procesador
- Mejor aprovechamiento de la CPU
- Servicio interactivo a varios usuarios
- Utilización de soluciones de diseño *concurrentes*
- Sincronización y comunicación mediante variables compartidas



### Conceptos básicos y motivación

Conceptos básicos relacionados con la Concurrencia

Motivación de la Programación Concurrente

### Modelo abstracto de la PC y consideraciones sobre el hardware

Modelo abstracto de la PC

Consideraciones sobre el hardware

### Notaciones para PC

#### Exclusión mutua y sincronización

Exclusión mutua

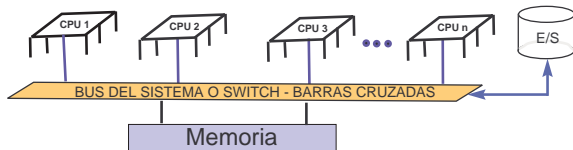
#### Propiedades de los sistemas concurrentes

#### Verificación de programas concurrentes

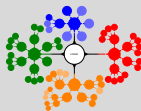
Introducción

Enfoque axiomático

# Concurrencia en multiprocesadores o sistemas *multinúcleo*



- Los procesadores pueden compartir o no físicamente la misma memoria, pero comparten algún espacio de direcciones común
- Los núcleos de los procesadores actuales comparten una memoria de acceso muy rápido y un computador podría tener además varios procesadores
- La interacción entre procesos se puede implementar mediante variables ubicadas en un espacio de memoria común a todos los procesadores (*variables compartidas*)



## Conceptos básicos y motivación

Conceptos básicos  
relacionados con la  
Concurrencia

Motivación de la  
Programación Concurrente

## Modelo abstracto de la PC y consideraciones sobre el hardware

Modelo abstracto de la PC

Consideraciones sobre el  
hardware

## Notaciones para PC

### Exclusión mutua y sincronización

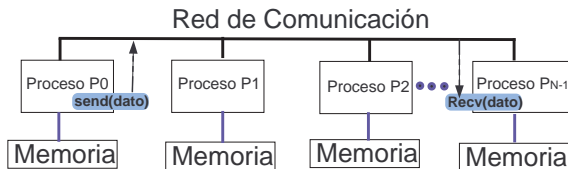
Exclusión mutua

### Propiedades de los sistemas concurrentes

### Verificación de programas concurrentes

Introducción

Enfoque axiomático



- No existe memoria común, cada procesador mantiene un espacio de direcciones propio
- Los procesos interactúan transfiriéndose datos a través de una red de interconexión (paso de mensajes)
- Es necesario utilizar *programación distribuida*, que además de la concurrencia trata con otros problemas: tratamiento de fallos, transparencia, heterogeneidad, etc.
- Los clusters de ordenadores o de GPUs (actualmente), Internet, etc.

## Conceptos básicos y motivación

Conceptos básicos  
relacionados con la  
Concurrencia

Motivación de la  
Programación Concurrente

## Modelo abstracto de la PC y consideraciones sobre el hardware

Modelo abstracto de la PC

Consideraciones sobre el  
hardware

## Notaciones para PC

### Exclusión mutua y sincronización

Exclusión mutua

### Propiedades de los sistemas concurrentes

### Verificación de programas concurrentes

Introducción

Enfoque axiomático

# Notaciones para expresar ejecución concurrente



## Conceptos básicos y motivación

Conceptos básicos relacionados con la Concurrency

Motivación de la Programación Concurrente

## Modelo abstracto de la PC y consideraciones sobre el hardware

Modelo abstracto de la PC  
Consideraciones sobre el hardware

## Notaciones para PC

### Exclusión mutua y sincronización

Exclusión mutua

### Propiedades de los sistemas concurrentes

### Verificación de programas concurrentes

Introducción

Enfoque axiomático

**Propuestas iniciales:** no separan la definición de los procesos de su sincronización

**Propuestas posteriores:** separan ambos conceptos e imponen una estructura al programa concurrente, diferente de uno secuencial

**Declaración de procesos:** rutinas específicas de programación concurrente  $\Rightarrow$  Estructura del programa concurrente más clara

## Sistemas Estáticos:

- Número de procesos fijado en el fuente del programa
- Los procesos se activan al lanzar el programa
- Ejemplo: Message Passing Interface (MPI-1)

## Sistemas Dinámicos:

- Número variable de procesos/hebras que se pueden activar en cualquier momento de la ejecución
- Ejemplos: OpenMP, PThreads, Java Threads, MPI-2

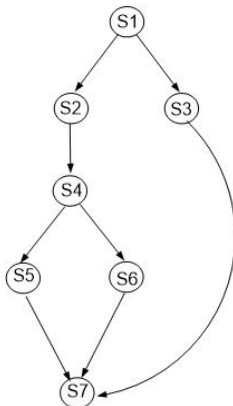


## Grafo de sincronización

El Grafo de Sincronización es un Grafo Dirigido Acíclico (DAG) donde cada nodo representa una secuencia de sentencias del programa (actividad):

Dadas dos actividades,  $S_1$  y  $S_2$ , una arista (flecha) desde  $S_1$  hacia  $S_2$  significa que  $S_2$  no puede comenzar su ejecución hasta que  $S_1$  haya finalizado

Muestra las restricciones de precedencia que determinan cuándo una actividad puede empezar en un programa



### Conceptos básicos y motivación

Conceptos básicos  
relacionados con la  
Concurrencia

Motivación de la  
Programación Concurrente

### Modelo abstracto de la PC y consideraciones sobre el hardware

Modelo abstracto de la PC  
Consideraciones sobre el hardware

### Notaciones para PC

#### Exclusión mutua y sincronización

Exclusión mutua

#### Propiedades de los sistemas concurrentes

#### Verificación de programas concurrentes

Introducción

Enfoque axiomático

# Definición estática de procesos

```
var ... \\variables compartidas

process Uno;
var ... \\variables locales
begin
    ... \\codigo
end;
process Dos;
var ... \\variables locales
begin
    ... \\codigo
end;
... \\otros procesos
```

