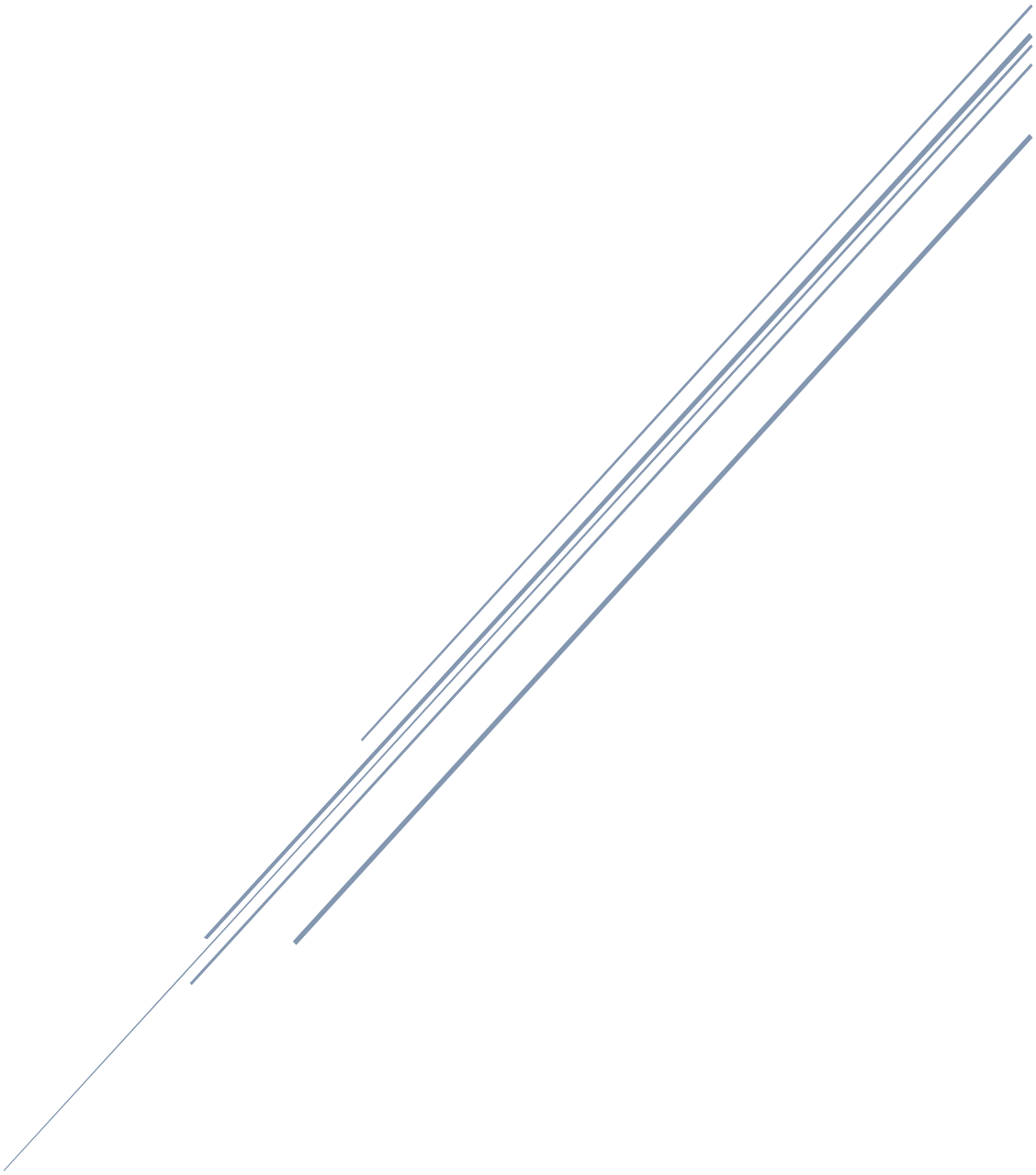
SEMANTICA THREADS

MFC – Modelos Formales de Computación



MITSS
Sergi Sanz Carreres

Índice

2.- Ejercicios a resolver

2.1. ¿Qué ocurre en la ejecución del siguiente programa?

```
let x = 0
in {
  spawn(x := x + 1) ;
  spawn(x := x + 1) ;
  x
}
```

Una vez ejecutado el programa anterior obtenemos los siguientes resultados:

```
search in CONCURRENT-FUN-SEMANTICS : eval (let x = 0 in {spawn(x := x + 1) ;
    spawn(x := x + 1) ; x}) =>! X:[ValueList] .

Solution 1 (state 4)
states: 8  rewrites: 55 in 0ms cpu (0ms real) (~ rewrites/second)
X:[ValueList] --> int(0)

Solution 2 (state 15)
states: 22  rewrites: 139 in 0ms cpu (0ms real) (~ rewrites/second)
X:[ValueList] --> int(1)

Solution 3 (state 25)
states: 26  rewrites: 162 in 0ms cpu (0ms real) (~ rewrites/second)
X:[ValueList] --> int(2)

No more solutions.
```

Nos devuelve las tres soluciones que pueden producirse que los dos primeros spawn no acaben de ejecutarse, que uno de ellos se ejecute y que ambos se ejecuten por completo.

