Estudio y desarrollo de técnicas para el testing de programas concurrentes

CABS: traducción de C a ABS

Marco Antonio Garrido Rojo[1]

17 de junio de 2018

[1] https://github.com/MaSteve/CABS-Slides

Introducción: historia y origen

- En un principio, la mayoría de sistemas solo podían ejecutar una única secuencia de instrucciones a la vez.
- Con el avance de los ordenadores durante las últimas décadas del siglo pasado, se permite la ejecución simultánea de varios hilos de ejecución.
- Hoy en día es común el uso de la programación concurrente en la mayoría de aplicaciones y programas de todo ámbito.

Introducción: inconvenientes

- La ejecución paralela conlleva riesgos adicionales no presentes en los programas secuenciales.
- El principal problema que presenta es la presencia de una memoria compartida sobre la que los distintos hilos realizan modificaciones en cualquier momento.
- Asociado a esto, pueden ocurrir deadlocks, carreras de datos o comportamientos impredecibles.

Introducción: soluciones

- Podemos evitar que ocurran estableciendo restricciones mediante el uso de semáforos y cerrojos o implementaciones de algoritmos de más bajo nivel como el tie-breaker.
- El uso de estos mecanismos puede ser bastante **complejo**.

Introducción: soluciones

```
bool in1 = false, in2 = false;
int last = 1;

process CS1 {
  while (true) {
    last = 1; in1 = true;     /* entry protocol */
    while (in2 and last == 1) skip;
    critical section;
    in1 = false;              /* exit protocol */
    noncritical section;
  }
}

process CS2 {
  while (true) {
    last = 2; in2 = true;     /* entry protocol */
    while (in1 and last == 2) skip;
    critical section;
    in2 = false;              /* exit protocol */
    noncritical section;
  }
}
```

Figura: Algoritmo tie-breaker para dos procesos. (Andrews')

Introducción: interleavings

- En este terreno se sigue investigando para encontrar un modo definitivo que permita solucionar los problemas o detectarlos.
- Uno de los primeros asuntos estudiados es determinar el estado o estados finales a los que se llega ejecutando un programa concurrente.

Introducción: interleavings

```
1  int var;
2
3  proc f1() {
4     var := 1;
5  }
6
7  proc f2() {
8     var := 2;
9  }
10
```

Introducción: interleavings

- Explosión exponencial en el número de interleavings en proporción al número de hilos y al número de instrucciones.
- No siempre hay una correlación con el número de estados finales.
- No todos los interleavings tienen la misma importancia y algunos de ellos son equivalentes entre sí.

Introducción: interleavings

```
1  int var1;
2  int var2;
3
4  proc f1() {
5     var1 := 1;   # 1
6     var1 := 2;   # 2
7  }
8
9  proc f2() {
10    var2 := 2;   # 3
11    var2 := 1;   # 4
12 }
13
```

Introducción: grano

- ¿Cuál es la atomicidad?
  - Asignaciones atómicas: grano grueso.
  - ¿Lectura de valores?: grano fino.
  - ¡Secciones de código enteras!: . . .
- ¿De qué depende el grano? ¿Qué ocurre si no hay memoria compartida?

Introducción: actores

- Cada objeto ejecuta sus tareas de forma concurrente con respecto a las del resto de objetos.
- Una única tarea a la vez por objeto. El resto de tareas esperan en una cola cuyo orden no es determinable.
- El paso de mensajes indica qué método desea ejecutar un objeto (pudiendo ser uno propio o perteneciente a otro objeto).
- Dependiendo del tipo de llamada, una tarea puede dejar paso a otra si aún no cuenta con los valores necesarios para proseguir.
- Se trata de una concurrencia donde el scheduler es non-preemptive.

Introducción: ventajas

- Reducción notable del número de interleavings: importante cuando se aplican técnicas de testing sistemático.
- Presente en lenguajes como Erlang, Scala y **ABS**.
- Cuenta con una amplia cantidad de herramientas, entre ellas, **SYCO**, que implementa **DPOR** para obtener los posibles resultados finales de un programa, reduciendo los interleavings redundantes.

Introducción: objetivo

Introducción: soluciones

- En este terreno se sigue investigando para encontrar un modo definitivo que permita solucionar los problemas o detectarlos.
- Cada vez es más necesario el uso de técnicas de validación como el testing.
- En esta línea es necesario detectar la redundancias de las distintas trazas de ejecución de un programa para crear tests más efectivos.

ABS, SYCO y aPET

- ABS es un lenguaje de modelado, orientado a objetos y concurrente que usa un modelo de paso de mensajes entre actores.
- Emplea llamadas asíncronas a objetos aislados unos de otros en términos de memoria.
- Cada objeto es capaz de gestionar un mensaje a la vez (e.g. reducción de la explosión de estados).
- Es posible usarlo con herramientas especializadas como SYCO y aPET para analizar implementaciones y generar tests para ellas.

ABS y sus inconvenientes

- Sintaxis compleja: interfaces y clases que las implementan, llamadas asíncronas...
- Boilerplate
- Alejado de las implementaciones reales (e.g. modelado e implementación separados).

¿Qué es CABS?

La idea principal es tener un lenguaje de programación sencillo que

- Tenga una sintaxis similar a C.
- Permita ejecuciones concurrentes de grano fino de forma simple (mismo comportamiento que pthread).
- Tenga un tipado estricto y estático.
- Paradigma imperativo con funciones y arrays.
- Que pueda usar las mismas herramientas que ABS.