PONTIFICIA UNIVESIDAD CATÓLICA DEL PERÚ
ESCUELA DE POSGRADO
MAESTRÍA EN INFORMÁTICA

INF646 MÉTODOS FORMALES
Examen 1
2017 – 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: **signaling_err.pml**, **Semaphore.h**, **bridge3.pml**, **bridge4.pml**. Cópialos a su directorio de trabajo.

<u>Pregunta 1</u>. (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	C1	C1
B1	B1	B1
C2	B2	B2
B2	A1	C2
A1	C2	A1
A2	A2	A2
B3	B3	B3
C3	C3	C3
A3	A3	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).

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.

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`**.

<u>Pregunta 2</u> (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.

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.

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`**.

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