PONTIFICIA UNIVESIDAD CATÓLICA DEL PERÚ
ESCUELA DE POSGRADO
MAESTRÍA EN INFORMÁTICA

INF646 MÉTODOS FORMALES
Examen 1
2016 – 2

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: **lbs_3.6.4a.pml**, **lbs_3.7.1a.pml**, **lbs_3.7.3a.pml**, y **zeroA.pml**. Cópialos a su directorio de trabajo.

Pregunta 1. (10 puntos – 90 min.) (*The Little Book of Semaphore (2.2.1) by A. Downey, pp.29-37*)
La solución de una barrera según el código de la sección 3.6.4:

```
$ cat -n lbs_3.6.4a.pml | expand
     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      3.6.4 Barrier 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 /* barrier is locked */
    16
    17  proctype P(byte i) {
    18      do
    19      ::  wait(mutex)
    20              count++
    21          signal(mutex)
    22
    23          if
    24          :: count == N ->
    25                          signal(barrier)
    26          :: else
    27          fi
    28
    29          wait(barrier)    /* turnstile pattern */
    30          signal(barrier)
    31
    32          break    /* one only iteration */
    33      od
```

```
    34  }
    35
    36  init {
    37      byte i
    38
    39      atomic {
    40          for (i: 1 .. N) {
    41              run P(i)
    42          }
    43      }
    44      _nr_pr == 1 ->
    45                      assert(barrier != 0) /* barrier (turnstile) is open! */
    46  }
```

```
$ spin -run lbs_3.6.4a.pml | expand > lbs_3.6.4a.pan_out
$ cat lbs_3.6.4a.pan_out

(Spin Version 6.4.5 -- 1 January 2016)
        + Partial Order Reduction

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

State-vector 48 byte, depth reached 40, errors: 0
      498 states, stored
      465 states, matched
      963 transitions (= stored+matched)
       12 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
      0.036          equivalent memory usage for states (stored*(State-vector +
overhead))
    0.289       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 P
        (0 of 19 states)
unreached in init
        (0 of 13 states)

pan: elapsed time 0 seconds
```

La verificación con la búsqueda de violaciones de asertos y estados finales inválidos no encuentra errores.

a) (`lbs_3.6.4b.pml`) (3 puntos – 27 min.) ¿Cuál es el valor final de la variable **`barrier`** para *N* procesos? Prepare el modelo correspondiente en el archivo **`lbs_3.6.4b.pml`**, presente los resultados del analizador en el archivo **`lbs_3.6.4b.pan.out`**, y, para cada caso encontrado, los resultados de simulación en el archivo **`lbs_3.6.4b.pml`***`i`***`.trail.out`** donde *i* corresponde al número del caso.

b) (`lbs_3.7.1a.pml`) (3 puntos – 27 min.) En el archivo **`lbs_3.7.1a.pml`**, está el modelo correspondiente al código de la sección 3.7.1:

```
$ cat -n lbs_3.7.1a.pml | expand
     1  /*  The Little Book of Semaphores (2.2.1)
     2      by A. Downey
     3
     4      Chapter 3. Basic synchronization patterns
     5
     6      3.7 Reusable barrier
     7      3.7.1 Reusable barrier non-solution #1
     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, turnstile=0 /* turnstile is locked */
    16
    17  proctype P(byte i) {
    18      do
    19      ::  wait(mutex)
    20              count++
    21          signal(mutex)
    22
    23          if
    24          :: count == N ->
    25                          signal(turnstile)
    26          :: else
    27          fi
    28
    29          wait(turnstile)
    30          signal(turnstile)
    31
    32          /* critical point */
    33
    34          wait(mutex)
    35              count--
    36          signal(mutex)
    37
    38          if
    39          :: count == 0 ->
    40                          wait(turnstile)
    41          :: else
    42          fi
    43
    44          break    /* one only iteration */
    45      od
    46  }
    47
    48  init {
    49      byte i
    50
    51      atomic {
    52          for (i: 1 .. N) {
    53              run P(i)
    54          }
    55      }
    56      _nr_pr == 1 ->
    57                      assert(turnstile == 0) /* must be locked */
    58  }
```

Se dice que no es una solución correcta porque varios procesos pueden encontrar la condición en la línea 24 como cierta completando la línea 25. Lo mismo sucede con la condición en la línea 39 que puede poner en *deadlock* varios procesos.

El analizador proporciona los siguientes resultados:

```
$ spin -run lbs_3.7.1a.pml | expand
pan:1: invalid end state (at depth 46)
pan: wrote lbs_3.7.1a.pml.trail

(Spin Version 6.4.5 -- 1 January 2016)
Warning: Search not completed
        + Partial Order Reduction

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

State-vector 48 byte, depth reached 51, errors: 1
       52 states, stored
        3 states, matched
       55 transitions (= stored+matched)
       12 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
    0.004       equivalent memory usage for states (stored*(State-vector + overhead))
    0.288       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



pan: elapsed time 0.01 seconds

$ spin -t lbs_3.7.1a.pml | expand
spin: trail ends after 47 steps
#processes: 2
                count = 0
                mutex = 1
                turnstile = 0
 47:    proc  1 (P:1) lbs_3.7.1a.pml:40 (state 23)
 47:    proc  0 (:init::1) lbs_3.7.1a.pml:56 (state 11)
4 processes created
```

según cuales se bloquea un solo proceso. No se necesita presentar ningún archivo, solamente se pide describir el escenario que lleva a la situación presentada.