2.2. ¿Qué ocurre en la ejecución del siguiente programa?

```
let x = 0
in {
  spawn(acquire lock(1) ; x := x + 1 ; release lock(1)) ;
  spawn(acquire lock(1) ; x := x + 1 ; release lock(1)) ;
  x
}
```

Una vez ejecutado el programa anterior obtenemos los siguientes resultados:

```
search in CONCURRENT-FUN-SEMANTICS : eval (let x = 0 in {spawn(acquire lock(1)
    ; x := x + 1 ; release lock(1)) ; spawn(acquire lock(1) ; x := x + 1 ;
    release lock(1)) ; x}) =>! X:[ValueList] .

Solution 1 (state 4)
states: 7  rewrites: 60 in 1ms cpu (0ms real) (60000 rewrites/second)
X:[ValueList] --> int(0)

Solution 2 (state 13)
states: 17  rewrites: 152 in 1ms cpu (0ms real) (152000 rewrites/second)
X:[ValueList] --> int(1)

Solution 3 (state 22)
states: 23  rewrites: 205 in 1ms cpu (0ms real) (205000 rewrites/second)
X:[ValueList] --> int(2)

No more solutions.
```

Hace las mismas operaciones que en el ejercicio anterior, pero en este caso al hacer uso de semáforos se evitan condiciones de carrera, de forma que se garantiza que se produzca el calculo de forma correcta.

2.3. ¿Qué ocurre en la ejecución del siguiente programa?

```
let rec f(x,y) = spawn(f(x,y) ; x := x + 1 ; y := 1)
in
 let x = 0 and a = 0
 in {
     f(x,a) ;
     while (a == 0) {} ;
     x
 }
```

Una vez ejecutado el programa anterior obtenemos los siguientes resultados:

```
search in CONCURRENT-FUN-SEMANTICS : eval (let rec f (x,y) = spawn(f (x,y) ; x
    := x + 1 ; y := 1) in let x = 0 and a = 0 in {f (x,a) ; while a == 0 {} ;
    x}) =>! X:[ValueList] .
```

Se produce un bucle infinito, debido a que llama de forma recursiva a f().

2.4. ¿Qué ocurre en la ejecución del siguiente programa?

```
let rec f(x,y) = { f(x,y) ; x := x + 1 ; y := 1 }
in
 let x = 0 and a = 0
 in {
     f(x,a) ;
     while (a == 0) {} ;
     x
 }
```

Una vez ejecutado el programa anterior obtenemos los siguientes resultados:

```
search in CONCURRENT-FUN-SEMANTICS : eval (let rec f (x,y) = {f (x,y) ; x := x
    + 1 ; y := 1} in let x = 0 and a = 0 in {f (x,a) ; while a == 0 {} ; x})
    =>! X:[ValueList] .
```

Al igual que en el ejercicio anterior, al producirse una llamada recursiva a f() hace que se cree un bucle infinito.

2.5. ¿Qué ocurre en la ejecución del siguiente programa?

```
let rec f(x,y) = spawn( x := x + 1 ; y := 1 )
in
 let x = 0 and a = 0
 in {
     f(x,a) ;
     while (a == 0) {} ;
     x
 }
```

Una vez ejecutado el programa anterior obtenemos los siguientes resultados:

```
search in CONCURRENT-FUN-SEMANTICS : eval (let rec f (x,y) = spawn(x := x + 1 ;
    y := 1) in let x = 0 and a = 0 in {f (x,a) ; while a == 0 {} ; x}) =>! X:[
    ValueList] .

No solution.
```

Nos devuelve que no existe solución porque Maude reconoce la presencia de un bucle infinito, ya que en f(x,a) se pasa por valor y no por referencia, creando un bucle infinito.

2.6. ¿Qué ocurre en la ejecución del siguiente programa?

```
let rec f(x,y) = { x := x + 1 ; y := 1 }
in
 let x = 0 and a = 0
 in {
      f(x,a) ;
      while (a == 0) {} ;
      x
 }
```

Una vez ejecutado el programa anterior obtenemos los siguientes resultados:

```
search in CONCURRENT-FUN-SEMANTICS : eval (let rec f (x,y) = {x := x + 1 ; y :=
    1} in let x = 0 and a = 0 in {f (x,a) ; while a == 0 {} ; x}) =>! X:[
    ValueList] .

No solution.
```

Al igual que en el ejemplo anterior se produce un bucle infinito, aunque en este caso se ha eliminado la función spawn pero se sigue realizando paso por valor y no referencia, produciéndose el bucle infinito.