Se *lanza* la ejecución concurrente de ambos procesos en 1 bloque del programa:

```
cobegin  Uno  ||  Dos  coend;
```

El programa acaba cuando acaban de ejecutarse todas las instrucciones de 2 los procesos. Las variables compartidas se inicializan antes de comenzar la ejecución concurrente de los procesos



## Conceptos básicos y motivación

Conceptos básicos  
relacionados con la  
Concurrencia

Motivación de la  
Programación Concurrente

## Modelo abstracto de la PC y consideraciones sobre el hardware

Modelo abstracto de la PC  
Consideraciones sobre el  
hardware

## Notaciones para PC

### Exclusión mutua y sincronización

Exclusión mutua

### Propiedades de los sistemas concurrentes

Verificación de  
programas  
concurrentes

Introducción

Enfoque axiomático

# Definición estática de vectores de procesos



```
var ... \\variables compartidas

process NomP[ind : a.. b];
var ... \\variables locales
begin
    ... \\codigo
    ...\\ (ind vale a, a+1, ... b)
end;
... \\otros procesos
```

## Conceptos básicos y motivación

Conceptos básicos  
relacionados con la  
Concurrencia

Motivación de la  
Programación Concurrente

## Modelo abstracto de la PC y consideraciones sobre el hardware

Modelo abstracto de la PC  
Consideraciones sobre el  
hardware

## Notaciones para PC

### Exclusión mutua y sincronización

Exclusión mutua

### Propiedades de los sistemas concurrentes

### Verificación de programas concurrentes

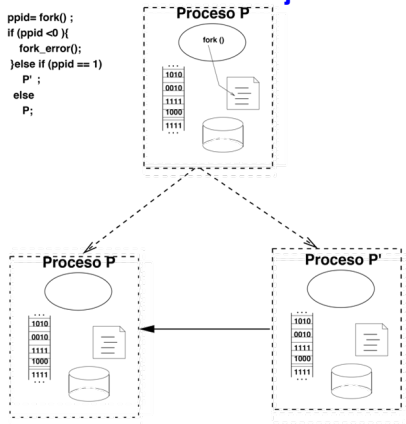
Introducción

Enfoque axiomático

# Creación de procesos no estructurada: fork-join

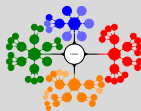
```
procedure P2;  
begin  
  D;  
end;  
process P1;  
begin  
  A;  
  fork P2;  
  B;  
  join P2;  
  C;  
end;
```

```
ppid= fork() ;  
if (ppid <0){  
  fork_error();  
}else if (ppid == 1)  
  P' ;  
else  
  P;
```



**fork:** sentencia que especifica que la rutina nombrada puede comenzar su ejecución, al mismo tiempo que comienza la sentencia siguiente (bifurcación)

**join:** sentencia que espera la terminación de la rutina nombrada, antes de comenzar la sentencia siguiente (unión)



## Conceptos básicos y motivación

Conceptos básicos  
relacionados con la  
Concurrencia

Motivación de la  
Programación Concurrente

## Modelo abstracto de la PC y consideraciones sobre el hardware

Modelo abstracto de la PC  
Consideraciones sobre el  
hardware

## Notaciones para PC

### Exclusión mutua y sincronización

Exclusión mutua

### Propiedades de los sistemas concurrentes

### Verificación de programas concurrentes

Introducción

Enfoque axiomático



No todas las secuencias de entrelazamiento de las instrucciones de los procesos a que da lugar la ejecución de un programa concurrente son posibles

- Los procesos no suelen ejecutarse de una forma totalmente independiente, sino que que **colaboran** entre ellos
- **Condición de sincronización:** restricción en el orden en que se pueden entremezclar las instrucciones que generan los procesos de un programa
- **Exclusión mutua:** se da en secuencias finitas de intrucciones del código de un programa, que han de ejecutarse de principio a fin por un único proceso, que no puede ser desplazado mientras ejecuta esta sección crítica de instrucciones

Conceptos básicos y motivación

Conceptos básicos relacionados con la Concurrency

Motivación de la Programación Concurrente

Modelo abstracto de la PC y consideraciones sobre el hardware

Modelo abstracto de la PC  
Consideraciones sobre el hardware

Notaciones para PC

Exclusión mutua y sincronización

Exclusión mutua

Propiedades de los sistemas concurrentes

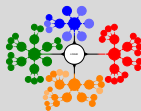
Verificación de programas concurrentes

Introducción

Enfoque axiomático

Al conjunto de dichas secuencias comunes de instrucciones consecutivas que aparecen en el texto de varios procesos de un programa concurrente, se le denomina **sección crítica (SC)**

- Ocurre **exclusión mutua (EM)** cuando los procesos solo funcionan correctamente si, en cada instante de tiempo, hay como mucho uno de ellos ejecutando cualquier instrucción de la sección crítica
- El entrelazamiento de las instrucciones de los procesos debe ser tal que cada secuencia de instrucciones de la SC se ejecuta como mucho por un proceso de principio a fin, sin que (durante ese tiempo) otros procesos ejecuten ninguna de esas instrucciones ni otras de la misma SC



Conceptos básicos y motivación

Conceptos básicos relacionados con la Concurrency

Motivación de la Programación Concurrente

Modelo abstracto de la PC y consideraciones sobre el hardware

Modelo abstracto de la PC

Consideraciones sobre el hardware

Notaciones para PC

Exclusión mutua y sincronización

Exclusión mutua

Propiedades de los sistemas concurrentes

Verificación de programas concurrentes

Introducción

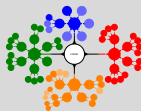
Enfoque axiomático

## Ejemplo sencillo de exclusión mutua

Sea  $x$  una variable entera en memoria compartida y afectada por operaciones aritméticas elementales