c) (`lbs_3.7.3b.pml`) (**4 puntos – 36 min.**) En el archivo **`lbs_3.7.3a.pml`**, está el modelo correspondiente al código de la sección 3.7.3:

```
$ cat -n lbs_3.7.3a.pml | expand
     1  /*  The Little Book of Semaphores (2.2.1)
     2      by A. Downey
     3
     4      Chapter 3. Basic synchronization patterns
     5
     6      3.7 Reusable barrier
     7      3.7.3 Reusable barrier non-solution #2
     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, turnstile=0 /* turnstile is locked */
    16
    17  proctype P(byte i) {
    18      do
    19      ::  wait(mutex)
    20              count++
    21              if
    22              :: count == N ->
    23                              signal(turnstile) /* the last opens turnstile */
    24              :: else
    25              fi
    26          signal(mutex)
    27
    28          wait(turnstile)
    29          signal(turnstile)
    30
    31          /* critical point */
    32
    33          wait(mutex)
    34              count--
    35              if
    36              :: count == 0 ->
    37                              wait(turnstile) /* the last closes turnstile */
    38              :: else
    39              fi
    40          signal(mutex)
    41      od
    42  }
    43
    44  init {
    45      byte i
    46
    47      atomic {
    48          for (i: 1 .. N) {
    49              run P(i)
    50          }
    51      }
    52  }
```

Y el analizador proporciona los siguientes resultados:

```
$ spin -run lbs_3.7.3a.pml | expand
pan:1: invalid end state (at depth 5646)
pan: wrote lbs_3.7.3a.pml.trail

(Spin Version 6.4.5 -- 1 January 2016)
Warning: Search not completed
        + Partial Order Reduction

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

State-vector 48 byte, depth reached 5647, errors: 1
     5636 states, stored
        0 states, matched
     5636 transitions (= stored+matched)
       12 atomic steps
hash conflicts:         0 (resolved)

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



pan: elapsed time 0.01 seconds

$ spin -t -g lbs_3.7.3a.pml | expand
spin: lbs_3.7.3a.pml:23, Error: value (256->0 (8)) truncated in assignment
spin: lbs_3.7.3a.pml:23, Error: value (256->0 (8)) truncated in assignment
spin: trail ends after 5647 steps
#processes: 4
                count = 3
                mutex = 1
                turnstile = 0
5647:   proc  3 (P:1) lbs_3.7.3a.pml:28 (state 13)
5647:   proc  2 (P:1) lbs_3.7.3a.pml:28 (state 13)
5647:   proc  1 (P:1) lbs_3.7.3a.pml:28 (state 13)
5647:   proc  0 (:init::1) lbs_3.7.3a.pml:52 (state 11) <valid end state>
4 processes created
```

En la página 37 del libro se dice: “Tragically, the code is *still* not correct. Remember that this barrier will be inside a loop. So, after executing the last line each thread will go back to the rendezvous.”

En el archivo **`lbs_3.7.3b.pml`** prepare el modelo que exija una sincronización entre los procesos durante el paso de la barrera para que ningún proceso avance demasiado y que permita detectar el error antes posible.

Pregunta 2. (`zeroB.pml`,`zeroC.pml`) (**4 puntos – 36 min.**) (*PCDP2E by M. Ben-Ari, Chapter 2, Exercise 5* (*Apt and Olderog*).) Usted ya conoce el modelo para el **Algorithm 2.11: Zero A**

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

```
$ cat -n zeroA.pml | expand
     1  #define MAX  100    /* 0..49, 50..99 */
     2  #define HALF MAX/2
     3
     4  #define f(x) (54 - 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                found = (f(i) == 0)
    17                if
    18                :: i==MAX-1 ->
    19                                i=HALF
    20                :: else     ->
    21                                i++
    22                fi
    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                found = (f(j) == 0)
    35                if
    36                :: j==0 ->
    37                           j=HALF-1
    38                :: else ->
    39                           j--
    40                fi
    41    od
    42  }
```

Prepare en los archivos **`zeroB.pml`**, **`zeroC.pml`** y verifique los modelos correspondientes a los siguientes algoritmos:

Algorithm 2.12: Zero B	
boolean found ← false	
p	**q**
integer i ← 0	integer j ← 1
p1: while not found	q1: while not found
p2: i ← i + 1	q2: j ← j − 1
p3: found ← f(i) = 0	q3: found ← f(j) = 0

Algorithm 2.13: Zero C	
boolean found ← false	
p	**q**
integer i ← 0	integer j ← 1
p1: while not found	q1: while not found
p2: i ← i + 1	q2: j ← j − 1
p3: if f(i) = 0	q3: if f(j) = 0
p4: found ← true	q4: found ← true

Pregunta 3. (`alg2.16a.pml`, `alg2.16b.pml`, `alg2.16c.pml`) (6 puntos – 54 min.) (*PCDP2E by M. Ben-Ari, Chapter 2, Exercise 5* (*Apt and Olderog*).) Consider the following algorithm where each of ten processes executes the statements with *i* set to a different number in 1, …, 10:

Algorithm 2.16: Concurrent algorithm A	
	integer array [1..10] C ← ten *distinct* initial values integer array [1..10] D
	integer myNumber, count
p1:	myNumber ← C[i]
p2:	count ← number of elements of C less than myNumber
p3:	D[count + 1] ← myNumber

a) (`alg2.16a.pml`) (2 puntos – 18 min.) En el archivo **`alg2.16a.pml`** prepare el modelo para el algoritmo dado.

b) (`alg2.16b.pml`) (2 puntos – 18 min.) If *D* in line *p3* is replaced by *C*, sometimes may occur the indexing error. Why? What scenario with?

c) (`alg2.16c.pml`) (2 puntos – 18 min.) What would happen if the array *C* were initialized with values that are not all distinct? Correct the algorithm to take care of this case.

Profesor: V. Khlebnikov
Pando, 14 de octubre de 2016