Clase 8: Preparación para el Examen 1
(Con respuestas)

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?
<u>Respuesta</u>: Porque mientras los 2 procesos no completan su primera iteración, un proceso lo logra e incrementa la variable `count` por 2da vez.

1b) ¿Qué significa y por qué sucede `timeout`?
<u>Respuesta</u>: `timeout` significa que en ningún proceso activo no hay sentencias ejecutables. Aquí sucede porque los 3 procesos activos `P` están en la línea 27 de `wait(barrier)` lo que es por definición `barrier > 0; ...` - la condición que es falsa y por eso no ejecutable mientras la variable `barrier` tiene el valor `0`.

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.

<u>Respuesta</u>: Identificaremos cada proceso con su parámetro y obtendremos el siguiente código:

```
$ cat -n lbs_3.6.3a.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     3.6.3 Barrier non-solution (sin deadlock)
     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 byte passed=0
    17
    18 proctype P(int i) {
    19     do
    20     ::  wait(mutex)
    21             count++
    22         signal(mutex)
    23         if
    24         :: count == N ->
    25                          signal(barrier)
    26         :: else
    27         fi
    28         wait(barrier)
    29         passed=passed*10 + i
    30         assert(passed!=123 && passed!=213  && passed!=231
    31                && passed!=312 && passed!=321)
    32     od
    33 }
    34
    35 init {
    36     byte i
    37
    38     atomic {
    39         for (i: 1 .. N) {
    40             run P(i)
```

```
    41        }
    42    }
    43 }
```

```
$ spin -run lbs_3.6.3a.pml | expand
pan:1: invalid end state (at depth 32)
pan: wrote lbs_3.6.3a.pml.trail
```

Ignoremos los estados finales inválidos con la opción `-E` del `pan`:

```
$ spin -run -E lbs_3.6.3a.pml | expand
pan:1:  assertion  violated  (((((passed!=123)&&(passed!=213))&&(passed!=231))&&(passed!
=312))&&(passed!=321)) (at depth 44)
pan: wrote lbs_3.6.3a.pml.trail
...
Full statespace search for:
        never claim             - (none specified)
        assertion violations    +
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      - (disabled by -E flag)

State-vector 48 byte, depth reached 49, errors: 1
...
```

Pero la traza corresponde al escenario cuando pasaron 3 procesos pero el contador ya está con 5 procesos, entonces el escenario obtenido no es el más corto:

```
$ spin -t -p -g lbs_3.6.3a.pml | expand
...
 44:    proc  1 (P:1) lbs_3.6.3a.pml:29 (state 14)      [passed = ((passed*10)+i)]
                passed = 231
spin: lbs_3.6.3a.pml:30, Error: assertion violated
spin:  text  of  failed  assertion:  assert(((((passed!=123)&&(passed!=213))&&(passed!
=231))&&(passed!=312))&&(passed!=321)))
  45:      proc    1  (P:1)  lbs_3.6.3a.pml:31  (state  15)          [assert(((((passed!
=123)&&(passed!=213))&&(passed!=231))&&(passed!=312))&&(passed!=321)))]
spin: trail ends after 45 steps
#processes: 4
                count = 5
                mutex = 1
                barrier = 0
                passed = 231
 45:    proc  3 (P:1) lbs_3.6.3a.pml:28 (state 13)
 45:    proc  2 (P:1) lbs_3.6.3a.pml:28 (state 13)
 45:    proc  1 (P:1) lbs_3.6.3a.pml:19 (state 16)
 45:    proc  0 (:init::1) lbs_3.6.3a.pml:43 (state 11) <valid end state>
4 processes created
```

