PONTIFICIA UNIVESIDAD CATÓLICA DEL PERÚ
ESCUELA DE POSGRADO
INFORMÁTICA (INGENIERÍA DE SOFTWARE)

INF646 MÉTODOS FORMALES
Examen 1
2018 – 2

Prepare un directorio de trabajo con el nombre <*su-código-de-8-dígitos*>.
En este serán desarrolladas los programas de las preguntas del examen. Los nombres de los programas se indican en las preguntas.
Las respuestas a las preguntas prepare 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 5 archivos: **2018-2_ex1_1.pml**, **atomic_swap_1.pml**, **atomic_swap_2.pml**, **atomic_swap_3.pml** y **tictactoe_v0.pml**. Cópialos a su directorio de trabajo.

Pregunta 1. (**6 puntos – 54 min.**) Alguien (con no mucha experiencia en Promela) preparó el siguiente programa:

```
$ cat -n 2018-2_ex1_1.pml | expand
     1  #define lock(sem)      atomic { sem > 0; sem-- }
     2  #define unlock(sem)    sem++
     3
     4  int time
     5  int t[2], a[2]
     6  byte mutex
     7
     8  active proctype A() {
     9      byte cond1
    10
    11      time = time + 1
    12      time = time + 2
    13      t[0] = 3
    14      a[0] = 2
    15      do
    16      :: a[0] == 0 -> break
    17      :: else ->  a[0] = a[0] - 1
    18                  do
    19                  :: t[0] <= t[1] -> break
    20                  od
    21                  if
    22                  :: cond1 != 0 ->
    23                                  lock(mutex)
    24                                  time = time + 1
    25                                  time = time + 2
    26                                  t[0] = t[0] + 3
    27                                  unlock(mutex)
    28                  :: cond1 == 0 ->
    29                                  time = time + 1
    30                  fi
    31      od
    32      t[0] = 1000
    33  }
```

```
$ spin 2015-2_ex1_1.pml | expand
      timeout
#processes: 1
                time = 3
                t[0] = 3
                t[1] = 0
                a[0] = 1
                a[1] = 0
                mutex = 0
  8:     proc  0 (A:1) 2018-2_ex1_1.pml:18 (state 11)
1 process created
```

```
$ spin -run 2018-2_ex1_1.pml | expand
pan:1: invalid end state (at depth 5)
pan: wrote 2018-2_ex1_1.pml.trail

(Spin Version 6.4.8 -- 2 March 2018)
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 36 byte, depth reached 6, errors: 1
        7 states, stored
        0 states, matched
        7 transitions (= stored+matched)
        0 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
    0.000       equivalent memory usage for states (stored*(State-vector + overhead))
    0.290       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 seconds
```

a) (`2018-2_ex1_1.pml`) (3 puntos – 27 min.) ¿Por qué la ejecución aleatoria produce **`timeout`**? ¿Qué errores busca el analizador construido y que significa "**`invalid end state`**"?

b) (`2018-2_ex1_1.pml`) (1 punto – 9 min.) Usando el programa **`dot`** prepare el grafo de estados del proceso **`A`** en el formato pdf (en el archivo **`2018-2_ex1_1.pdf`**). Según este grafo (y el resultado de ejecución), ¿en qué estado sucedió el **`timeout`** y qué estado debería ser el siguiente?

c) (`2018-2_ex1_1a.pml`) (2 puntos – 18 min.) Modifique el programa **`2018-2_ex1_1.pml`** para obtener el programa **`2018-2_ex1_1a.pml`** que no tenga el error de **`timeout`**:

```
$ spin 2018-2_ex1_1a.pml
1 process created
```

Este programa seguirá siendo incorrecto. ¿Cómo se descubre esto y cuál es el error? (Presente su material de trabajo.) El autor del programa pensaba en usar el indeterminismo (*non-determinism*) de la construcción **`if`**, ayúdele lograrlo.

Pregunta 2. (**7 puntos – 63 min.**) (**atomic_swap_1.pml**) Considere el siguiente programa que intercambia los valores de un arreglo protegiéndolos durante el acceso:

```
$ cat -n atomic_swap_1.pml
     1  #define lock(sem)      atomic { sem > 0; sem-- }
     2  #define unlock(sem)    sem++
     3
     4  #define N 8
     5
     6  byte cell[N] = 0
     7  byte mutex[N]=1
     8
     9
    10  proctype AtomicSwap(byte f, s) {
    11      byte item[2]
    12
    13      lock(mutex[f])
    14      item[0] = cell[f]
    15      if
    16      :: item[0] != 0 -> lock(mutex[s])
    17                         item[1] = cell[s]
    18                         if
    19                         :: item[1] != 0 ->
    20                                              cell[s] = item[0]
    21                                              cell[f] = item[1]
    22                                              unlock(mutex[s])
    23                                              unlock(mutex[f])
    24                         :: else -> skip
    25                         fi
    26      :: else -> skip
    27      fi
    28  }
    29
    30  init {
    31      byte i
    32
    33      cell[0]=8; cell[2]=2; cell[4]=4; cell[6]=6
    34      cell[1]=1; cell[3]=3; cell[5]=5; cell[7]=7
    35
    36      printf("cell:")
    37      for (i in cell) {
    38          printf("%d",cell[i])
    39      }
    40      printf("\n")
    41
    42      i = 0
    43      do
    44      :: _nr_pr == 1 -> run AtomicSwap(i,i+1); i = i+2
    45      :: i == 8 -> break
    46      od
    47      printf("cell:")
    48      for (i in cell) {
    49          printf("%d",cell[i])
    50      }
    51      printf("\n")
    52  }
```

