# Clases de instrucciones

Un programa concurrente está formado por un conjunto de programas secuenciales. La programación secuencial se expresa con 3 instrucciones básicas. **Asignación**, **alternativa** e **iteración**. Se requiere una clase de instrucción para representar la concurrencia.

Asignación: x= y , x = x +1 etc.

Alternativa: If normal y los if de alternativa múltiple

Iteración: While (cond) S, for de alternativa múltiple (do), for all.

Para los for de alternativa múltiple si las condiciones se solapan se elije de forma no determinística.

Para programación concurrente se utiliza la sentencia **co**, la cual ejecuta S tareas concurrentemente. La ejecución del co termina cuando terminan todas las tareas.

También se utiliza **Process**, que se usa para definir un proceso independiente (process A { sentencias } ) o n procesos independientes.

La diferencia entre ambos radica en que **process** ejecuta en background (ya se están ejecutando desde que comienza el programa), mientras que la sentencia co espera a que el proceso creado por la sentencia co termine antes de ejecutar la siguiente sentencia.

# Acciones atómicas y sincronización

Atomicidad de grano fino

El **estado** de un programa concurrente es el valor que tienen todas las variables en un cierto instante de tiempo (incluye compartidas, privadas, registros, etc.).

Cada proceso ejecuta un conjunto de sentencias, las cuales están compuestas por una o más **acciones atómicas**. Una **acción atómica** hace una transformación de estado indivisible (por ejemplo, modificar una variable).

Al ejecutar un programa concurrente se produce un **intercalado** de las acciones atómicas ejecutadas por los procesos individuales. La ejecución de un programa con un intercalado particular se denomina **Historia** (o **trace**) de un programa concurrente. Hay muchas historias posibles en un programa concurrente, pero no todas son válidas. La **interacción entre procesos** determina cuáles historias son válidas (o permitidas).

Se debe asegurar un orden temporal entre las acciones que ejecutan los procesos para que las historias sean válidas. La **sincronización por condición** permite restringir las historias de un programa para asegurar el orden temporal necesario.

Las acciones atómicas de **grano fino** deben ser implementadas por hardware.

Tabla

Descripción generada automáticamente con confianza media

No podemos confiar en la intuición para determinar la ejecución de un programa concurrente

En la mayoría de los sistemas el tiempo absoluto no es importante, ya que los sistemas son actualizados con componentes más rápidos. La corrección entonces no debe depender del tiempo absoluto, solo las secuencias son importantes.

Puede haber distintas órdenes en que se ejecutan las instrucciones de los diferentes procesos, y los programas deben ser correctos para todos ellos.

Si una expresión e en un proceso no referencia una variable alterada por otro proceso la evaluación será atómica independientemente de que necesite ejecutar varias acciones atómicas de grano fino.

Se necesita un requerimiento más débil, no solo las expresiones atómicas independientes, sino también el acceso a variables compartidas

**Referencia crítica:** Referencia a una variable que es modificada por otro proceso. Asumimos que toda referencia crítica es a una variable simple leída y escrita atómicamente.

Propiedad de “a lo sumo una vez”

Una sentencia x = e satisface la propiedad de “**a lo sumo una vez**” si:

1. e contiene a lo sumo una referencia crítica y x no es referenciada por otro proceso
2. si e no contiene referencias críticas, pero x puede ser leída por otro proceso.

Una expresión e que no está en una sentencia de asignación satisface la propiedad si no contiene más de una referencia crítica.

**Puede haber a lo sumo una variable compartida y puede ser referenciada a lo sumo una vez**

Si una sentencia de asignación cumple la propiedad ASV entonces su ejecución parece atómica, pues la variable compartida será leída o escrita solo una vez.

Especificación de la sincronización

Si una expresión o asignación no satisface ASV con frecuencia entonces necesitamos ejecutarla atómicamente . En general, es necesario ejecutar secuencias de sentencias como una única acción atómica.

Esto se llama hacer **sincronización por exclusión mutua ,** es decir ejecutar una secuencia de sentencias que estoy seguro de que se van a ejecutar sin que ningún proceso interfiera entre medio.

De esto sale un mecanismo de sincronización para construir una acción atómica de **grano grueso** como secuencia de acciones atómicas de grano fino que aparecen como indivisibles (< e > y < await (B) S >).

Await es una sentencia de alto poder expresivo, pero el costo de implementación de la forma general de await es alto.

# Propiedades y Fairness

Una **propiedad** de un programa concurrente es un atributo verdadero en cualquiera de las historias de ejecución de este. Toda propiedad puede ser formulada en términos de dos clases:

* **seguridad**
* Nada malo le ocurre a un proceso
* Una falla de seguridad indica que algo anda mal
* Un ejemplo es la exclusión mutua o la ausencia de interferencia entre procesos
* **Vida**
* Eventualmente ocurre algo bueno con una actividad
* Una falla de vida indica que las cosas dejan de ejecutar
* Un ejemplo es la terminación, que un mensaje llegue a destino, que un proceso eventualmente alcance su SC, entre otros.

**Total correctness** indica que el programa siempre termina y termina bien.

**Fairness:** Trata de garantizar que los procesos tengan chance de avanzar sin importar lo que haga el resto.

Una acción atómica en un proceso es **elegible** si es la próxima acción atómica en el proceso que será ejecutada. Si hay varios procesos hay varias acciones atómicas elegibles

Cual es la próxima acción en ejecutarse está determinado por la **política de scheduling.**

**Fairness incondicional:** Una política de scheduling es incondicionalmente fair si toda acción atómica incondicional que es elegible eventualmente se ejecuta. Round Robin por ejemplo es incondicionalmente fair en monoprocesador.

**Fairness débil:** Una política de scheduling es débilmente fair si es incondicionalmente fair y toda acción atómica condicional que se vuelve elegible eventualmente se ejecuta, asumiendo que se vuelve true y permanece true hasta que el proceso que la ejecuta la ve.

Una sentencia await no se puede asegurar que sea débilmente fair, ya que se puede cambiar el valor de la condición de true a false o viceversa mientras un proceso está demorado.

**Fairness fuerte:** Una política es fuertemente fair si es incondicionalmente fair y toda acción atómica condicional elegible es ejecutada porque su condición se vuelve true con infinita frecuencia.

No es simple tener una política práctica que sea fuertemente fair. Round-Robin por ejemplo es práctica, pero no es fuertemente fair.