Usaremos Breadth-First Search:

```
$ spin -run -E -DBFS lbs_3.6.3a.pml | expand
Depth=      10 States=       11 Transitions=       11 Memory=   128.195
Depth=      20 States=       62 Transitions=       83 Memory=   128.195
Depth=      30 States=      502 Transitions=    1e+03 Memory=   128.195
pan:1:  assertion  violated  (((((passed!=123)&&(passed!=213))&&(passed!=231))&&(passed!
=312))&&(passed!=321)) (at depth 34)
pan: wrote lbs_3.6.3a.pml.trail
...
Warning: Search not completed
        + Breadth-First Search
        + Partial Order Reduction

Full statespace search for:
```

```
        never claim             - (none specified)
        assertion violations    +
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      - (disabled by -E flag)

State-vector 48 byte, depth reached 34, errors: 1
...

$ spin -t -p -g lbs_3.6.3a.pml | expand
...
 14:    proc  3 (P:1) lbs_3.6.3a.pml:20 (state 1)       [((mutex>0))]
 14:    proc  3 (P:1) lbs_3.6.3a.pml:20 (state 2)       [mutex = (mutex-1)]
                mutex = 0
 15:    proc  3 (P:1) lbs_3.6.3a.pml:21 (state 4)       [count = (count+1)]
                count = 1
 16:    proc  3 (P:1) lbs_3.6.3a.pml:22 (state 5)       [mutex = (mutex+1)]
                mutex = 1
 17:    proc  2 (P:1) lbs_3.6.3a.pml:20 (state 1)       [((mutex>0))]
 17:    proc  2 (P:1) lbs_3.6.3a.pml:20 (state 2)       [mutex = (mutex-1)]
                mutex = 0
 18:    proc  2 (P:1) lbs_3.6.3a.pml:21 (state 4)       [count = (count+1)]
                count = 2
 19:    proc  2 (P:1) lbs_3.6.3a.pml:22 (state 5)       [mutex = (mutex+1)]
                mutex = 1
 20:    proc  1 (P:1) lbs_3.6.3a.pml:20 (state 1)       [((mutex>0))]
 20:    proc  1 (P:1) lbs_3.6.3a.pml:20 (state 2)       [mutex = (mutex-1)]
                mutex = 0
 21:    proc  1 (P:1) lbs_3.6.3a.pml:21 (state 4)       [count = (count+1)]
                count = 3
 22:    proc  3 (P:1) lbs_3.6.3a.pml:24 (state 6)       [((count==3))]
 23:    proc  3 (P:1) lbs_3.6.3a.pml:25 (state 7)       [barrier = (barrier+1)]
                barrier = 1
 24:    proc  3 (P:1) lbs_3.6.3a.pml:28 (state 11)      [((barrier>0))]
 24:    proc  3 (P:1) lbs_3.6.3a.pml:28 (state 12)      [barrier = (barrier-1)]
                barrier = 0
 25:    proc  2 (P:1) lbs_3.6.3a.pml:24 (state 6)       [((count==3))]
 26:    proc  2 (P:1) lbs_3.6.3a.pml:25 (state 7)       [barrier = (barrier+1)]
                barrier = 1
 27:    proc  2 (P:1) lbs_3.6.3a.pml:28 (state 11)      [((barrier>0))]
 27:    proc  2 (P:1) lbs_3.6.3a.pml:28 (state 12)      [barrier = (barrier-1)]
                barrier = 0
 28:    proc  2 (P:1) lbs_3.6.3a.pml:29 (state 14)      [passed = ((passed*10)+i)]
                passed = 2
 29:    proc  3 (P:1) lbs_3.6.3a.pml:29 (state 14)      [passed = ((passed*10)+i)]
                passed = 23
 30:    proc  1 (P:1) lbs_3.6.3a.pml:22 (state 5)       [mutex = (mutex+1)]
                mutex = 1
 31:    proc  1 (P:1) lbs_3.6.3a.pml:24 (state 6)       [((count==3))]
 32:    proc  1 (P:1) lbs_3.6.3a.pml:25 (state 7)       [barrier = (barrier+1)]
                barrier = 1
 33:    proc  1 (P:1) lbs_3.6.3a.pml:28 (state 11)      [((barrier>0))]
 33:    proc  1 (P:1) lbs_3.6.3a.pml:28 (state 12)      [barrier = (barrier-1)]
                barrier = 0
 34:    proc  1 (P:1) lbs_3.6.3a.pml:29 (state 14)      [passed = ((passed*10)+i)]
                passed = 231
spin: lbs_3.6.3a.pml:30, Error: assertion violated
spin: text of failed assertion: assert(((((passed!=123)&&(passed!=213))&&(passed!
=231))&&(passed!=312))&&(passed!=321)))
 35:    proc  3 (P:1) lbs_3.6.3a.pml:31 (state 15)      [assert(((((passed!
=123)&&(passed!=213))&&(passed!=231))&&(passed!=312))&&(passed!=321)))]
spin: trail ends after 35 steps
#processes: 4
                count = 3
                mutex = 1
                barrier = 0
                passed = 231
```

```
 35:    proc  3 (P:1) lbs_3.6.3a.pml:19 (state 16)
 35:    proc  2 (P:1) lbs_3.6.3a.pml:31 (state 15)
 35:    proc  1 (P:1) lbs_3.6.3a.pml:31 (state 15)
 35:    proc  0 (:init::1) lbs_3.6.3a.pml:43 (state 11) <valid end state>
4 processes created
```