- La sección crítica estará formada por todas las secuencias de instrucciones máquina que se obtienen al traducir (compilar) operaciones de escritura (o lectura y escritura) de la variable, asignaciones tales como  $x := x + 1$  o  $x := 4 * z$
- Si varios procesos ejecutan las instrucciones máquina de la sección crítica de forma simultánea, se producirán *condiciones de carrera* entre los procesos  $\Rightarrow$  los resultados de este tipo de asignaciones son **indeterminados**

Aquí, el término **indeterminado** indica que para cada valor inicial de  $x$ , existe un conjunto de posibles valores de  $x$  al finalizar. El valor concreto que toma  $x$  depende la secuencia de instrucciones concreta que ocurre



### Conceptos básicos y motivación

Conceptos básicos  
relacionados con la  
Concurrencia

Motivación de la  
Programación Concurrente

### Modelo abstracto de la PC y consideraciones sobre el hardware

Modelo abstracto de la PC  
Consideraciones sobre el  
hardware

### Notaciones para PC

### Exclusión mutua y sincronización

#### Exclusión mutua

### Propiedades de los sistemas concurrentes

### Verificación de programas concurrentes

Introducción

Enfoque axiomático

# Traducción y ejecución (de las instrucciones máquina)

Si consideramos la instrucción  $x := x + 1$  (que forma una SC), una traducción típica a código máquina tendría estas tres instrucciones:

- 1 `load ri ← x` cargar el valor de la variable  $x$  en un registro  $r$  de la CPU (por el proceso número  $i$ )
- 2 `add ri, 1` incrementar en una unidad el valor del registro  $r$
- 3 `store ri → x` guardar el valor del registro  $r$  en la posición de memoria de la variable  $x$

Dos procesos ( $P_0$  y  $P_1$ ) ejecutan la asignación: El entrelazamiento de instrucciones es arbitrario:

- Cada proceso tiene su registro (los llamaremos  $r_0$  y  $r_1$ )
- Ambos comparten la variable  $x$ , cuyos accesos vía `load` y `store` son atómicos



## Conceptos básicos y motivación

Conceptos básicos  
relacionados con la  
Concurrencia

Motivación de la  
Programación Concurrente

## Modelo abstracto de la PC y consideraciones sobre el hardware

Modelo abstracto de la PC  
Consideraciones sobre el  
hardware

## Notaciones para PC

## Exclusión mutua y sincronización

### Exclusión mutua

## Propiedades de los sistemas concurrentes

## Verificación de programas concurrentes

Introducción

Enfoque axiomático



## Secuencias de instrucciones del ejemplo



Suponemos que inicialmente  $x$  vale 0 y ambos procesos ejecutan la asignación, puede haber varias secuencias de entrelazamiento, aquí vemos dos:

$P_0$	$P_1$	$x$	$P_0$	$P_1$	$x$
load $r0 \leftarrow x$		0	load $r0 \leftarrow x$		0
add $r0, 1$		0		load $r1 \leftarrow x$	0
store $r0 \rightarrow x$		1	add $r0, 1$		0
	load $r1 \leftarrow x$	1		add $r1, 1$	0
	add $r1, 1$	1	store $r0 \rightarrow x$		1
	store $r1 \rightarrow x$	2		store $r1 \rightarrow x$	1

Partiendo de  $x == 0$ , al final  $x$  puede valer **1 ó 2**, dependiendo del entrelazamiento

### Conceptos básicos y motivación

Conceptos básicos relacionados con la Concurrency

Motivación de la Programación Concurrente

### Modelo abstracto de la PC y consideraciones sobre el hardware

Modelo abstracto de la PC

Consideraciones sobre el hardware

### Notaciones para PC

### Exclusión mutua y sincronización

Exclusión mutua

### Propiedades de los sistemas concurrentes

### Verificación de programas concurrentes

Introducción

Enfoque axiomático

## Instrucciones compuestas atómicas

En nuestra notación de pseudo-código, podemos escribir sentencias compuestas indicando que se deben de ejecutar de forma atómica, usando los caracteres `<` y `>`:

```
{ instr. no atómicas }
begin
x := 0 ;
cobegin
x:= x+1 ;
x:= x-1 ;
coend
end
////////////////////
{ instr. atómicas }
begin
x := 0 ;
cobegin
< x:= x+1 > ;
< x:= x-1 > ;
coend
end
```

- En el primer código, al acabar, **x** puede tener un valor cualquiera del conjunto  $\{-1, 0, 1\}$
- En el segundo código **x** finaliza siempre con el valor 0



### Conceptos básicos y motivación

Conceptos básicos  
relacionados con la  
Concurrencia

Motivación de la  
Programación Concurrente

### Modelo abstracto de la PC y consideraciones sobre el hardware

Modelo abstracto de la PC  
Consideraciones sobre el  
hardware

### Notaciones para PC

### Exclusión mutua y sincronización

#### Exclusión mutua

### Propiedades de los sistemas concurrentes

### Verificación de programas concurrentes

Introducción

Enfoque axiomático

En un programa concurrente, una condición de sincronización establece que todos los posibles entrelazamientos (secuencias de instrucciones atómicas de los procesos) son correctas

- Suele ocurrir cuando, en un punto concreto de su ejecución, uno o varios procesos deben esperar a que se cumpla una determinada condición global (depende de varios procesos)

Un ejemplo sencillo de **condición de sincronización**: procesos concurrentes que pueden utilizar variables comunes para comunicarse (memoria compartida), ya que los accesos a las variables no pueden ordenarse arbitrariamente (p.ej.: leer de la variable antes de que sea escrita)



