PONTIFICIA UNIVESIDAD CATÓLICA DEL PERÚ
ESCUELA DE POSGRADO
MAESTRÍA EN INFORMÁTICA

INF646 MÉTODOS FORMALES
Examen 1
2017 – 2

Con respuestas

Prepare un directorio de trabajo con el nombre <*su-código-de-8-dígitos*>.
Este directorio es para desarrollar los programas de las preguntas del examen. Los nombres de los programas se indican en las preguntas.
Las respuestas a las preguntas y su comentarios usted puede preparar en el archivo <*su-código-de-8-dígitos*>**.txt**.
Al final del examen, comprime todo el directorio de trabajo al archivo <*su-código-de-8-dígitos*>**.zip** y colóquelo en la carpeta **Documentos del curso/Examen 1/Buzón/** en el Campus Virtual.

A esta hoja están acompañando los 4 archivos: **signaling_err.pml**, **Semaphore.h**, **bridge3.pml**, **bridge4.pml**. Cópialos a su directorio de trabajo.

Pregunta 1. (**10 puntos – 90 min.**) Con el siguiente programa se pretendía una sincronización entre las partes de 3 procesos:

```
$ cat -n signaling_err.pml | expand
     1  /***
     2   *  Deben ejecutarse:
     3   *  A1 después de B2
     4   *  A3 después de C3
     5   *  B1 después de C1
     6   *  B3 después de A2
     7   *  C2 después de B1
     8   *  C3 después de B3
     9   ***/
    10
    11  #include "Semaphore.h"
    12
    13  Semaphore a2=1, b1=1, b2=1, b3=1, c1=1, c3=1
    14
    15  proctype A() {
    16      wait(b2)
    17  A1: printf("A1\n")
    18  A2: printf("A2\n")
    19      signal(a2)
    20      wait(c3)
    21  A3: printf("A3\n")
    22  }
    23
    24  proctype B() {
    25      wait(c1)
    26  B1: printf("B1\n")
    27      signal(b1)
    28  B2: printf("B2\n")
    29      signal(b2)
    30      wait(a2)
    31  B3: printf("B3\n")
    32  }
```

```
33
34  proctype C() {
35  C1: printf("C1\n")
36      signal(c1)
37      wait(b1)
38  C2: printf("C2\n")
39      wait(b3)
40  C3: printf("C3\n")
41      signal(c3)
42  }
43
44  init {
45    atomic { run A(); run B(); run C() }
46  }
```

Se usó el archivo auxiliar **Semaphore.h**:

```
$ cat -n signaling_err.pml | expand
     1
     2  #define Semaphore   byte
     3
     4  #define wait(sem)       atomic { sem > 0; sem-- }
     5  #define signal(sem)     sem++
     6  #define signalN(sem,NN) for (_i: 1 .. NN) { sem++ } /* no atomic */
     7
     8  byte _i=0
     9
```

Pero la primera ejecución con la simulación aleatoria produjo la salida incorrecta:

```
$ spin signaling_err.pml
                C1      correcto, no depende de otras partes
            B1          correcto, está después de C1
        A1              incorrecto, debe estar después de B2
        A2
                C2
            B2
        A3
                C3
            B3
4 processes created
```

a) (**signaling.pml**) (**3 puntos – 27 min.**) Se necesita obtener el programa correcto **signaling.pml** que produzca las salidas como estas:

C1 B1 C2 B2 A1 A2 B3 C3 A3	C1 B1 B2 A1 C2 A2 B3 C3 A3	C1 B1 B2 C2 A1 A2 B3 C3 A3

En el archivo <*su-código-de-8-dígitos*>**.txt** indique cómo se obtiene el programa correcto. El programa **signaling.pml** debe quedarse en su directorio de trabajo que será comprimido y subido al buzón del Campus Virtual.

Con las órdenes

```
$ spin ... signaling.pml | expand > signaling.out1
$ spin ... signaling.pml | expand > signaling.out2
$ spin ... signaling.pml | expand > signaling.out3
```

prepare 3 archivos de resultados de 3 simulaciones diferentes (use la semilla del generador de números aleatorios diferente para cada caso).