Este escenario es casi de intercalación perfecta y es el más corto.

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.

Respuesta:

```
$ cat -n bridgeNa.pml | expand
     1  #define N 4
     2
     3  byte a[N]=0, t=0
     4
     5  proctype Bridge() {
     6      byte i
     7
     8      for (i: 2 .. N/2) {
     9          t=a[1]
    10          printf("(0,1) -->     , t = %d\n", t)
    11          t=t+a[0]
    12          printf("      <-- (0), t = %d\n", t)
    13          t=t+a[2*i-1]
    14          printf("(%d,%d) -->     , t = %d\n", a[2*i-2],a[2*i-1],t)
    15          t=t+a[1]
    16          printf("      <-- (1), t = %d\n", t)
    17      }
    18      t=t+a[1]
    19      printf("(0,1) -->     , t = %d\n", t)
    20  }
    21
    22  init {
    23      a[0]=1; a[1]=2; a[2]=5; a[3]=10
    24      run Bridge()
    25  }
```

```
$ spin bridgeNa.pml
          (0,1) -->     , t = 2
                <-- (0), t = 3
          (5,10) -->     , t = 13
                <-- (1), t = 15
          (0,1) -->     , t = 17
2 processes created
```

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.

Respuesta: El arreglo `a` debe estar ordenado, `n` debe ser par y mayor que 3.

```
$ cat -n bridgeNb.pml | expand
     1  #define N 4
     2
     3  byte a[N]=0, t=0
```

```
bool sorted=true

proctype Bridge() {
    byte i

    for (i: 2 .. N/2) {
        t=a[1]
        printf("(0,1) -->     , t = %d\n", t)
        t=t+a[0]
        printf("      <-- (0), t = %d\n", t)
        t=t+a[2*i-1]
        printf("(%d,%d) -->     , t = %d\n", a[2*i-2],a[2*i-1],t)
        t=t+a[1]
        printf("      <-- (1), t = %d\n", t)
    }
    t=t+a[1]
    printf("(0,1) -->     , t = %d\n", t)
}

init {
    byte i

    a[0]=1; a[1]=2; a[2]=5; a[3]=10
    for (i: 0 .. N-2) {
        sorted=sorted && (a[i]<=a[i+1])
    }
    assert(sorted)
    assert(N%2==0 && N>3)
    run Bridge()
}
```

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?

<u>Respuesta</u>: El algoritmo propuesto proporciona 83 minutos, mientras que el modelo del código `bridge4.pml` proporciona 65 minutos como el mejor tiempo de 65, 83, 84, 85, 86, 87, 88, 103, 104, 105, 106, 107, 108, 110 minutos.

