Práctica SPIN

Joaquín Solla Vázquez
Lucas Campos Camiña

Índice

Archivos

Para la realización de la práctica se han creado 2 ficheros .pml:

- *semaforos_simple.pml*: La versión más simple del programa, un sistema de semáforos en el que cualquiera de los dos sentidos puede ponerse en verde (cumpliendo las restricciones safety) en un orden aleatorio.
  Se emplea en la resolución de la práctica definida en el enunciado.

- *semaforos_complejo.pml*: El programa más sofisticado, en el que los semáforos se alternan para ponerse en verde (..., V, H, V, H, V, H…).
  Además, cada sentido cuenta con 2 semáforos (semV, semV_clon, semH y semH_clon) tal y como se ilustra en la foto del enunciado de la práctica.
  Ambos semáforos de cada sentido tienen el mismo comportamiento.
  Se emplea para resolver la parte opcional (extra) de la práctica.

Aclaraciones sobre el código

La ejecución de ambos programas es infinita.
Las luces de los semáforos están definidas con un mtype y tienen 3 posibles valores: green, yellow y red.
El código comienza con el proceso *init* el cual se encarga de arrancar a los procesos V (vertical) y H (horizontal) que gestionan el funcionamiento de sus respectivos semáforos (semV para el semáforo vertical y semH para el semáforo horizontal).

La salida por terminal de los programas sigue el siguiente formato:

```
V: RED
H: RED
        V: GREEN
        V: YELLOW
        V: RED
        —--------
                H: GREEN
                H: YELLOW
                H: RED
                —--------
        V: GREEN
        V: YELLOW
        V: RED
        —--------
```

El estado inicial de ambos semáforos (rojo) y, posteriormente, comienza el intercambio de colores.

Comprobación de “Deadlocks”

Para demostrar que el programa no sufre deadlocks, se han ejecutado los siguientes comandos (modo verificación):

spin -a semaforos_simple.pml
gcc -o pan pan.c
./pan

Se obtiene la siguiente salida:

```
joaquin@joaquin-VirtualBox:~/Escritorio/Practica_Spin$ ./pan

(Spin Version 6.5.2 -- 6 December 2019)
        + Partial Order Reduction

Full statespace search for:
        never claim             - (none specified)
        assertion violations    +
        acceptance   cycles     - (not selected)
        invalid end states      +

State-vector 36 byte, depth reached 6, errors: 0
       10 states, stored
        2 states, matched
       12 transitions (= stored+matched)
        0 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
    0.001       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


unreached in proctype V
        semaforos_simple.pml:23, state 14, "-end-"
        (1 of 14 states)
unreached in proctype H
        semaforos_simple.pml:40, state 14, "-end-"
        (1 of 14 states)
unreached in init
        (0 of 5 states)

pan: elapsed time 0.11 seconds
pan: rate 90.909091 states/second
```

Se pueden obtener la siguientes conclusiones:
“Errors: 0”, por lo que SPIN nos indica que el programa no sufre errores ni deadlocks.
A mayores, no se ha generado ningún archivo .trail, que justifica una vez más que no se han encontrado errores.
Esto verifica que en ningún momento los semáforos se van a quedar bloqueados esperando a que el otro cambie el estado, sino que la ejecución es infinita.

Comprobación de “Safety”

Se debe comprobar que el escenario de “ambos semáforos están NO rojos a la vez” no ocurre nunca.

Para ello se va a comprobar (que no ocurre) la siguiente condición:

[]!(semV != red && semH != red)
(Siempre negación de los dos semáforos NO rojos a la vez)

Por lo que se ejecutan los siguientes comandos:

spin -a -f ‘![]!(semV != red && semH != red)’ semaforos_simple.pml
gcc -o pan pan.c
./pan -a -f

Se obtiene la siguiente salida:

```
joaquin@joaquin-VirtualBox:~/Escritorio/Practica_Spin$ ./pan -a -f
warning: for p.o. reduction to be valid the never claim must be stutter-invariant
(never claims generated from LTL formulae are stutter-invariant)

(Spin Version 6.5.2 -- 6 December 2019)
        + Partial Order Reduction

Full statespace search for:
        never claim             + (never_0)
        assertion violations    + (if within scope of claim)
        acceptance   cycles     + (fairness enabled)
        invalid end states      - (disabled by never claim)

State-vector 44 byte, depth reached 13, errors: 0
       10 states, stored
        2 states, matched
       12 transitions (= stored+matched)
        0 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
    0.001       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


unreached in proctype V
        semaforos_simple.pml:23, state 14, "-end-"
        (1 of 14 states)
unreached in proctype H
        semaforos_simple.pml:40, state 14, "-end-"
        (1 of 14 states)
unreached in init
        (0 of 5 states)
unreached in claim never_0
        semaforos_simple.pml.nvr:10, state 10, "-end-"
        (1 of 10 states)

pan: elapsed time 0.02 seconds
pan: rate       500 states/second
```

SPIN devuelve 0 errores, por lo que podemos afirmar que la condición indicada nunca ocurre, es decir, nunca están los dos semáforos a la vez en un estado diferente del rojo.
En caso de sí darse la condición, SPIN habría detectado errores, y tendrían que analizarse a través del archivo .trail que sería generado.

Comprobado esto, se verifica que los semáforos van a funcionar de forma correcta y en ningún momento ambos sentidos van a estar abiertos al tráfico a la vez.