a) (atomic_swap_1.pml) (2 puntos – 18 min.) El programa intercambia los valores del arreglo **cell** en las posiciones pares e impares:

```
$ spin atomic_swap_1.pml
      cell:      8      1      2      3      4      5      6      7
      cell:      1      8      3      2      5      4      7      6
5 processes created
```

Pero tiene un error (por lo menos :-)). ¿Cuál es y cómo corregirlo? Presente su material de trabajo.

Aparentemente el programa **atomic_swap_2.pml** que, a diferencia del programa **atomic_swap_1.pml**, hace los intercambios de valores en paralelo en vez de secuencialmente, no tiene errores:

```
$ cat -n atomic_swap_2.pml
     1  #define lock(sem)      atomic { sem > 0; sem-- }
     2  #define unlock(sem)    sem++
     3
     4  #define print    printf("cell:"); \
     5                   for (i in cell) { \
     6                       printf("%d",cell[i]) \
     7                   }; \
     8                   printf("\n")
     9
    10  #define N 8
    11
    12  byte cell[N] = 0
    13  byte mutex[N]=1
    14
    15  proctype AtomicSwap(byte f, s) {
    16      byte item[2]
    17
    18      lock(mutex[f])
    19      item[0] = cell[f]
    20      if
    21      :: item[0] != 0 -> lock(mutex[s])
    22                         item[1] = cell[s]
    23                         if
    24                         :: item[1] != 0 ->
    25                                           cell[s] = item[0]
    26                                           cell[f] = item[1]
    27                                           unlock(mutex[s])
    28                                           unlock(mutex[f])
    29                         :: else -> skip
    30                         fi
    31      :: else -> skip
    32      fi
    33  }
    34
    35  init {
    36      byte i
    37
    38      cell[0]=8; cell[2]=2; cell[4]=4; cell[6]=6
    39      cell[1]=1; cell[3]=3; cell[5]=5; cell[7]=7
    40      print
    41
    42      atomic {
    43          run AtomicSwap(0,1); run AtomicSwap(2,3);
    44          run AtomicSwap(4,5); run AtomicSwap(6,7);
    45      }
    46
    47      _nr_pr == 1 -> print
    48      assert(cell[0]==1 && cell[2]==3 && cell[4]==5 && cell[6]==7 &&
    49             cell[1]==8 && cell[3]==2 && cell[5]==4 && cell[7]==6)
    50  }
```

b) (`atomic_swap_3.pml`) (5 puntos – 45 puntos) En esta versión del programa fueron hechos algunos cambios:

- después de recoger el valor, la celda se limpia con 0 (líneas 19 y 22);
- aleatoriamente, la celda puede tener un valor o puede ser vacía (con el valor 0). El intercambio de valores sucede solamente en el caso cuando ***ambas*** celdas no están vacías;
- las celdas cuyos valores se pretende intercambiar se eligen aleatoriamente;
- la celda no se puede intercambiar el valor con sí misma.

Aquí está el código del programa:

```
$ cat -n atomic_swap_3.pml
     1  #define lock(sem)      atomic { sem > 0; sem-- }
     2  #define unlock(sem)    sem++
     3
     4  #define print    printf("cell:"); \
     5                   for (i in cell) { \
     6                       printf("%d",cell[i]) \
     7                   }; \
     8                   printf("\n")
     9
    10  #define N 4
    11
    12  byte cell[N] = 0
    13  byte mutex[N]=1
    14
    15  proctype AtomicSwap(byte f, s) {
    16      byte item[2]
    17
    18      lock(mutex[f])
    19      item[0] = cell[f]; cell[f] = 0
    20      if
    21      :: item[0] != 0 -> lock(mutex[s])
    22                         item[1] = cell[s]; cell[s] = 0
    23                         if
    24                         :: item[1] != 0 ->
    25                                           cell[s] = item[0]
    26                                           cell[f] = item[1]
    27                                           unlock(mutex[s])
    28                                           unlock(mutex[f])
    29                         :: else -> skip
    30                         fi
    31      :: else -> skip
    32      fi
    33  }
    34
    35  init {
    36      byte i,j,k,l
    37
    38      do
    39      ::
    40          if :: cell[0]=8 :: cell[0]=0 fi; if :: cell[2]=2 :: cell[2]=0 fi
    41          if :: cell[1]=1 :: cell[1]=0 fi; if :: cell[3]=3 :: cell[3]=0 fi
    42          select (i : 0 .. 3)
    43          do
    44          :: select (j : 0 .. 3); if :: j != i -> break :: else -> skip fi
    45          od
    46          select (k : 0 .. 3)
    47          do
```

```
48          :: select (l : 0 .. 3); if :: l != k -> break :: else -> skip fi
49          od
50          printf("i=%d, j=%d, k=%d, l=%d\n", i,j,k,l)
51          atomic {
52              run AtomicSwap(i,j); run AtomicSwap(k,l)
53          }
54          _nr_pr == 1
55      od
56  }
```