2.7. ¿Qué ocurre en la ejecución del siguiente programa?

```
let x = 0 and a = 0
and f(x,y) = spawn(x := x + 1 ; y := 1)
in {
     f(x,a) ;
     while (a == 0) {} ;
     x
}
```

Una vez ejecutado el programa anterior obtenemos los siguientes resultados:

```
search in CONCURRENT-FUN-SEMANTICS : eval (let x = 0 and a = 0 and f (x,y) =
    spawn(x := x + 1 ; y := 1) in {f (x,a) ; while a == 0 {} ; x}) =>! X:[
    ValueList] .

No solution.
```

Al igual que en el ejemplo anterior se produce un bucle infinito, debido a que se realiza el paso por valor y no referencia, la única diferencia es que en este caso las variables son globales.

2.8. ¿Qué ocurre en la ejecución del siguiente programa?

```
let rec x = 0 and a = 0
and f() = spawn( x := x + 1 ; a := 1 )
in {
    f() ;
    while (a == 0) {} ;
    x
}
```

Una vez ejecutado el programa anterior obtenemos los siguientes resultados:

```
search in CONCURRENT-FUN-SEMANTICS : eval (let rec x = 0 and a = 0 and f () =
    spawn(x := x + 1 ; a := 1) in {f () ; while a == 0 {} ; x}) =>! X:[
    ValueList] .

Solution 1 (state 11)
states: 12  rewrites: 156 in 0ms cpu (0ms real) (~ rewrites/second)
X:[ValueList] --> int(1)

No more solutions.
```

En este caso la función f() modifica directamente el valor del contenido dentro de las variables ‘x’ y ‘a’, por lo tanto se incrementara en 1 el valor de x, de manera que se actualizara a 1, devolviendo ese valor y saliendo del bucle en caso de haber llegado al mismo.

2.9. ¿Qué ocurre en la ejecución del siguiente programa?

```
let rec x = 0 and a = 0
and f() = { x := x + 1 ; a := 1 }
in {
    f() ;
    while (a == 0) {} ;
    x
}
```

Una vez ejecutado el programa anterior obtenemos los siguientes resultados:

```
search in CONCURRENT-FUN-SEMANTICS : eval (let rec x = 0 and a = 0 and f () = {
    x := x + 1 ; a := 1} in {f () ; while a == 0 {} ; x}) =>! X:[ValueList] .

Solution 1 (state 11)
states: 12  rewrites: 109 in 0ms cpu (0ms real) (~ rewrites/second)
X:[ValueList] --> int(1)

No more solutions.
```

En este caso la función f() modifica el valor del contenido de ‘x’ y ‘a’ de manera secuencial de forma que se ejecuta primero f() actualizando el valor de ‘x’ a 1 y de ‘a’ a 1 también, por tanto no se cumple la condición del bucle.

2.10. ¿Qué ocurre en la ejecución del siguiente programa?

```
let rec x = 0 and a = 0
and f() = spawn( f(); x := x + 1 ; a := 1 )
in {
    f() ;
    while (a == 0) {} ;
    x
}
```

Una vez ejecutado el programa anterior obtenemos los siguientes resultados:

```
search in CONCURRENT-FUN-SEMANTICS : eval (let rec x = 0 and a = 0 and f () =
    spawn(f () ; x := x + 1 ; a := 1) in {f () ; while a == 0 {} ; x}) =>! X:[
    ValueList] .

Solution 1 (state 60)
states: 96  rewrites: 2148 in 2ms cpu (2ms real) (1074000 rewrites/second)
X:[ValueList] --> int(1)

Solution 2 (state 234)
states: 362  rewrites: 8944 in 10ms cpu (10ms real) (894400 rewrites/second)
X:[ValueList] --> int(2)

Solution 3 (state 843)
states: 1281  rewrites: 34002 in 43ms cpu (43ms real) (790744 rewrites/second)
X:[ValueList] --> int(3)

Solution 4 (state 2858)
states: 4249  rewrites: 120133 in 161ms cpu (161ms real) (746167
    rewrites/second)
X:[ValueList] --> int(4)

Solution 5 (state 9187)
states: 13480  rewrites: 401412 in 578ms cpu (578ms real) (694484
    rewrites/second)
X:[ValueList] --> int(5)

Solution 6 (state 28394)
states: 41105  rewrites: 1286285 in 2105ms cpu (2105ms real) (611061
    rewrites/second)
X:[ValueList] --> int(6)

Solution 7 (state 84881)
states: 121862  rewrites: 3986866 in 7472ms cpu (7473ms real) (533574
    rewrites/second)
X:[ValueList] --> int(7)
```

El hilo principal llama a la función f() que a su vez realiza la creación de más hilos que ejecutan la función f(). De manera que los hilos creados van devolviendo su x, debido a este motivo cada vez se produce un incremento de 1 en x.