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 or 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  }
```

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).

Respuesta:

Simplificamos el modelo para $N = 4$ e introducimos las variables ficticias para indicar la terminación de cada proceso con la fórmula de LTL correspondiente (`zeroA_1.pml`):

```
$ cat -n zeroA_1.pml | expand
     1  #define MAX  4        /* 1..2, 3..4 */
     2  #define HALF MAX/2
     3
     4  #define f(x) (3 - x) /* P will find it */
     5
     6  #define ok (Pexited && Qexited)     /* both are terminated */
     7
     8  ltl { <>ok }
     9
    10  bool found
    11  bool Pexited=false, Qexited=false  /* ghost variables */
    12
    13  active proctype P() {
    14    byte i=HALF
    15
    16    found=false
    17    do
    18    :: found ->
    19                break
    20    :: else ->
    21                i++
    22                if
    23                :: i==MAX+1 ->
    24                               i=HALF+1
    25                :: else
    26                fi
    27                found = (f(i) == 0)
    28    od
    29    Pexited=true
    30  }
    31
    32  active proctype Q() {
    33    byte j = HALF+1
    34
    35    found = false
    36    do
    37    :: found ->
    38                break
    39    :: else ->
    40                j--
    41                if
    42                :: j==0 ->
    43                           j=HALF
    44                :: else
    45                fi
    46                found = (f(j) == 0)
    47    od
    48    Qexited=true
    49  }

$ spin -run -a -f zeroA_1.pml | expand
pan:1: acceptance cycle (at depth 66)
pan: wrote zeroA_1.pml.trail
...
Full statespace search for:
        never claim             + (ltl_0)
        assertion violations    + (if within scope of claim)
        acceptance   cycles     + (fairness enabled)
```

```
        invalid end states      - (disabled by never claim)

State-vector 36 byte, depth reached 156, errors: 1
...

$ spin -t -p -g -l zeroA_1.pml | expand
ltl ltl_0: <> ((Pexited) && (Qexited))
starting claim 2
using statement merging
Never claim moves to line 4     [(!((Pexited&&Qexited)))]
  2:    proc  1 (Q:1) zeroA_1.pml:35 (state 1)  [found = 0]
  4:    proc  1 (Q:1) zeroA_1.pml:39 (state 4)  [else]
  6:    proc  1 (Q:1) zeroA_1.pml:40 (state 5)  [j = (j-1)]
                Q(1):j = 2
  8:    proc  1 (Q:1) zeroA_1.pml:44 (state 8)  [else]
 10:    proc  1 (Q:1) zeroA_1.pml:46 (state 11) [found = ((3-j)==0)]