Pero este programa tiene, por lo menos, 3 errores. Algunos son fáciles de encontrar analizando la lógica del intercambio en el mismo código. Es claro también que Spin le puede ayudar. Encuentre errores y corríjalos obteniendo la versión del programa **`atomic_swap_4.pml`** que podría funcionar algo así (lo presentado le puede ayudar a resolver el problema):

```
$ spin atomic_swap_4.pml | less
      i=0, j=1; k=0, l=1:      0 <-> 1; 0 <-> 1
      i=1, j=2; k=0, l=3:      0 <-> 0; 0 <-> 0
      i=0, j=3; k=0, l=3:      8 <-> 0; 8 <-> 0
      i=1, j=3; k=2, l=1:      0 <-> 3; 0 <-> 0
      i=1, j=0; k=2, l=3:      0 <-> 0; 2 <-> 3
              Swapped f=2, s=3:              3 <-> 2
      i=0, j=3; k=0, l=1:      0 <-> 0; 0 <-> 1
      i=2, j=0; k=1, l=2:      2 <-> 0; 1 <-> 2
              Swapped f=1, s=2:              2 <-> 1
      i=3, j=1; k=1, l=2:      0 <-> 1; 1 <-> 2
              Swapped f=1, s=2:              2 <-> 1
      i=0, j=1; k=3, l=2:      8 <-> 0; 0 <-> 2
      i=0, j=2; k=1, l=3:      8 <-> 2; 0 <-> 0
          Swapped f=0, s=2:          2 <-> 8
      i=3, j=2; k=2, l=0:      3 <-> 0; 0 <-> 0
      i=1, j=2; k=1, l=3:      1 <-> 2; 1 <-> 3
              Swapped f=1, s=3:              3 <-> 1
          Swapped f=1, s=2:          2 <-> 3
      i=2, j=1; k=3, l=0:      0 <-> 1; 3 <-> 8
              Swapped f=0, s=3:              3 <-> 8
      i=1, j=0; k=2, l=1:      0 <-> 0; 2 <-> 0
      i=3, j=0; k=1, l=2:      3 <-> 0; 0 <-> 0
      i=1, j=2; k=3, l=0:      0 <-> 2; 3 <-> 8
:
q
```

Pregunta 3. (**7 puntos – 63 min.**) (**tictactoe_v0.pml**) En este archivo se proporciona la planilla incompleta para el modelo del juego Tic-Tac-Toe:

```
$ cat -n bridge3.pml
     1  #define SQ(x,y) !b.r[x].s[y] -> b.r[x].s[y] = z+1
     2  #define H(v,w)  b.r[v].s[0]==w && ...
     3  #define V(v,w)  b.r[0].s[v]==w && ...
     4  #define UD(w)   b.r[0].s[0]==w && ...
     5  #define DD(w)   b.r[2].s[0]==w && ...
     6
     7  typedef Row   { byte s[3]; };
     8  typedef Board { Row  r[3]; };
     9
    10  Board b
    11  bit z, won
    12
    13  init {
    14      do
    15      :: atomic { /* do not store intermediate states */
    16           !won ->
    17               if  /* all valid moves */
    18               :: SQ(0,0)  :: SQ(0,1)  :: SQ(0,2)
    19               ...
    20               ...
    21               :: else -> break /* a draw: game over */
    22               fi
    23
    24               if  /* winning positions */
    25               :: H(0,z+1) || ... ||\
    26                  ...
    27                  ...  ->
    28                   /* print winning position */
    29                   printf("%d %d %d\n%d %d %d\n%d %d %d\n",
    30                       b.r[0].s[0], ...
    31                       ...
    32                       ...
    33                   won = true /* and force a stop */
    34               :: else -> z = 1 - z                  /* continue */
    35               fi
    36          } /* end of atomic */
    37      od
    38  }
```

Desarrolle el modelo **tictactoe_v1.pml** que permitiría observar el desarrollo del juego paso por paso (jugada por jugada). También es interesante descubrir con Spin qué opciones tiene el 2do jugador para ganar u obtener el empate después de que el 1er jugador marcó su primera jugada en una celda específica.

Profesor: V. Khlebnikov
Pando, 19 de octubre de 2018