¿Qué es que observamos?

- El algoritmo es no-determinista,
- en el sentido que no se sabe de antemano en qué orden (en un procesador o en un conjunto de procesadores) se van a ejecutar las instrucciones,
- o más preciso, cómo se van a intercalar las instrucciones atómicas de ambos procesos.
- El no-determinismo puede provocar situaciones con errores, es decir, el fallo ocurre solamente si las instrucciones se ejecutan en un orden específico.
- El resultado del programa no es predecible, y lo peor es, a veces es correcto.
- Desde el punto de vista de concurrencia tiene varios *problemas*.

funcionamiento correcto I

Generalmente se dice que un programa es correcto, si dada una entrada, el programa produce los resultados deseados.
Más formal:

- Sea $P(x)$ una propiedad de una variable x de entrada (aquí el símbolo x refleja cualquier conjunto de variables de entradas).
- Sea $Q(x, y)$ una propiedad de una variable x de entrada y de una variable y de salida.

funcionamiento correcto II

Se define dos tipos de funcionamiento correcto de un programa:

funcionamiento correcto parcial:

dada una entrada a, si $P(a)$ es verdadero, y si se lanza el programa con la entrada a, entonces si el programa termina habrá calculado b y $Q(a, b)$ también es verdadero.

funcionamiento correcto total:

dado una entrada a, si $P(a)$ es verdadero, y si se lanza el programa con la entrada a, entonces el programa termina y habrá calculado b con $Q(a, b)$ siendo también verdadero.

funcionamiento correcto III

Para un programa secuencial existe solamente un orden total de las instrucciones atómicas (en el sentido que un procesador secuencial siempre sigue el mismo orden de las instrucciones... bueno, es mentira..., hay *out-of-order and speculative execution*), mientras que para un programa concurrente puedan existir varios órdenes.
Por eso se tiene que exigir:

funcionamiento correcto concurrente:

un programa concurrente funciona correctamente, si el resultado $Q(x, y)$ no depende del orden de las instrucciones atómicas entre todos los órdenes posibles.

Funcionamiento correcto IV

Entonces:

- Se debe asumir que los hilos pueden intercalarse en cualquier punto en cualquier momento.
- Los programas no deben estar basados en la suposición de que habrá un intercalado específico entre los hilos por parte del planificador (que conmuta los procesos).

funcionamiento correcto V

- Para comprobar si un programa concurrente es *incorrecto* basta con encontrar una sola intercalación de instrucciones que nos lleva en un fallo.
- Para comprobar si un programa concurrente es *correcto* hay que comprobar que no se produce ningún fallo en ninguna de las intercalaciones posibles.

comprobación exhaustiva no es práctico

- El número de posibles intercalaciones de los procesos en un programa concurrente crece exponencialmente con el número de unidades que maneja el planificador y líneas por intercalar.
- ¿Cuántos son? (Ayuda: calcular las combinaciones posibles de una lista dentro de otra)
- Por eso es prácticamente imposible comprobar con la mera enumeración si un programa concurrente es correcto bajo todas las ejecuciones posibles.
- En la argumentación hasta ahora era muy importante que las instrucciones se ejecutaran de forma atómica, es decir, sin interrupción ninguna.
- Por ejemplo, se observa una gran diferencia si el procesador trabaja directamente en memoria o si trabaja con registros.

dependencia de atomicidad I

Si `increment` es atómico:

```
P1:   inc N
P2:   inc N

P2:   inc N
P1:   inc N
```

Se observa: las dos intercalaciones posibles producen el resultado correcto.

dependencia de atomicidad II

Si `increment` no es atómico:

```
P1:  load  R1,N
P2:  load  R2,N
P1:  inc   R1
P2:  inc   R2
P1:  store R1,N
P2:  store R2,N
```

Es decir, existe una intercalación que produce un resultado falso.
Ejemplos de Java:

- accesos a variables con más de 4 bytes no son atómicos.
- el operador ++ no es atómico.
- o en otras palabras: lectura y escritura de flotantes no es linearizable en Java

Una mejora

Nuestro programa con dos hilos ejecutaba muy lento (e incorrecto). Hacemos una primera modificación: usamos como condición para `q` que sea mayor que será.

```
  public void run() {
    try {
      System.out.println("starting worker... "+id);
      Int minusOne=new Int(-1);
      while(q.Get()>0) {
        r.Add(p);
        q.Add(minusOne);
      }
    }
    catch(Exception E) { System.out.println("??? "+id);
    finally { System.out.println("exiting... "+id); }
  }
}
```

Nota que sigue correcto en su version secuencial.

¿Y?

- ¿El algoritmo concurrente de multiplicación con dos hilos de arriba es correcto? (correcto parcial? correcto total?)
- ¿Cómo lo compruebas?
- ¿Te ocurre un algoritmo concurrente correcto que funcione con varios hilos?
- ¿Te ocurre un algoritmo concurrente correcto y eficiente, es decir, donde varios hilos juntos son más rápido que uno sólo?