Clase 8: Preparación para el Examen 1

1. (*Allen B. Downey, The Little Book of Semaphores, Second Edition, Version 2.2.1, 2016*) *From Chapter 3 Basic synchronization patterns. 3.6 Barrier*:

El código del patrón de la barrera, en su primera versión de la página 25, se puede modelar con el programa `lbs_3.6.2a.pml`:

```
$ cat -n lbs_3.6.2a.pml
     1 /*  The Little Book of Semaphores (2.2.1)
     2     by A. Downey
     3
     4     Chapter 3. Basic synchronization patterns
     5
     6     3.6 Barrier
     7     3.6.2 Barrier non-solution
     8 */
     9
    10 #define N 3
    11
    12 #define wait(sem)   atomic { sem > 0; sem-- }
    13 #define signal(sem) sem++
    14
    15 byte count=0, mutex=1, barrier=0
    16
    17 proctype P() {
    18     do
    19     ::  wait(mutex)
    20             count++
    21         signal(mutex)
    22         if
    23         :: count == N ->
    24                           signal(barrier)
    25         :: else
    26         fi
    27         wait(barrier)
    28     od
    29 }
    30
    31 init {
    32     byte i
    33
    34     atomic {
    35         for (i: 1 .. N) {
    36             run P()
    37         }
    38     }
    39 }
```

```
$ spin lbs_3.6.2a.pml | expand
      timeout
#processes: 4
                count = 4
                mutex = 1
                barrier = 0
 41:    proc  3 (P:1) lbs_3.6.2a.pml:27 (state 13)
 41:    proc  2 (P:1) lbs_3.6.2a.pml:27 (state 13)
 41:    proc  1 (P:1) lbs_3.6.2a.pml:27 (state 13)
 41:    proc  0 (:init::1) lbs_3.6.2a.pml:39 (state 11) <valid end state>
4 processes created
```

1a) ¿Por qué es valor final de la variable `count` es 4 si esta variable se incrementa solamente por 3 procesos?

1b) ¿Qué significa y por qué sucede `timeout`?

2. El código anterior **no siempre** produce *deadlock*. **Modifique el código anterior** obteniendo el programa `lbs_3.6.3a.pml` y encuentre el escenario sin *deadlock* más corto posible.

Nota: El escenario buscado significa que los 3 procesos **diferentes** pasan la barrera y no que un proceso la pasa 2 veces. Para esto podría ser necesaria una identificación de cada proceso.

3. (*Bridge Crossing Problem*) El problema de cruce de puente para 3 personas se puede modelar y resolver con el código presentado en el programa `bridge3.pml`.

```
$ spin -run -e bridge3.pml | expand
pan:1: assertion violated (t==7) (at depth 3)
pan: wrote bridge3.pml1.trail
pan: wrote bridge3.pml2.trail
pan: wrote bridge3.pml3.trail

...
Full statespace search for:
        never claim             - (none specified)
        assertion violations    +
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      +

State-vector 16 byte, depth reached 5, errors: 3
...
```

Los tiempos de las soluciones posibles son: 8, 9 y 15 minutos para 3 (2*n*-3) viajes:

```
$ spin -t1 bridge3.pml | expand
      (a,b)-->, t = 2
      <--a, t = 3
      (a,c)-->, t = 8
      tiempo total = 8
...

$ spin -t2 bridge3.pml | expand
      (a,b)-->, t = 2
      <--b, t = 4
      (b,c)-->, t = 9
      tiempo total = 9
...

$ spin -t3 bridge3.pml | expand
      (a,c)-->, t = 5
      <--c, t = 10
      (b,c)-->, t = 15
      tiempo total = 15
...
```

Para 4 personas el código se presenta en el programa `bridge4.pml`.

```
$ spin -run -e bridge4.pml | expand
pan:1: assertion violated (t==7) (at depth 5)
pan: wrote bridge4.pml1.trail
```

```
pan: wrote bridge4.pml2.trail
pan: wrote bridge4.pml3.trail
pan: wrote bridge4.pml4.trail
pan: wrote bridge4.pml5.trail
pan: wrote bridge4.pml6.trail
pan: wrote bridge4.pml7.trail
pan: wrote bridge4.pml8.trail
pan: wrote bridge4.pml9.trail
pan: wrote bridge4.pml10.trail
pan: wrote bridge4.pml11.trail
pan: wrote bridge4.pml12.trail
pan: wrote bridge4.pml13.trail
pan: wrote bridge4.pml14.trail
pan: wrote bridge4.pml15.trail

...
Full statespace search for:
        never claim             - (none specified)
        assertion violations    +
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      +

State-vector 16 byte, depth reached 7, errors: 15
...
```