 12:    proc  1 (Q:1) zeroA_1.pml:39 (state 4)  [else]
 14:    proc  1 (Q:1) zeroA_1.pml:40 (state 5)  [j = (j-1)]
                Q(1):j = 1
 16:    proc  1 (Q:1) zeroA_1.pml:44 (state 8)  [else]
 18:    proc  1 (Q:1) zeroA_1.pml:46 (state 11) [found = ((3-j)==0)]
 20:    proc  1 (Q:1) zeroA_1.pml:39 (state 4)  [else]
 22:    proc  1 (Q:1) zeroA_1.pml:40 (state 5)  [j = (j-1)]
                Q(1):j = 0
 24:    proc  0 (P:1) zeroA_1.pml:16 (state 1)  [found = 0]
 26:    proc  1 (Q:1) zeroA_1.pml:42 (state 6)  [((j==0))]
 26:    proc  1 (Q:1) zeroA_1.pml:43 (state 7)  [j = (4/2)]
                Q(1):j = 2
 28:    proc  1 (Q:1) zeroA_1.pml:46 (state 11) [found = ((3-j)==0)]
 30:    proc  1 (Q:1) zeroA_1.pml:39 (state 4)  [else]
 32:    proc  1 (Q:1) zeroA_1.pml:40 (state 5)  [j = (j-1)]
                Q(1):j = 1
 34:    proc  1 (Q:1) zeroA_1.pml:44 (state 8)  [else]
 36:    proc  1 (Q:1) zeroA_1.pml:46 (state 11) [found = ((3-j)==0)]
 38:    proc  1 (Q:1) zeroA_1.pml:39 (state 4)  [else]
 40:    proc  1 (Q:1) zeroA_1.pml:40 (state 5)  [j = (j-1)]
                Q(1):j = 0
 42:    proc  0 (P:1) zeroA_1.pml:20 (state 4)  [else]
 44:    proc  1 (Q:1) zeroA_1.pml:42 (state 6)  [((j==0))]
 44:    proc  1 (Q:1) zeroA_1.pml:43 (state 7)  [j = (4/2)]
                Q(1):j = 2
 46:    proc  0 (P:1) zeroA_1.pml:21 (state 5)  [i = (i+1)]
                P(0):i = 3
 48:    proc  0 (P:1) zeroA_1.pml:25 (state 8)  [else]
 50:    proc  1 (Q:1) zeroA_1.pml:46 (state 11) [found = ((3-j)==0)]
 52:    proc  1 (Q:1) zeroA_1.pml:39 (state 4)  [else]
 54:    proc  1 (Q:1) zeroA_1.pml:40 (state 5)  [j = (j-1)]
                Q(1):j = 1
 56:    proc  1 (Q:1) zeroA_1.pml:44 (state 8)  [else]
 58:    proc  1 (Q:1) zeroA_1.pml:46 (state 11) [found = ((3-j)==0)]
 60:    proc  1 (Q:1) zeroA_1.pml:39 (state 4)  [else]
 62:    proc  1 (Q:1) zeroA_1.pml:40 (state 5)  [j = (j-1)]
                Q(1):j = 0
 64:    proc  1 (Q:1) zeroA_1.pml:42 (state 6)  [((j==0))]
 64:    proc  1 (Q:1) zeroA_1.pml:43 (state 7)  [j = (4/2)]
                Q(1):j = 2
 66:    proc  0 (P:1) zeroA_1.pml:27 (state 11) [found = ((3-i)==0)]
                found = 1
  <<<<<START OF CYCLE>>>>>
 68:    proc  1 (Q:1) zeroA_1.pml:46 (state 11) [found = ((3-j)==0)]
                found = 0
 70:    proc  1 (Q:1) zeroA_1.pml:39 (state 4)  [else]
 72:    proc  1 (Q:1) zeroA_1.pml:40 (state 5)  [j = (j-1)]
                Q(1):j = 1
 74:    proc  1 (Q:1) zeroA_1.pml:44 (state 8)  [else]
 76:    proc  1 (Q:1) zeroA_1.pml:46 (state 11) [found = ((3-j)==0)]
```

