Course "Formal Methods"
Lab Test

Roberto Sebastiani
DISI, Università di Trento, Italy

September 07^{th}, 2015

769857918

[COPY WITH SOLUTIONS]

Implement a 4-bit shifter with alternating behaviour. Use a variable "`out`" to represent the value of the shifter. Use four boolean variables "`b0`", "`b1`", "`b2`", "`b3`" to represent the bits of the shifter, from the least-significative to the most-significative one. Use a variable "`d`", with values in the set {`L, R`}, to keep track of the current shift direction.
Initially, the shift direction is set to `R`, "`b`" is `TRUE` and all other bits are `FALSE`.
Implement, using the **assignment-style** approach, the following transitions:

- "`d`" is set to `R` if "`b1`" is `TRUE`, is set to `L` if "`b2`" is `TRUE` and keeps its value otherwise.
- "`b0`" is set to `FALSE` if `d=R`, is set to `TRUE` if `d=L&b1` and keeps its value otherwise.
- "`b1`" is set to `TRUE` if `(d=L&b2)|(d=R&b0)`, is set to `FALSE` if "`b1`" is `TRUE` and keeps its value otherwise.
- "`b2`" is set to `TRUE` if `(d=L&b3)|(d=R&b1)`, is set to `FALSE` if "`b2`" is `TRUE` and keeps its value otherwise.
- "`b3`" is set to `FALSE` if `d=L`, is set to `TRUE` if `(d=R&b2)` and keeps its value otherwise.

Check that the simulation returns the sequence of values `1, 2, 4, 8, 4, 2, 1, 2, ...` for the output variable "`out`".
Model the 4-bit shifter, Express the following properties, and have nuXmv verify that all properties are `TRUE`.

- it is always the case that at least one bit changes value at each transition
- it is always the case that if `d` is `R` and a bit b_i is `TRUE`, then b_{i+1} will be `TRUE` at the next state
- it is always the case that if `d` is `L` and a bit b_{i+1} is `TRUE`, then b_i will be `TRUE` at the next state
- it always the case that if `b0` is `TRUE` then `d` is `R` and if `b3` is `TRUE` then `d` is `L`
- infinitely often the value of the shifter is 8

Solution:

```
MODULE main
VAR
  b0: boolean; b1: boolean; b2: boolean;
  b3: boolean; d: {L, R};

DEFINE
  out := toint(b0) + 2*toint(b1) + 4*toint(b2) + 8*toint(b3);

ASSIGN
  init(b0) := TRUE; init(b1) := FALSE; init(b2) := FALSE;
  init(b3) := FALSE; init(d) := R;

  next(d) := case
    b1          : R;
    b2          : L;
    TRUE        : d;
  esac;
  next(b0) := case
    d = R       : FALSE;
    d=L&b1      : TRUE;
    TRUE        : b0;
  esac;
  next(b1) := case
    d = L & b2 : TRUE;
    d = R & b0 : TRUE;
    b1          : FALSE;
    TRUE        : b1;
  esac;
  next(b2) := case
    d = L & b3 : TRUE;
    d = R & b1 : TRUE;
    b2          : FALSE;
    TRUE        : b2;
  esac;
  next(b3) := case
    d = L       : FALSE;
    d = R & b2 : TRUE;
    TRUE        : b3;
  esac;

-- it is always the case that at least one bit changes value at each
-- transition
CTLSPEC AG (
           (b0<->!AX b0)|
```

```
            (b1<->!AX b1)|
            (b2<->!AX b2)|
            (b3<->!AX b3)
)

-- it is always the case that if a d is R and a bit b_i is true,
-- then b_(i+1) will be true at the next state
CTLSPEC AG (
             ((d=R&b0)->AX b1)&
             ((d=R&b1)->AX b2)&
             ((d=R&b2)->AX b3)
)

-- it is always the case that if a d is L and a bit b_(i+1) is true,
-- then b_(i) will be true at the next state
CTLSPEC AG (
             ((d=L&b1)->AX b0)&
             ((d=L&b2)->AX b1)&
             ((d=L&b3)->AX b2)
)

-- it always the case that if b0 is true then d is R and if
-- b3 is true then d is L
CTLSPEC AG ((b0 -> d=R) & (b3 -> d=L))

-- infinitely often the value of the counter is 8
LTLSPEC G F (out = 8)
```

1 Spin

Write a Promela program that runs 10 processes of type P, each of which accesses a shared critical section. Use lock and counter as global variables, respectively initialized to false and 0. The pseudo-code of the P process, based on the well-known TestAndSet() mutual-exclusion mechanism, is the following:

```
P() {
  bool tmp;
  while (true) {
    while(true) {
        atomic { tmp = lock; lock = true; }
        if (!tmp) break;
    }

    // critical section
    counter++;
    assert(counter == 1);
    counter--;

    // exiting...
    lock = false;

    // non-critical section
  }
}
```

Verify that the critical section is not accessed by more than one process at the same time by generating a verifier from the Promela program:

```
~$ spin -a <file_name>.pml
~$ gcc pan.c
~$ ./a.out
```

Solution:

```
bool lock = false;
int counter = 0;

active [10] proctype P()
{
    bool tmp;
    do
        :: true ->
        do
            :: atomic {tmp = lock; lock = true;} ->
            if
                :: tmp;
                :: else -> break;
            fi;
        od;

        // critical section
        counter++;
        printf("Process %d in critical section.\n", _pid);
        assert(counter == 1);
        counter--;

        // exiting...
        lock = false;

        // non-critical section
    od;
}
```