Conceptos básicos y motivación

Conceptos básicos relacionados con la Concurrency

Motivación de la Programación Concurrente

Modelo abstracto de la PC y consideraciones sobre el hardware

Modelo abstracto de la PC  
Consideraciones sobre el hardware

Notaciones para PC

Exclusión mutua y sincronización

Exclusión mutua

Propiedades de los sistemas concurrentes

Verificación de programas concurrentes

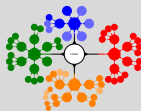
Introducción

Enfoque axiomático

# El paradigma del *Productor Consumidor*

Dos procesos cooperantes en los cuales uno de ellos (*productor*) genera una secuencia de valores (p.ej.enteros) y el otro (*consumidor*) utiliza cada uno de estos valores

```
var x : integer ; { contiene cada valor producido }
{ Proceso productor: calcula 'x' }
process Productor ;
var a : integer ; { no compartida }
begin
  while true do begin
    { calcular un valor }
    a := ProducirValor() ;
    { escribir en mem. compartida }
    x := a ; { sentencia E }
  end
end
process Consumidor ; { Proceso Consumidor: lee 'x' }
var b : integer ; { no compartida }
begin
  while true do begin
    { leer de mem. compartida }
    b := x ; { sentencia L }
    { utilizar el valor leído }
    UsarValor(b) ;
  end
end
```



## Conceptos básicos y motivación

Conceptos básicos  
relacionados con la  
Concurrencia

Motivación de la  
Programación Concurrente

## Modelo abstracto de la PC y consideraciones sobre el hardware

Modelo abstracto de la PC  
Consideraciones sobre el  
hardware

## Notaciones para PC

### Exclusión mutua y sincronización

#### Exclusión mutua

## Propiedades de los sistemas concurrentes

### Verificación de programas concurrentes

Introducción

Enfoque axiomático

## Trazas incorrectas de un programa concurrente

Los procesos sólo *funcionan* como se espera si el orden en el que se entrelazan las sentencias elementales etiquetadas como E (escritura) y L (lectura) es: E, L, E, L, E, L, . . .

Trazas incorrectas del programa *Productor Consumidor*:

- **L, E, L, E, . . .** es incorrecta: se hace una lectura de **x** previa a cualquier escritura (se lee valor indeterminado)
- **E, L, E, E, L, . . .** es incorrecta: hay dos escrituras sin ninguna lectura entre ellas (se produce un valor que no se lee)
- **E, L, L, E, L, . . .** es incorrecta (para el código del P/C anterior): hay dos lecturas de un mismo valor que, por tanto, es usado dos veces

La secuencia válida asegura la condición de sincronización:

- *Consumidor* no lee hasta que *Productor* escribe un nuevo valor en **x** (cada valor producido es usado una sola vez)
- *Productor* no escribe un nuevo valor hasta que *Consumidor* lea el último valor almacenado en **x** (ningún valor producido se pierde)



### Conceptos básicos y motivación

Conceptos básicos  
relacionados con la  
Concurrencia

Motivación de la  
Programación Concurrente

### Modelo abstracto de la PC y consideraciones sobre el hardware

Modelo abstracto de la PC  
Consideraciones sobre el  
hardware

### Notaciones para PC

### Exclusión mutua y sincronización

#### Exclusión mutua

### Propiedades de los sistemas concurrentes

### Verificación de programas concurrentes

Introducción

Enfoque axiomático

# Concepto de corrección de un programa concurrente

Propiedad de un programa concurrente: Algo que se puede afirmar del programa que es cierto para todas las posibles secuencias de entrelazamiento (trazas del programa)

Hay 2 tipos de propiedades:

- Propiedad de seguridad (safety).
- Propiedad de vivacidad (liveness).



## Conceptos básicos y motivación

Conceptos básicos relacionados con la Concurrency

Motivación de la Programación Concurrente

## Modelo abstracto de la PC y consideraciones sobre el hardware

Modelo abstracto de la PC

Consideraciones sobre el hardware

## Notaciones para PC

## Exclusión mutua y sincronización

Exclusión mutua

## Propiedades de los sistemas concurrentes

## Verificación de programas concurrentes

Introducción

Enfoque axiomático



Condiciones que deben cumplirse en cada instante de la traza del programa; son del tipo: *nunca pasará nada malo*

- Requeridas en especificaciones estáticas del programa
- Son fáciles de demostrar y para cumplirlas se suelen impedir determinadas posibles trazas

Ejemplos:

- *Exclusión mutua*: 2 procesos nunca entrelazan ciertas subsecuencias de operaciones
- *Ausencia Interbloqueo* (Deadlock-freedom): Nunca ocurrirá que los procesos se encuentren esperando algo que nunca sucederá
- *Propiedad de seguridad en el Productor-Consumidor*: El consumidor debe consumir todos los datos producidos por el productor en el orden en que se van produciendo

Conceptos básicos y motivación

Conceptos básicos relacionados con la Concurrencia

Motivación de la Programación Concurrente

Modelo abstracto de la PC y consideraciones sobre el hardware

Modelo abstracto de la PC  
Consideraciones sobre el hardware

Notaciones para PC

Exclusión mutua y sincronización

Exclusión mutua

Propiedades de los sistemas concurrentes

Verificación de programas concurrentes

Introducción

Enfoque axiomático

Son propiedades que deben cumplirse en algún momento futuro y no especificado del programa; son del tipo: *realmente sucede (o va a suceder) algo bueno*

- Son propiedades dinámicas, más difíciles de demostrar que las propiedades de seguridad

Ejemplos:

- *Ausencia de inanición* (starvation-freedom): Un proceso o grupo de procesos del programa no puede ser indefinidamente pospuesto. Asegura que, en algún momento, podrá avanzar
- *Equidad* (fairness): Tipo particular de propiedad de vivacidad. Un proceso que pueda progresar debe hacerlo con justicia relativa con respecto a los demás procesos del programa. Más ligado a la implementación y a veces se incumple: existen distintos grados



### Conceptos básicos y motivación

Conceptos básicos  
relacionados con la  
Concurrencia

Motivación de la  
Programación Concurrente

### Modelo abstracto de la PC y consideraciones sobre el hardware

Modelo abstracto de la PC  
Consideraciones sobre el  
hardware

### Notaciones para PC

### Exclusión mutua y sincronización

Exclusión mutua

### Propiedades de los sistemas concurrentes

### Verificación de programas concurrentes

Introducción

Enfoque axiomático



# Pruebas simples de programas concurrentes

¿Cómo demostrar que un programa cumple una determinada propiedad ?