Respuesta:

```
$ cat -n signaling.pml | expand
...
    13  Semaphore a2=0, b1=0, b2=0, b3=0, c1=0, c3=0
...
    32      signal(b3)
    33  }
...
```

```
$ spin -n1 signaling.pml | expand > signaling.out1
```

```
$ cat signaling.out1 | expand
                    C1
               B1
               B2
          A1
          A2
                    C2
               B3
                    C3
          A3
4 processes created
```

```
$ spin -n2 signaling.pml | expand > signaling.out2
```

```
$ cat signaling.out2 | expand
                    C1
               B1
               B2
                    C2
          A1
          A2
               B3
                    C3
          A3
4 processes created
```

```
$ spin -n4 signaling.pml | expand > signaling.out3
```

```
$ cat signaling.out3 | expand
                    C1
               B1
                    C2
               B2
          A1
          A2
               B3
                    C3
          A3
4 processes created
```

b) (**`signaling_verified.pml`**) (**5 puntos – 45 min.**) Prepare el modelo para verificarlo con la lógica temporal. Use el operador *until* descrito en la diapositiva 59 de la clase 7 y en la página 90 (103 del archivo pdf) del libro *Principles of the Spin Model Checker* de M. Ben-Ari, sección 5.9.3 *Precedence*. Verifique el modelo. Los resultados de verificación presente en los archivos **`signaling_verified.pan_resulti`**, donde ***`i`*** corresponde al número de la verificación.