Comprobación de “Liveness”

Se nos pide que comprobemos si ambos semáforos pueden alcanzar cada cada color infinitamente.
Como la ejecución de este programa es infinita, debemos demostrar esto de forma cíclica.
Para demostrar si esto es así, definimos la condición:

[](<>(semH != red) && <>(semV != red))

Siempre (en algún momento semH es diferente de rojo y en algún momento semV es diferente de rojo).

Para probar la condición ejecutamos los siguientes comandos (probando que NO ocurre la condición):

spin -a -f ‘![](<>(semH != red) && <>(semV != red))’ semaforos_simple.pml
gcc -o pan pan.c
./pan -a -f

Obtenemos esta salida:

```
joaquin@joaquin-VirtualBox:~/Escritorio/Practica_Spin$ ./pan -a -f
warning: for p.o. reduction to be valid the never claim must be stutter-invariant
(never claims generated from LTL formulae are stutter-invariant)
pan:1: acceptance cycle (at depth 22)
pan: wrote semaforos_simple.pml.trail

(Spin Version 6.5.2 -- 6 December 2019)
Warning: Search not completed
        + Partial Order Reduction

Full statespace search for:
        never claim             + (never_0)
        assertion violations    + (if within scope of claim)
        acceptance   cycles     + (fairness enabled)
        invalid end states      - (disabled by never claim)

State-vector 44 byte, depth reached 37, errors: 1
        8 states, stored (23 visited)
        5 states, matched
       28 transitions (= visited+matched)
        0 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
    0.001       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
```

“Errors: 1”, es decir, la condición que buscábamos no se cumple, por lo que procedemos a comprobar el archivo .trail para ver un ejemplo de ciclo en el que la condición no se cumple:

```
joaquin@joaquin-VirtualBox:~/Escritorio/Practica_Spin$ spin -t semaforos_simple.pml
spin: couldn't find claim 3 (ignored)
          V: RED
          H: RED
                    H: GREEN
                    H: YELLOW
                    H: RED
                    ------
                    H: GREEN
                    H: YELLOW
                    H: RED
                    ------
  <<<<<START OF CYCLE>>>>>
                    H: GREEN
                    H: YELLOW
                    H: RED
                    ------
                    H: GREEN
                    H: YELLOW
                    H: RED
                    ------
spin: trail ends after 38 steps
#processes: 3
                 semV = red
                 semH = red
 38:     proc  2 (H:1) semaforos_simple.pml:26 (state 11)
 38:     proc  1 (V:1) semaforos_simple.pml:9 (state 11)
 38:     proc  0 (:init::1) semaforos_simple.pml:47 (state 5) <valid end state>
3 processes created
```

En el ciclo que nos devuelve SPIN, el semáforo horizontal siempre “roba el turno” al vertical, por lo que no se cumple la condición de que ambos semáforos pasen por todos los colores infinitamente (en este caso solo lo hace el horizontal).

Tras esta demostración, sabemos que con este programa pueden darse ejecuciones en las que solo se abra un semáforo de forma infinita (aunque sea muy poco probable).

EXTRA - Solución al problema de “Liveness”

Para solucionar el problema del apartado anterior y lograr que se cumpla la condición definida, se ha tenido que mejorar el programa, dando resultado al archivo *semaforos_complejo.pml*.

En este programa se registra qué sentido ha sido el último en estar NO rojo, de forma que la siguiente vez ceda el turno al otro sentido.

Si el programa está bien definido, debería darse la condición del apartado anterior, se ha modificado ligeramente para aplicarse a los 2 semáforos de cada sentido (aunque a efectos prácticos no cambiaría nada si solo se usase un semáforo por sentido):

[](<>(semH != red && semH_clon != red) && <>(semV != red && semV_clon != red))

Se realiza la prueba (una vez más negando la condición que buscamos que se cumpla):

```
joaquin@joaquin-VirtualBox:~/Escritorio/Practica_Spin$ ./pan -a -f
warning: for p.o. reduction to be valid the never claim must be stutter-invariant
(never claims generated from LTL formulae are stutter-invariant)

(Spin Version 6.5.2 -- 6 December 2019)
        + Partial Order Reduction

Full statespace search for:
        never claim             + (never_0)
        assertion violations    + (if within scope of claim)
        acceptance   cycles     + (fairness enabled)
        invalid end states      - (disabled by never claim)

State-vector 44 byte, depth reached 27, errors: 0
       34 states, stored (98 visited)
       55 states, matched
      153 transitions (= visited+matched)
        0 atomic steps
hash conflicts:         0 (resolved)

Stats on memory usage (in Megabytes):
    0.002       equivalent memory usage for states (stored*(State-vector + overhead))
    0.287       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 V
        semaforos_complejo.pml:34, state 18, "-end-"
        (1 of 18 states)
unreached in proctype H
        semaforos_complejo.pml:55, state 18, "-end-"
        (1 of 18 states)
unreached in init
        (0 of 5 states)
unreached in claim never_0
        semaforos_complejo.pml.nvr:17, state 20, "-end-"
        (1 of 20 states)

pan: elapsed time 0 seconds
```

SPIN devuelve 0 errores, por lo que la condición SÍ se cumple. Podemos asegurar que con este programa los dos semáforos van a pasar por todos los colores infinitamente.