- *Posibilidad*: realizar diferentes ejecuciones del programa y comprobar que se verifica la propiedad
- *Problema*: Sólo permite considerar un número finito y muy limitado de trazas del programa y no demuestra la ausencia de casos indeseables (en alguna traza no explorada)
- Ejemplo: Comprobar que el proceso P produce al final  $x == 3$ :

```
process P ;  
var x : integer := 0 ;  
cobegin  
x = x+1 ; x = x+2 ;  
coend
```

(hay varias trazas que llevan al resultado:  $x==1$  o  $x==2$ , y estas historias podrían ocurrir en algunas ejecuciones, por tanto, produciendo un resultado no-determinado)



## Conceptos básicos y motivación

Conceptos básicos  
relacionados con la  
Concurrencia

Motivación de la  
Programación Concurrente

## Modelo abstracto de la PC y consideraciones sobre el hardware

Modelo abstracto de la PC  
Consideraciones sobre el  
hardware

## Notaciones para PC

### Exclusión mutua y sincronización

Exclusión mutua

## Propiedades de los sistemas concurrentes

## Verificación de programas concurrentes

### Introducción

Enfoque axiomático

- *Enfoque operacional*: Análisis exhaustivo de casos. Se comprueba la corrección de todas las posibles trazas
- *Problema*: Su utilidad está muy limitada cuando se aplica a los programas concurrentes, ya que el número de entrelazamientos crece exponencialmente con el número de instrucciones de los procesos

Para el sencillo programa P, formado por 2 procesos y 3 sentencias atómicas por proceso, tendríamos que estudiar 20 historias diferentes (*=permutaciones con repetición de 6 elementos con 2 grupos de 3 elementos*)



#### Conceptos básicos y motivación

Conceptos básicos relacionados con la Concurrency

Motivación de la Programación Concurrente

#### Modelo abstracto de la PC y consideraciones sobre el hardware

Modelo abstracto de la PC

Consideraciones sobre el hardware

#### Notaciones para PC

#### Exclusión mutua y sincronización

Exclusión mutua

#### Propiedades de los sistemas concurrentes

#### Verificación de programas concurrentes

#### Introducción

Enfoque axiomático



## Conceptos básicos y motivación

Conceptos básicos  
relacionados con la  
ConcurrenciaMotivación de la  
Programación ConcurrenteModelo abstracto de la  
PC y consideraciones  
sobre el hardwareModelo abstracto de la PC  
Consideraciones sobre el  
hardware

## Notaciones para PC

Exclusión mutua y  
sincronización

Exclusión mutua

Propiedades de los  
sistemas concurrentesVerificación de  
programas  
concurrentes

Introducción

Enfoque axiomático

## Verificación: enfoque axiomático

Se define un sistema lógico formal que permite establecer propiedades de programas en base a axiomas y reglas de inferencia

- Se usan fórmulas lógicas (llamados: *asertos* o *predicados*) para caracterizar un conjunto de estados
- Las sentencias atómicas actúan como transformadores de tales asertos. Los teoremas se escriben de la forma:

$$\{P\} S \{Q\}$$

**Interpretación** de un teorema (llamado también: *triple*) en esta lógica: “*Si la ejecución de la sentencia **S** comienza en algún estado en el que es verdadero el aserto **P** (precondición), entonces el aserto **Q** (poscondición) será verdadero en el estado resultante*”

**Menor Complejidad** que el enfoque operacional: El trabajo que conlleva la prueba de corrección es proporcional al número de sentencias atómicas en el programa



## Notación: *Triples de Hoare*

- Se introduce la notación de *especificación de corrección parcial* para especificar lo que hace un programa

$$\{P\} C \{Q\}$$

- C es un programa del lenguaje cuyos programas están siendo especificados
- P y Q son **asertos** definidos con las variables programa que representa C
- Las variables libres de P y Q son del programa o son variables lógicas
- Los asertos caracterizan los estados aceptables del programa:
  - V caracteriza a todos los estados del programa; sin embargo, F no caracteriza a ninguno
  - Los *triples* ( $\{P\} C \{Q\}$ ) son los teoremas (= *proposiciones demostrables*) con la lógica que estamos tratando

### Conceptos básicos y motivación

Conceptos básicos relacionados con la Concurrency

Motivación de la Programación Concurrente

### Modelo abstracto de la PC y consideraciones sobre el hardware

Modelo abstracto de la PC  
Consideraciones sobre el hardware

### Notaciones para PC

#### Exclusión mutua y sincronización

Exclusión mutua

### Propiedades de los sistemas concurrentes

#### Verificación de programas concurrentes

Introducción

Enfoque axiomático



## Conceptos básicos y motivación

Conceptos básicos  
relacionados con la  
Concurrencia

Motivación de la  
Programación Concurrente

## Modelo abstracto de la PC y consideraciones sobre el hardware

Modelo abstracto de la PC  
Consideraciones sobre el  
hardware

## Notaciones para PC

### Exclusión mutua y sincronización

Exclusión mutua

## Propiedades de los sistemas concurrentes

### Verificación de programas concurrentes

Introducción

Enfoque axiomático

$\{P\} C \{Q\}$  es cierto si

- siempre que  $C$  es ejecutado en un estado que
- satisface  $P$  y si la ejecución de  $C$  termina,
- el estado en que  $C$  termina satisface  $Q$

Ejemplo:  $\{X == 1\} X = X + 1 \{X == 2\}$

- el aserto  $P$  dice que el valor de  $X$  es 1
- el aserto  $Q$  dice que el valor de  $X$  es 2
- $C$  es la sentencia de asignación  $X = X + 1$
- $\{X == 1\} X = X + 1 \{X == 2\}$  es cierto
- $\{X == 1\} X = X + 1 \{X == 3\}$  es falso
- $\{X == 1\} \text{WHILE } T \text{ DO NULL } \{Y == 3\}$  ¡es cierto!