Respuesta:

```
$ cat -n signaling_verified.pml | expand
     1  /***
     2   *  Deben ejecutarse:
     3   *  A1 después de B2
     4   *  A3 después de C3
     5   *  B1 después de C1
     6   *  B3 después de A2
     7   *  C2 después de B1
     8   *  C3 después de B3
     9   ***/
    10
    11  #include "Semaphore.h"
    12
    13  Semaphore a2=0, b1=0, b2=0, b3=0, c1=0, c3=0
    14  bool doneA1=false, doneA2=false, doneA3=false
    15  bool doneB1=false, doneB2=false, doneB3=false
    16  bool doneC1=false, doneC2=false, doneC3=false
    17
    18  ltl p1 { !doneA1 U doneB2 }
    19  ltl p2 { !doneA3 U doneC3 }
    20  ltl p3 { !doneB1 U doneC1 }
    21  ltl p4 { !doneB3 U doneA2 }
    22  ltl p5 { !doneC2 U doneB1 }
    23  ltl p6 { !doneC3 U doneB3 }
    24
    25  proctype A() {
    26      wait(b2)
    27  A1: doneA1=true
    28  A2: doneA2=true
    29      signal(a2)
    30      wait(c3)
    31  A3: doneA3=true
    32  }
    33
    34  proctype B() {
    35      wait(c1)
    36  B1: doneB1=true
    37      signal(b1)
    38  B2: doneB2=true
    39      signal(b2)
    40      wait(a2)
    41  B3: doneB3=true
    42      signal(b3)
    43  }
    44
    45  proctype C() {
    46  C1: doneC1=true
    47      signal(c1)
    48      wait(b1)
    49  C2: doneC2=true
    50      wait(b3)
    51  C3: doneC3=true
```

```
    52      signal(c3)
    53  }
    54
    55  init {
    56    atomic { run A(); run B(); run C() }
    57  }
```

```
$ spin -run signaling_verified.pml | expand > signaling_verified.pan_result1

$ cat signaling_verified.pan_result1 | expand
0: Claim p1 (4), from state 6

(Spin Version 6.4.6 -- 2 December 2016)
        + Partial Order Reduction

Full statespace search for:
        never claim             + (p1)
        assertion violations    + (if within scope of claim)
        acceptance   cycles     + (fairness disabled)
        invalid end states      - (disabled by never claim)

State-vector 60 byte, depth reached 20, errors: 0
       12 states, stored (24 visited)
        8 states, matched
       32 transitions (= visited+matched)
        4 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
    0.001       equivalent memory usage for states (stored*(State-vector + overhead))
    0.281       actual memory usage for states
  128.000       memory used for hash table (-w24)
    0.534       memory used for DFS stack (-m10000)
  128.730       total actual memory usage


unreached in proctype A
        signaling_verified.pml:27, state 4, "doneA1 = 1"
        signaling_verified.pml:28, state 5, "doneA2 = 1"
        signaling_verified.pml:29, state 6, "a2 = (a2+1)"
        signaling_verified.pml:30, state 9, "((c3>0))"
        signaling_verified.pml:31, state 10, "doneA3 = 1"
        signaling_verified.pml:32, state 11, "-end-"
        (6 of 11 states)
unreached in proctype B
        signaling_verified.pml:40, state 10, "((a2>0))"
        signaling_verified.pml:41, state 11, "doneB3 = 1"
        signaling_verified.pml:42, state 12, "b3 = (b3+1)"
        signaling_verified.pml:43, state 13, "-end-"
        (4 of 13 states)
unreached in proctype C
        signaling_verified.pml:51, state 10, "doneC3 = 1"
        signaling_verified.pml:52, state 11, "c3 = (c3+1)"
        signaling_verified.pml:53, state 12, "-end-"
        (3 of 12 states)
unreached in init
        (0 of 5 states)
unreached in claim p1
        _spin_nvr.tmp:9, state 10, "-end-"
        (1 of 10 states)
unreached in claim p2
        _spin_nvr.tmp:19, state 10, "-end-"
        (1 of 10 states)
```

```
unreached in claim p3
        _spin_nvr.tmp:29, state 10, "-end-"
        (1 of 10 states)
unreached in claim p4
        _spin_nvr.tmp:39, state 10, "-end-"
        (1 of 10 states)
unreached in claim p5
        _spin_nvr.tmp:49, state 10, "-end-"
        (1 of 10 states)
unreached in claim p6
        _spin_nvr.tmp:59, state 10, "-end-"
        (1 of 10 states)

pan: elapsed time 0 seconds
ltl p1: (! (doneA1)) U (doneB2)
ltl p2: (! (doneA3)) U (doneC3)
ltl p3: (! (doneB1)) U (doneC1)
ltl p4: (! (doneB3)) U (doneA2)
ltl p5: (! (doneC2)) U (doneB1)
ltl p6: (! (doneC3)) U (doneB3)
  the model contains 6 never claims: p6, p5, p4, p3, p2, p1
  only one claim is used in a verification run
  choose which one with ./pan -a -N name (defaults to -N p1)
  or use e.g.: spin -search -ltl p1 signaling_verified.pml

$ spin -run -ltl p2 signaling_verified.pml | expand > signaling_verified.pan_result2
$ cat signaling_verified.pan_result2 | expand
pan: ltl formula p2
...
Full statespace search for:
        never claim             + (p2)

State-vector 60 byte, depth reached 40, errors: 0
...

$ spin -run -ltl p3 signaling_verified.pml | expand > signaling_verified.pan_result3
$ cat signaling_verified.pan_result3 | expand
pan: ltl formula p3
...
Full statespace search for:
        never claim             + (p3)

State-vector 60 byte, depth reached 6, errors: 0
...

...

$ spin -run -ltl p6 signaling_verified.pml | expand > signaling_verified.pan_result6
$ cat signaling_verified.pan_result6 | expand
pan: ltl formula p6
...
Full statespace search for:
        never claim             + (p6)

State-vector 60 byte, depth reached 34, errors: 0
...
```

c) (**signaling_verified_err.pml**) (**2 puntos – 18 min.**) Modifique ligeramente el modelo anterior para que que este no cumpla con uno de requerimientos establecidos y guárdelo en el archivo **signaling_verified_err.pml**. Verifique el modelo para que Spin encuentre el error. El resultado de verificación presente en el archivo **signaling_verified_err.pan_result**.