```
 78:    proc  1 (Q:1) zeroA_1.pml:39 (state 4)  [else]
 80:    proc  1 (Q:1) zeroA_1.pml:40 (state 5)  [j = (j-1)]
                Q(1):j = 0
 82:    proc  1 (Q:1) zeroA_1.pml:42 (state 6)  [((j==0))]
 82:    proc  1 (Q:1) zeroA_1.pml:43 (state 7)  [j = (4/2)]
                Q(1):j = 2
 84:    proc  0 (P:1) zeroA_1.pml:20 (state 4)  [else]
 86:    proc  0 (P:1) zeroA_1.pml:21 (state 5)  [i = (i+1)]
                P(0):i = 4
 88:    proc  0 (P:1) zeroA_1.pml:25 (state 8)  [else]
 90:    proc  1 (Q:1) zeroA_1.pml:46 (state 11) [found = ((3-j)==0)]
 92:    proc  1 (Q:1) zeroA_1.pml:39 (state 4)  [else]
 94:    proc  1 (Q:1) zeroA_1.pml:40 (state 5)  [j = (j-1)]
                Q(1):j = 1
 96:    proc  1 (Q:1) zeroA_1.pml:44 (state 8)  [else]
 98:    proc  1 (Q:1) zeroA_1.pml:46 (state 11) [found = ((3-j)==0)]
100:    proc  1 (Q:1) zeroA_1.pml:39 (state 4)  [else]
102:    proc  1 (Q:1) zeroA_1.pml:40 (state 5)  [j = (j-1)]
                Q(1):j = 0
104:    proc  1 (Q:1) zeroA_1.pml:42 (state 6)  [((j==0))]
104:    proc  1 (Q:1) zeroA_1.pml:43 (state 7)  [j = (4/2)]
                Q(1):j = 2
106:    proc  0 (P:1) zeroA_1.pml:27 (state 11) [found = ((3-i)==0)]
108:    proc  1 (Q:1) zeroA_1.pml:46 (state 11) [found = ((3-j)==0)]
110:    proc  1 (Q:1) zeroA_1.pml:39 (state 4)  [else]
112:    proc  1 (Q:1) zeroA_1.pml:40 (state 5)  [j = (j-1)]
                Q(1):j = 1
114:    proc  1 (Q:1) zeroA_1.pml:44 (state 8)  [else]
116:    proc  1 (Q:1) zeroA_1.pml:46 (state 11) [found = ((3-j)==0)]
118:    proc  1 (Q:1) zeroA_1.pml:39 (state 4)  [else]
120:    proc  1 (Q:1) zeroA_1.pml:40 (state 5)  [j = (j-1)]
                Q(1):j = 0
122:    proc  1 (Q:1) zeroA_1.pml:42 (state 6)  [((j==0))]
122:    proc  1 (Q:1) zeroA_1.pml:43 (state 7)  [j = (4/2)]
                Q(1):j = 2
124:    proc  0 (P:1) zeroA_1.pml:20 (state 4)  [else]
126:    proc  0 (P:1) zeroA_1.pml:21 (state 5)  [i = (i+1)]
                P(0):i = 5
128:    proc  1 (Q:1) zeroA_1.pml:46 (state 11) [found = ((3-j)==0)]
130:    proc  0 (P:1) zeroA_1.pml:23 (state 6)  [((i==(4+1)))]
130:    proc  0 (P:1) zeroA_1.pml:24 (state 7)  [i = ((4/2)+1)]
                P(0):i = 3
132:    proc  1 (Q:1) zeroA_1.pml:39 (state 4)  [else]
134:    proc  1 (Q:1) zeroA_1.pml:40 (state 5)  [j = (j-1)]
                Q(1):j = 1
136:    proc  1 (Q:1) zeroA_1.pml:44 (state 8)  [else]
138:    proc  1 (Q:1) zeroA_1.pml:46 (state 11) [found = ((3-j)==0)]
140:    proc  1 (Q:1) zeroA_1.pml:39 (state 4)  [else]
142:    proc  1 (Q:1) zeroA_1.pml:40 (state 5)  [j = (j-1)]
                Q(1):j = 0
144:    proc  1 (Q:1) zeroA_1.pml:42 (state 6)  [((j==0))]
144:    proc  1 (Q:1) zeroA_1.pml:43 (state 7)  [j = (4/2)]
                Q(1):j = 2
146:    proc  0 (P:1) zeroA_1.pml:27 (state 11) [found = ((3-i)==0)]
                found = 1
spin: trail ends after 146 steps
#processes: 2
                found = 1
                Pexited = 0
                Qexited = 0
146:    proc  1 (Q:1) zeroA_1.pml:46 (state 11)
146:    proc  0 (P:1) zeroA_1.pml:17 (state 12)
146:    proc  - (ltl_0:1) _spin_nvr.tmp:3 (state 3)
2 processes created
```

O eliminando las variables ficticias (`zeroA_2.pml`):