## La estructura de las demostraciones

Una demostración es una secuencia de líneas (o *triples*) de la forma  $\{P\} C \{Q\}$ , cada una de los cuales es un *axioma* o deriva de los anteriores aplicando una *regla de inferencia* de la lógica

- Una demostración consiste en una secuencia de *líneas*:
  - como la definición de  $()^2$  del ejemplo siguiente
- Cada una de las líneas es una instancia de un *axioma*
- o se deriva de las líneas anteriores por medio de una *regla de inferencia*
- La sentencia de la última línea de una demostración es lo que se quiere demostrar
  - como ocurre con:  $(x + 1)^2 == x^2 + 2 \times x + 1$



### Conceptos básicos y motivación

Conceptos básicos  
relacionados con la  
Concurrencia

Motivación de la  
Programación Concurrente

### Modelo abstracto de la PC y consideraciones sobre el hardware

Modelo abstracto de la PC  
Consideraciones sobre el  
hardware

### Notaciones para PC

### Exclusión mutua y sincronización

Exclusión mutua

### Propiedades de los sistemas concurrentes

### Verificación de programas concurrentes

Introducción

Enfoque axiomático

## Ejemplo de demostración

- Ejemplo de una demostración formal:

1.  $(X + 1)^2 == (X + 1) \times (X + 1)$   
Definición de  $()^2$

2.  $(X + 1) \times (X + 1) == (X + 1) \times X + (X + 1) \times 1$   
Distributiva izquierda de  $\times$  con  $+$

3.  $(X + 1)^2 == (X + 1) \times X + (X + 1) \times 1$   
Sustituir línea 2 en 1

4.  $(X + 1) \times 1 == X + 1$  Ley identidad para 1

5.  $(X + 1) \times X == X \times X + 1 \times X$   
Distributiva derecha de  $\times$  con  $+$

6.  $(X + 1)^2 == X \times X + 1 \times X + X + 1$   
Sustituir líneas 4 y 5 en 3

7.  $1 \times X == X$  Ley identidad para 1

8.  $(X + 1)^2 == X \times X + X + X + 1$   
Sustituir línea 7 en 6

9.  $X \times X == X^2$  Definición de  $()^2$

10.  $X + X == 2 \times X$   $2 == 1 + 1$ , ley distributiva

11.  $(X + 1)^2 == X^2 + 2 \times X + 1$   
Sustituir líneas 9 y 10 en 8



### Conceptos básicos y motivación

Conceptos básicos  
relacionados con la  
Concurrencia

Motivación de la  
Programación Concurrente

### Modelo abstracto de la PC y consideraciones sobre el hardware

Modelo abstracto de la PC  
Consideraciones sobre el  
hardware

### Notaciones para PC

### Exclusión mutua y sincronización

Exclusión mutua

### Propiedades de los sistemas concurrentes

### Verificación de programas concurrentes

Introducción

Enfoque axiomático

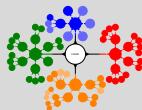
# Propiedades de seguridad y complección

- Un aserto  $\{P\}$  caracteriza un estado **aceptable** del programa, es decir, un estado que podría ser alcanzado por el programa si sus variables tomaran unos determinados valores

- Concepto de **Interpretación** de las fórmulas de un SLF:

$$\text{asertos} \rightarrow \{V, F\}$$

- Fórmula satisfascible
- Fórmula válida
- el aserto  $\{V\}$  caracteriza a todos los estados del programa ( o *precondición más débil*)
- el aserto  $\{F\}$  no se cumple por parte de ningún estado del programa ( o *precondición más fuerte*)
- **SLF seguro**:  $\text{asertos} \subseteq \{\text{hechos ciertos}\}$  – expresados como fórmulas de la lógica
- **SLF completo**:  $\{\text{hechos ciertos}\} \subseteq \{\text{asertos}\}$
- Los sistemas lógicos para demostración de programas no suelen poseer la propiedad de **complección**, pero a efectos prácticos es suficiente con la de **complección relativa** (todas las proposiciones son demostrables excepto si contienen expresiones aritméticas)



## Conceptos básicos y motivación

Conceptos básicos  
relacionados con la  
Concurrencia

Motivación de la  
Programación Concurrente

## Modelo abstracto de la PC y consideraciones sobre el hardware

Modelo abstracto de la PC  
Consideraciones sobre el  
hardware

## Notaciones para PC

## Exclusión mutua y sincronización

Exclusión mutua

## Propiedades de los sistemas concurrentes

## Verificación de programas concurrentes

Introducción

Enfoque axiomático





## Conceptos básicos y motivación

Conceptos básicos  
relacionados con la  
Concurrencia

Motivación de la  
Programación Concurrente

## Modelo abstracto de la PC y consideraciones sobre el hardware

Modelo abstracto de la PC  
Consideraciones sobre el  
hardware

## Notaciones para PC

## Exclusión mutua y sincronización

Exclusión mutua

## Propiedades de los sistemas concurrentes

## Verificación de programas concurrentes

Introducción

Enfoque axiomático

- Leyes distributivas:
  - $P \vee (Q \wedge R) = (P \vee Q) \wedge (P \vee R)$
  - $P \wedge (Q \vee R) = (P \wedge Q) \vee (P \wedge R)$
- Leyes de De Morgan:
  - $\neg(P \wedge Q) = \neg P \vee \neg Q$
  - $\neg(P \vee Q) = \neg P \wedge \neg Q$
- Eliminación-*And*:  $(P \wedge Q) \rightarrow P$
- Eliminación-*Or*:  $P \rightarrow (P \vee Q)$



Sirven para poder llevar a cabo las demostraciones de programas, ya que cada línea de la demostración es o bien un axioma o se deriva de la anterior mediante una regla de inferencia

- **Axioma:** fórmulas que sabemos que son ciertas en cualquier estado del programa
- **Regla de inferencia:** para derivar fórmulas ciertas a partir de los axiomas o de otras que se han demostrado ciertos previamente

Notación de una **Regla de inferencia:**

(*nombre de la regla*)  $\frac{H_1, H_2, \dots, H_n}{C}$

Un **aserto** (o *teorema*) es una fórmula con una interpretación cierta que pertenece al dominio de los hechos que queremos probar (o *dominio del discurso*)