Respuesta:

```
$ cat -n signaling_verified_err.pml | expand
     1  /***
     2   *  Deben ejecutarse:
     3   *  A1 después de B2
     4   *  A3 después de C3
     5   *  B1 después de C1
     6   *  B3 después de A2
     7   *  C2 después de B1
     8   *  C3 después de B3
     9   ***/
...
    49  C2: doneC2=true
    50  /*
    51      wait(b3)
    52  */
    53  C3: doneC3=true
    54      signal(c3)
    55  }
    56
    57  init {
    58    atomic { run A(); run B(); run C() }
    59  }


$ spin -run -ltl p6 signaling_verified_err.pml | expand > signaling_verified_err.pan_result

$ cat signaling_verified_err.pan_result | expand
pan: ltl formula p6
pan:1: assertion violated  !(( !( !(doneC3))&& !(doneB3))) (at depth 42)
pan: wrote signaling_verified_err.pml.trail

(Spin Version 6.4.6 -- 2 December 2016)
Warning: Search not completed
        + Partial Order Reduction

Full statespace search for:
        never claim             + (p6)
        assertion violations    + (if within scope of claim)
        acceptance   cycles     + (fairness disabled)
        invalid end states      - (disabled by never claim)

State-vector 60 byte, depth reached 44, errors: 1
...
```

Pregunta 2 **(5 puntos – 45 min.)** Consider the following two processes, *A* and *B*, to be run concurrently in a shared memory (all variables are shared between the two processes):

```
PROCESS A:                          PROCESS B:

1    for i := 1 to 5 do             1    x := x << 1
2        x := x + 1
3    od
```

Assume that load (read) and store (write) of the single shared register *x* are atomic, *x* is initialized to 0, and *x* must be loaded into a register before being incremented or being used in any operation. What are all the possible values for *x* after both processes have terminated?

a) (**gt_1_1.pml**) **(3 puntos – 27 min.)** Prepare el modelo correspondiente en el archivo **gt_1_1.pml**. Con Spin verifique el modelo para obtener todos los valores posibles de la variable. En el documento, publicado en el Campus Virtual y el que contiene la solución de los problemas presentados en preparación para este examen, se describen las órdenes de *bash* (*shell*, el intérprete de órdenes) que permiten rápidamente preparar todos los archivos necesarios. Usted debe presentar, además del archivo **gt_1_1.pml**, los archivos **gt_1_1.pan_result**, **gt_1_1.all_errors**, **gt_1_1.final_values**. En el archivo <*su-código-de-8-dígitos*>**.txt** indique los valores finales posibles.

Respuesta: el valor final de la variable *x* está en el rango 0 .. 10.

```
$ cat -n gt_1_1.pml | expand
     1  byte x = 0
     2
     3  active proctype A() {
     4      byte i, t
     5
     6      for (i : 1 .. 5) {
     7          t = x
     8          x = t + 1
     9      }
    10  }
    11
    12  active proctype B() {
    13      byte t
    14
    15      t = x
    16      x = t<<1
    17  }
    18
    19  init {
    20      _nr_pr == 1
    21      assert(x == 0)
    22  }

$ spin -run -e gt_1_1.pml | expand > gt_1_1.pan_result

$ cat gt_1_1.pan_result | expand
pan:1: invalid end state (at depth 23)
pan: wrote gt_1_1.pml1.trail
pan: wrote gt_1_1.pml2.trail
pan: wrote gt_1_1.pml3.trail
pan: wrote gt_1_1.pml4.trail
pan: wrote gt_1_1.pml5.trail
pan: wrote gt_1_1.pml6.trail
```

```
pan: wrote gt_1_1.pml7.trail
pan: wrote gt_1_1.pml8.trail
pan: wrote gt_1_1.pml9.trail
pan: wrote gt_1_1.pml10.trail
pan: wrote gt_1_1.pml11.trail
pan: wrote gt_1_1.pml12.trail
pan: wrote gt_1_1.pml13.trail
pan: wrote gt_1_1.pml14.trail
pan: wrote gt_1_1.pml15.trail
pan: wrote gt_1_1.pml16.trail
pan: wrote gt_1_1.pml17.trail
pan: wrote gt_1_1.pml18.trail
pan: wrote gt_1_1.pml19.trail
pan: wrote gt_1_1.pml20.trail
pan: wrote gt_1_1.pml21.trail
pan: wrote gt_1_1.pml22.trail
pan: wrote gt_1_1.pml23.trail

(Spin Version 6.4.6 -- 2 December 2016)
        + Partial Order Reduction

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

State-vector 28 byte, depth reached 24, errors: 23
...

$ > gt_1_1.all_errors; for i in {1..23}; do ~/spin646/Spin/Src6.4.6/spin -t$i gt_1_1.pml
| expand >> gt_1_1.all_errors; done

$ cat gt_1_1.all_errors | expand | grep x > gt_1_1.final_values

$ cat gt_1_1.final_values | expand
                x = 5
                x = 4
                x = 3
                x = 2
                x = 1
                x = 0
                x = 6
                x = 5
                x = 4
                x = 3
                x = 2
                x = 7
                x = 5
                x = 6
                x = 4
                x = 8
                x = 5
                x = 7
                x = 6
                x = 9
                x = 5
                x = 8
                x = 10
```