Son 15 soluciones para 5 (2*n*-3) viajes con los tiempos: 17, 19, 20, 21, 23, 24, 26, 27, 30, 33, 34, 36, 37, 40, 50:

```
$ spin -t1 bridge4.pml | expand
      (a,b)-->, t = 2
      <--a, t = 3
      (a,c)-->, t = 8
      <--a, t = 9
      (a,d)-->, t = 19
      tiempo total = 19
...

$ spin -t2 bridge4.pml | expand
      (a,b)-->, t = 2
      <--a, t = 3
      (a,c)-->, t = 8
      <--b, t = 10
      (b,d)-->, t = 20
      tiempo total = 20
...

$ spin -t15 bridge4.pml | expand
      (a,d)-->, t = 10
      <--d, t = 20
      (b,d)-->, t = 30
      <--d, t = 40
      (c,d)-->, t = 50
      tiempo total = 50
...
```

La solución con el mejor tiempo es la siguiente:

```
$ spin -t5 bridge4.pml | expand
      (a,b)-->, t = 2
      <--a, t = 3
      (c,d)-->, t = 13
      <--b, t = 15
      (a,b)-->, t = 17
      tiempo total = 17
```

...

En esta solución los 2 más rápidos son los quien devuelven la linterna a la orilla izquierda.

Buscaremos la solución para *n* personas usando las siguientes consideraciones:

- Entrada: un *array* `a` que contiene los tiempos de *n* personas numeradas 0, 1, …, *n*-1.
- Salida: el tiempo total del cruce.
- Estrategia: usar las personas 0 y 1 como *shuttles* con linterna y enviar los otros en parejas:

```
for i ← 2 to n/2 do
    t ← a[1]          // 0 & 1 cruzan
    t ← t + a[0]      // 0 regresa
    t ← t + a[2i-1]   // 2i-1 & 2i-2 cruzan
    t ← t + a[1]      // 1 regresa
t ← t + a[1]          // 0 & 1 cruzan
return t
```

3a) Prepare el modelo correspondiente a esta propuesta en el archivo `bridgeNa.pml` para resolver el problema para 4 personas visto anteriormente obteniendo el mejor tiempo de 17 minutos.

3b) ¿Cuáles son las precondiciones? Incorpóralas en el código obteniendo la versión del modelo en el archivo `bridgeNb.pml`. Verifique que su modelo procesa todas las precondiciones.

3c) ¿Siempre se obtiene la respuesta óptima? ¿Por ejemplo, para el caso de 1, 20, 21, 22? Prepare este caso en el código `bridgeNc.pml` para obtener el tiempo. ¿Pero cuál es el mejor tiempo?

4. (*PCDP2E by M. Ben-Ari, Exercise 5 (Apt and Olderog)*) Assume that for the function *f*, there is some integer value *i* for which *f*(*i*) = 0. This concurrent algorithm searches for *i*. The algorithm is correct if for all scenarios, ***both*** processes terminate after one of them has found the zero. Show that this algorithm is correct of find a scenario that is a counterexample.

Algorithm 2.11: Zero A	
boolean found	
p	**q**
integer i ← 0	integer j ← 1
p1: found ← false	q1: found ← false
p2: while not found	q2: while not found
p3: i ← i + 1	q3: j ← j − 1
p4: found ← f(i) = 0	q4: found ← f(j) = 0

Se propone el código presentado en el programa `zeroA.pml`:

```
$ cat -n zeroA.pml | expand
     1  #define MAX  100
     2  #define HALF MAX/2
     3
     4  #define f(x) (44 - x)
     5
```

```
 6  bool found
 7
 8  active proctype P() {
 9    byte i=HALF
10
11    found=false
12    do
13    :: found ->
14                break
15    :: else ->
16                i++
17                if
18                :: i==MAX+1 ->
19                                i=HALF+1
20                :: else
21                fi
22                found = (f(i) == 0)
23    od
24  }
25
26  active proctype Q() {
27    byte j = HALF+1
28
29    found = false
30    do
31    :: found ->
32                break
33    :: else ->
34                j--
35                if
36                :: j==0 ->
37                            j=HALF
38                :: else
39                fi
40                found = (f(j) == 0)
41    od
42  }
```

4a) A veces la ejecución de este programa termina, a veces no, y hay que cortar su ejecución con Ctrl-C:

```
$ spin zeroA.pml
2 processes created
$ spin zeroA.pml
2 processes created
$ spin zeroA.pml
2 processes created
$ spin zeroA.pml
2 processes created
$ spin zeroA.pml
^C
$
```

¿Cómo se puede verificar el modelo? ¿Cómo se puede encontrar el escenario cuando el proceso `P` encuentra el valor buscado pero, en seguida, el proceso `Q` pone la variable `found` en `false` en la línea `29` (sin saber previamente que este escenario existe).