Los **asertos** de nuestra lógica coinciden con las líneas o sentencias lógicas de las que se compone la demostración de un programa

Conceptos básicos y motivación

Conceptos básicos relacionados con la Concurrencia

Motivación de la Programación Concurrente

Modelo abstracto de la PC y consideraciones sobre el hardware

Modelo abstracto de la PC  
Consideraciones sobre el hardware

Notaciones para PC

Exclusión mutua y sincronización

Exclusión mutua

Propiedades de los sistemas concurrentes

Verificación de programas concurrentes

Introducción

Enfoque axiomático



- 1 **Axioma de la sentencia nula:**  $\{P\} \text{ NULL } \{P\}$ : si el aserto es cierto antes de ejecutarse esta sentencia, tendrá la misma interpretación cuando la sentencia termine
- 2 **Sustitución textual:**  $\{P_e^x\}$  es el resultado de sustituir la expresión  $e$  en cualquier aparición de la variable  $x$  en  $P$
- 3 **Axioma de asignación:**  $\{P_e^x\}_x = \{P\}$ : una asignación cambia solo el valor de la *variable objetivo*, el resto de las variables conservan los mismos valores.

Ejemplos:

- A  $\{V\} x = 5 \{x == 5\} \equiv \{P\}$  es un aserto pues:  
 $P_5^x \equiv \{x == 5\}_5^x \equiv \{5 == 5\} \equiv V$
- B  $\{x > 0\} x = x + 1 \{x > 1\} \equiv \{P\}$  es un aserto pues:  
 $P_{x+1}^x \equiv \{x > 1\}_{x+1}^x \equiv \{x + 1 > 1\} \equiv \{x > 0\}$

Reglas para *conectar* los triples en las demostraciones:

- 4 **Regla de la consecuencia (1):** 
$$\frac{\{P\}S\{Q\}, \{Q\} \rightarrow \{R\}}{\{P\}S\{R\}}$$

Conceptos básicos y motivación

Conceptos básicos relacionados con la Concurrencia

Motivación de la Programación Concurrente

Modelo abstracto de la PC y consideraciones sobre el hardware

Modelo abstracto de la PC  
Consideraciones sobre el hardware

Notaciones para PC

Exclusión mutua y sincronización

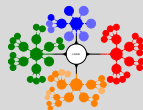
Exclusión mutua

Propiedades de los sistemas concurrentes

Verificación de programas concurrentes

Introducción

Enfoque axiomático



① Regla de la consecuencia (2):  $\frac{\{R\} \rightarrow \{P\}, \{P\} S \{Q\}}{\{R\} S \{Q\}}$

② Regla de la composición:  $\frac{\{P\} S_1 \{Q\}, \{Q\} S_2 \{R\}}{\{P\} S_1; S_2 \{R\}}$ : para obtener la pre y pos-condición de 2 sentencias juntas

③ Regla del IF:  $\frac{\{P\} \wedge \{B\} S_1 \{Q\}, \{P \wedge \neg B\} S_2 \{Q\}}{\{P\} \text{if } B \text{ then } S_1 \text{ else } S_2 \text{ fi } \{Q\}}$

④ Regla de la iteración:  $\frac{\{I \wedge B\} S \{I\}}{\{I\} \text{while } B \text{ do } S \text{ enddo } \{I \wedge \neg B\}}$ : podrá iterar un número arbitrario de veces (incluso 0); el *invariante* I se satisface antes y después de cada iteración

Conceptos básicos y motivación

Conceptos básicos relacionados con la Concurrency

Motivación de la Programación Concurrente

Modelo abstracto de la PC y consideraciones sobre el hardware

Modelo abstracto de la PC  
Consideraciones sobre el hardware

Notaciones para PC

Exclusión mutua y sincronización

Exclusión mutua

Propiedades de los sistemas concurrentes

Verificación de programas concurrentes

Introducción

Enfoque axiomático

## Verificación de sentencias concurrentes

- La ejecución concurrente de los procesos hace que el SLF deje de ser seguro: *problema de la interferencia*

```
y=0; z=0;
cobegin
  x= y+z || y=1; z=2
coend;
```

estados del programa	valores de x
x==y==z==0	0
x==y==1, z==0	1
x==3, y==1, z==2	3
-	2

### ¿Cómo evitar la interferencia?

- Evaluación de expresiones que no hacen referencia a variables modificadas concurrentemente

```
{x==0; y==0}
x= 0; y=0;
cobegin x= x+1 || y= y+1 coend;
z=x+y
{x==1, y==1, z==2}
```

- ¡Condición demasiado restrictiva!: casi ningún programa la cumpliría.



#### Conceptos básicos y motivación

Conceptos básicos  
relacionados con la  
Concurrencia

Motivación de la  
Programación Concurrente

#### Modelo abstracto de la PC y consideraciones sobre el hardware

Modelo abstracto de la PC  
Consideraciones sobre el  
hardware

#### Notaciones para PC

#### Exclusión mutua y sincronización

Exclusión mutua

#### Propiedades de los sistemas concurrentes

#### Verificación de programas concurrentes

Introducción

Enfoque axiomático



## Conceptos básicos y motivación

Conceptos básicos  
relacionados con la  
Concurrencia

Motivación de la  
Programación Concurrente

## Modelo abstracto de la PC y consideraciones sobre el hardware

Modelo abstracto de la PC  
Consideraciones sobre el  
hardware

## Notaciones para PC

## Exclusión mutua y sincronización

Exclusión mutua

## Propiedades de los sistemas concurrentes

## Verificación de programas concurrentes

Introducción

Enfoque axiomático

Acción atómica elemental  $\langle \dots \rangle$

Propiedad **como máximo una vez**: “*La evaluación de una expresión por un proceso concurrente se hace de forma atómica si las variables que comparte son leídas o escritas por un único proceso*”

Ejemplo 1: *Evaluación de expresiones*

①  $x = 0; y = 0; \text{cobegin } x = y + 1 \parallel y = x + 1 \text{ coend};$   
¡Incorrecta!