b) (**2 puntos – 18 min.**) En el archivo <*su-código-de-8-dígitos*>**.txt** explique cómo se obtienen el valor mínimo, el valor máximo y el valor 6.

Respuesta:
***x* será 0 con la secuencia 15, 6-9 cinco veces, 10, 16**
***x* será 10 si el proceso B termina después del proceso A**
***x* será 6 si, por ejemplo, el proceso A hace 3 iteraciones, el proceso B recoge x igual a 3, se interrumpe, el proceso A completa su trabajo llevando x a 5, el proceso B duplica el valor 3 recogido anteriormente y lo duplica obteniendo 6.**

Pregunta 3 (**bridge4.pml**) **(5 puntos – 45 min.)** Prepare el modelo para el siguiente **Bridge Crossing Problem** (se recomenda modificar el código de **bridge3.pml**):

Four people begin on the same side of a bridge. You must help them across to the other side. It is night. There is one flashlight. A maximum of two people can cross at a time. Any party who crosses, either one or two people, must have the flashlight to see. The flashlight must be walked back and forth, it cannot be thrown, etc. Each person walks at a different speed. A pair must walk together at the rate of the slower person's pace, based on this information: Person 1 takes t1 = 1 minutes to cross, and the other persons take t2 = 2 minutes, t3 = 5 minutes, and t4 = 10 minutes to cross, respectively.

```
$ cat -n bridge4.pml | expand
```

```
     1  #define max(a,b) ((a>b) -> a : b)
     2  #define N 4
     3
     4  byte a[N+1] = 0      /* crossing times of N persons, a[0] is dummy */
     5  bool c[N+1] = false  /* nobody crossed, c[0] corresponds to the flashlight */
     6  byte t = 0           /* total time */
     7
     8  active proctype Bridge() {
     9      skip
    10  }
```

Complete el modelo.

Encuentre todos los valores finales posibles para la variable `t`.

Presente los archivos

bridge4.pan_result (un único archivo con todos los errores),
bridge4.all_errors (un único archivo con todos los *trails*),
bridge4.final_values (el archivo de las líneas con el tiempo total filtrados del archivo anterior).

Con la orden

```
$ cat -n bridge4.final_values | expand | sort -n -k 5 | head -n 1
```

enumere las líneas del archivo **bridge4.final_values**, ordene numéricamente por el campo #5 (el valor del tiempo) y despliegue solamente la primera línea.

En el primer campo de la línea desplegada se indica el número de *trail*-archivo que corresponde al

tiempo mínimo del cruce del puente.

Ejecute Spin con este *trail* para ver de qué manera el cruce del puente será óptimo guardando el resultado en el archivo **`bridge4.min_time`**.