```
$ cat -n zeroA_2.pml | expand
     1  #define MAX  4
     2  #define HALF MAX/2
     3
     4  #define f(x) (3 - x)
     5
     6  #define ok (P@Pexited && Q@Qexited)
     7
     8  ltl { <>ok }
     9
    10  bool found
    11
    12  active proctype P() {
    13    byte i=HALF
    14
    15    found=false
    16    do
    17    :: found ->
    18                break
    19    :: else ->
    20                i++
    21                if
    22                :: i==MAX+1 ->
    23                               i=HALF+1
    24                :: else
    25                fi
    26                found = (f(i) == 0)
    27    od
    28  Pexited:
    29  }
    30
    31  active proctype Q() {
    32    byte j = HALF+1
    33
    34    found = false
    35    do
    36    :: found ->
    37                break
    38    :: else ->
    39                j--
    40                if
    41                :: j==0 ->
    42                           j=HALF
    43                :: else
    44                fi
    45                found = (f(j) == 0)
    46    od
    47  Qexited:
    48  }

$ spin -run -a -f zeroA_2.pml | expand
pan:1: acceptance cycle (at depth 144)
pan: wrote zeroA_2.pml.trail
...
Full statespace search for:
        never claim             + (ltl_0)
        assertion violations    + (if within scope of claim)
        acceptance   cycles     + (fairness enabled)
        invalid end states      - (disabled by never claim)

State-vector 36 byte, depth reached 145, errors: 1
...
```

```
$ spin -t -p -g -l zeroA_2.pml | expand
ltl ltl_0: <> (((P@Pexited)) && ((Q@Qexited)))
...
 66:    proc  0 (P:1) zeroA_2.pml:26 (state 11) [found = ((3-i)==0)]
                found = 1
 68:    proc  1 (Q:1) zeroA_2.pml:45 (state 11) [found = ((3-j)==0)]
                found = 0
...
```

Para aislar el caso específico cuando el proceso `P` encuentra el valor colocando `true` en la variable `found` pero, en seguida, recién comienza ejecutarse el proceso `Q` inicializando con `false` la misma variable (`zeroA_3.pml`):

```
$ cat -n zeroA_3.pml | expand
#define MAX  4
#define HALF MAX/2

#define f(x) (3 - x)

ltl { !P@Pfound U Q@Qstart }  /* Process P will not pass to Pfound UNTIL
                                 process Q reaches Qstart */

bool found

active proctype P() {
  byte i=HALF

  found=false
  do
  :: found ->
              break
  :: else ->
              i++
              if
              :: i==MAX+1 ->
                             i=HALF+1
              :: else
              fi
              found = (f(i) == 0)
Pfound:
  od
}

active proctype Q() {
  byte j = HALF+1

  found = false
Qstart:
  do
  :: found ->
              break
  :: else ->
              j--
              if
              :: j==0 ->
                         j=HALF
              :: else
              fi
              found = (f(j) == 0)
  od
}
```

```
$ spin -run -a -f zeroA_3.pml | expand
pan:1: assertion violated  !(( !( !((P._p==Pfound))&& !((Q._p==Qstart)))) (at depth
10)
pan: wrote zeroA_3.pml.trail
...
Full statespace search for:
        never claim             + (ltl_0)
        assertion violations    + (if within scope of claim)
        acceptance   cycles     + (fairness enabled)
        invalid end states      - (disabled by never claim)

State-vector 36 byte, depth reached 16, errors: 1
...

$ spin -t -p -g -l zeroA_3.pml | expand
ltl ltl_0: (! ((P@Pfound))) U ((Q@Qstart))
starting claim 2
using statement merging
Never claim moves to line 4     [(!((Q._p==Qstart)))]
  2:    proc  0 (P:1) zeroA_3.pml:14 (state 1)  [found = 0]
  4:    proc  0 (P:1) zeroA_3.pml:18 (state 4)  [else]
  6:    proc  0 (P:1) zeroA_3.pml:19 (state 5)  [i = (i+1)]
                P(0):i = 3
  8:    proc  0 (P:1) zeroA_3.pml:23 (state 8)  [else]
 10:    proc  0 (P:1) zeroA_3.pml:25 (state 11) [found = ((3-i)==0)]
                found = 1
spin: _spin_nvr.tmp:5, Error: assertion violated
spin: text of failed assertion: assert(!((!(!((P._p==Pfound))&&!((Q._p==Qstart)))))
Never claim moves to line 5     [assert(!((!(!((P._p==Pfound))&&!((Q._p==Qstart)))))]
spin: trail ends after 11 steps
#processes: 2
                found = 1
                Pfound = 0
                Qstart = 0
 11:    proc  1 (Q:1) zeroA_3.pml:33 (state 1)
 11:    proc  0 (P:1) zeroA_3.pml:27 (state 12)
 11:    proc  - (ltl_0:1) _spin_nvr.tmp:3 (state 6)
2 processes created
```