②  $x = 0; y = 0; \text{cobegin } x = y + 1 \parallel y = z + 1 \text{ coend};$

③  $x = 0; y = 0; \text{cobegin } x = z + 1 \parallel y = x + 1 \text{ coend};$

**No interfieren. entre un aserto y una instrucción atómica  $a$ :**

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

## Regla de inferencia de la composición concurrente de sentencias

Si  $\{P_i\} S_i \{Q_i\}$  no interfieren, entonces se puede demostrar el triple:

$$\begin{array}{c} \{P_1 \wedge P_2 \dots \wedge P_n\} \\ \text{COBEGIN} \\ S_1 \parallel S_2 \parallel \dots \parallel S_n \\ \text{COEND} \\ \{Q_1 \wedge Q_2 \dots \wedge Q_n\} \end{array}$$

Si los asertos de los procesos no se invalidan entre sí, entonces su composición concurrente transforma la conjunción de sus precondiciones en la conjunción de sus postcondiciones:

$$\begin{array}{c} \{x == 0\} \\ \text{COBEGIN} \\ \langle x = x + 1; \rangle \parallel \langle x = x + 2 \rangle \\ \text{COEND} \\ \{x == 3\} \end{array}$$



### Conceptos básicos y motivación

Conceptos básicos  
relacionados con la  
Concurrencia

Motivación de la  
Programación Concurrente

### Modelo abstracto de la PC y consideraciones sobre el hardware

Modelo abstracto de la PC  
Consideraciones sobre el  
hardware

### Notaciones para PC

### Exclusión mutua y sincronización

Exclusión mutua

### Propiedades de los sistemas concurrentes

### Verificación de programas concurrentes

Introducción

Enfoque axiomático

# Regla de inferencia de la composición concurrente de sentencias -II

Traza de la demostración libre de interferencias:

$$\begin{array}{c} \{x == 0\} \\ \text{COBEGIN} \\ \{x == 0 \vee x == 2\} \quad \{x == 0 \vee x == 1\} \\ \quad < x = x + 1; > \parallel < x = x + 2 > \\ \{x == 1 \vee x == 3\} \quad \{x == 2 \vee x == 3\} \\ \text{COEND} \\ \{x == 3\} \end{array}$$

Aplicando la *regla de la no interferencia* a la expresión de asertos concurrentes anterior, la precondition de la sentencia `cobegin` ha de ser equivalente a la conjunción de las preconditiones de sus sentencias componentes  $(x == 0 \vee x == 2) \wedge (x == 0 \vee x == 1) \equiv (x == 0)$  Así mismo ocurre con la poscondición de `coend`  $(x == 1 \vee x == 3) \wedge (x == 2 \vee x == 3) \equiv (x == 3)$



## Conceptos básicos y motivación

Conceptos básicos  
relacionados con la  
Concurrencia

Motivación de la  
Programación Concurrente

## Modelo abstracto de la PC y consideraciones sobre el hardware

Modelo abstracto de la PC  
Consideraciones sobre el  
hardware

## Notaciones para PC

### Exclusión mutua y sincronización

Exclusión mutua

### Propiedades de los sistemas concurrentes

### Verificación de programas concurrentes

Introducción

Enfoque axiomático



Invariante global: se trata de un predicado que referencia variables globales del programa que se demuestra cierto en el estado inicial de cada proceso y se mantiene cierto ante cualquier *asignación* dentro de los procesos

Ejemplo de invariante global:

- En una solución correcta del Productor-Consumidor, un invariante global sería:

$$\text{consumidos} \leq \text{producidos} \leq \text{consumidos} + 1$$

- Condición de invariante global (IG):** Dado un aserto  $I$  definido a partir de las *variables compartidas* entre los procesos de un programa concurrente, puede ser considerado un IG válido si y sólo si:
  - Es cierto para los valores iniciales de las variables
  - Se mantiene cierto después de la ejecución de cada *instrucción atómica*  $a$  del programa:  $\{I \wedge \text{pre}(a)\} a \{I\}$



Conceptos básicos y  
motivación

Conceptos básicos  
relacionados con la  
Concurrencia

Motivación de la  
Programación Concurrente

Modelo abstracto de la  
PC y consideraciones  
sobre el hardware

Modelo abstracto de la PC  
Consideraciones sobre el  
hardware

Notaciones para PC

Exclusión mutua y  
sincronización

Exclusión mutua

Propiedades de los  
sistemas concurrentes

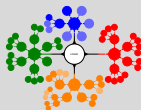
Verificación de  
programas  
concurrentes

Introducción

Enfoque axiomático

Para más información, ejercicios, bibliografía adicional, se puede consultar:

- 1.1. [Conceptos básicos y Motivación](#), Palma (2003), Capel-Rodríguez Valenzuela (2012): capítulo 1
- 1.2. [Modelo abstracto y Consideraciones sobre el hardware](#), Ben-Ari (2006), capítulo 2. Andrews (2000), Palma (2003), Capel-Rodríguez Valenzuela (2012): capítulo 1
- 1.3. [Exclusión mutua y sincronización](#), Palma (2003), capítulo 1. Capel-Rodríguez Valenzuela (2012): capítulo 2
- 1.4. [Propiedades de los Sistemas Concurrentes](#), Palma (2003), Capel-Rodríguez Valenzuela (2012): capítulo 1
- 1.5. [Verificación de Programas concurrentes](#), Andrews (2000), capítulo 2. Capel-Rodríguez Valenzuela (2012): capítulo 1



### Conceptos básicos y motivación

Conceptos básicos relacionados con la Concurrency

Motivación de la Programación Concurrente

### Modelo abstracto de la PC y consideraciones sobre el hardware

Modelo abstracto de la PC  
Consideraciones sobre el hardware

### Notaciones para PC

### Exclusión mutua y sincronización

Exclusión mutua

### Propiedades de los sistemas concurrentes

### Verificación de programas concurrentes

Introducción

Enfoque axiomático