Respuesta:

```
$ cat -n bridge4.pml | expand
     1  #define max(a,b) ((a>b) -> a : b)
     2  #define N 4
     3
     4  byte a[N+1] = 0      /* crossing times of N persons, a[0] is dummy */
     5  bool c[N+1] = false  /* nobody crossed, c[0] corresponds to the flashlight */
     6  byte t = 0           /* total time */
     7
     8  active proctype Bridge() {
     9      a[1]=1; a[2]=2; a[3]=5; a[4]=10
    10
    11      do
    12      ::  c[1]&&c[2]&&c[3]&&c[4] -> break    /* todos cruzaron */
    13      ::  else ->
    14              if
    15              ::  !c[0]&&!c[1]&&!c[2] ->
    16                                  c[1]=true; c[2]=true    /* cruzan 1+2 */
    17                                  c[0]=true
    18                                  t=t+max(a[1],a[2])
    19                                  printf("1,2 -->\n")
    20              ::  !c[0]&&!c[1]&&!c[3] ->
    21                                  c[1]=true; c[3]=true    /* cruzan 1+3 */
    22                                  c[0]=true
    23                                  t=t+max(a[1],a[3])
    24                                  printf("1,3 -->\n")
    25              ::  !c[0]&&!c[1]&&!c[4] ->
    26                                  c[1]=true; c[4]=true    /* cruzan 1+4 */
    27                                  c[0]=true
    28                                  t=t+max(a[1],a[4])
    29                                  printf("1,4 -->\n")
    30              ::  !c[0]&&!c[2]&&!c[3] ->
    31                                  c[2]=true; c[3]=true    /* cruzan 2+3 */
    32                                  c[0]=true
    33                                  t=t+max(a[2],a[3])
    34                                  printf("2,3 -->\n")
    35              ::  !c[0]&&!c[2]&&!c[4] ->
    36                                  c[2]=true; c[4]=true    /* cruzan 2+4 */
    37                                  c[0]=true
    38                                  t=t+max(a[2],a[4])
    39                                  printf("2,4 -->\n")
    40              ::  !c[0]&&!c[3]&&!c[4] ->
    41                                  c[3]=true; c[4]=true    /* cruzan 3+4 */
    42                                  c[0]=true
    43                                  t=t+max(a[3],a[4])
    44                                  printf("3,4 -->\n")
    45              ::   c[0]&&c[1] ->
    46                                  c[1]=false              /* regresa 1 */
    47                                  c[0]=false
    48                                  t=t+a[1]
    49                                  printf("    <-- 1\n")
    50              ::   c[0]&&c[2] ->
    51                                  c[2]=false              /* regresa 2 */
    52                                  c[0]=false
    53                                  t=t+a[2]
```

```
    54                                  printf("    <-- 2\n")
    55              ::   c[0]&&c[3] ->
    56                                  c[3]=false              /* regresa 3 */
    57                                  c[0]=false
    58                                  t=t+a[3]
    59                                  printf("    <-- 3\n")
    60              ::   c[0]&&c[4] ->
    61                                  c[4]=false              /* regresa 4 */
    62                                  c[0]=false
    63                                  t=t+a[4]
    64                                  printf("    <-- 4\n")
    65              fi
    66      od
    67
    68      printf("total time = %d\n", t)
    69      assert(t==0)
    70  }
```

```
$ spin -run -e bridge4.pml | expand > bridge4.pan_result

$ cat bridge4.pan_result | expand
pan:1: assertion violated (t==0) (at depth 39)
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

(Spin Version 6.4.6 -- 2 December 2016)
        + Partial Order Reduction

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

State-vector 28 byte, depth reached 41, errors: 15
      991 states, stored
       61 states, matched
     1052 transitions (= stored+matched)
        0 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
    0.053       equivalent memory usage for states (stored*(State-vector + overhead))
    0.286       actual memory usage for states
  128.000       memory used for hash table (-w24)
    0.534       memory used for DFS stack (-m10000)
  128.730       total actual memory usage


unreached in proctype Bridge
```

```
        (0 of 71 states)

pan: elapsed time 0 seconds

$ > bridge4.all_errors; for i in {1..15}; do ~/spin646/Spin/Src6.4.6/spin -t$i
bridge4.pml | expand >>bridge4.all_errors; done

$ cat bridge4.all_errors | expand | grep total > bridge4.final_values

$ cat -n bridge4.final_values | expand | sort -n -k 5 | head -n 1
     5        total time = 17

$ spin -t5 bridge4.pml | expand > bridge4.min_time

$ cat bridge4.min_time | expand
      1,2 -->
          <-- 1
      3,4 -->
          <-- 2
      1,2 -->
      total time = 17
spin: bridge4.pml:69, Error: assertion violated
spin: text of failed assertion: assert((t==0))
spin: trail ends after 40 steps
#processes: 1
                a[0] = 0
                a[1] = 1
                a[2] = 2
                a[3] = 5
                a[4] = 10
                c[0] = 1
                c[1] = 1
                c[2] = 1
                c[3] = 1
                c[4] = 1
                t = 17
 40:    proc  0 (Bridge:1) bridge4.pml:70 (state 71) <valid end state>
1 process created
```

Profesor: V. Khlebnikov
Pando, 13 de octubre de 2017