Course "Formal Methods"
Lab Test

Roberto Sebastiani
DISI, Università di Trento, Italy

June 18^{th}, 2015

976985749

[COPY WITH SOLUTIONS]

Implement a 4-bit counter that counts all even numbers starting from 0 *(e.g. 0, 2, 4, 6, 8, 10, 12, 14, 0, 2, ...)* when the "`reset`" input is FALSE. The counter shall always be immediately set to 0 when the "`reset`" input is TRUE. Use a variable "`out`" to represent the output of the counter. Use four boolean variables "`b0`", "`b1`", "`b2`", "`b3`" to represent the bits of the counter, from the least-significative to the most-significative ones. Notice that, assuming `reset == FALSE`, the following is true:

- "`b0`" is always FALSE
- "`b1`" changes value at each transition
- "`b2`" changes value only when "`b1`" is TRUE
- "`b3`" changes value only when both "`b1`" and "`b2`" are TRUE

b3	b2	b1	b0	out
0	0	0	0	0
0	0	1	0	2
0	1	0	0	4
0	1	1	0	6
1	0	0	0	8
1	0	1	0	10
1	1	0	0	12
1	1	1	0	14

Figure 1: bits evolution at each transition

Model the 4-bit counter, express the following properties, and have nuXmv verify them or provide a counter-examples.

- CTL Properties:
  - it is never the case that the counter is odd
  - it is necesarily always the case that when reset is true or the number is 14, then necessarily at the next step the value of the counter is 0
  - it is always the case that if reset is FALSE, then the next value of b1 is !b1
  - it is always the case that, if both b1 and b2 are TRUE, then the next value of b3 is equal to !b3
- LTL Properties:
  - infinitely often the value of the counter is 0
  - infinitely often the value of the counter is 2
  - if reset is always false, then infinitely often the value of the counter is 2

Solution:

```
MODULE main
VAR
  b0: boolean;
  b1: boolean;
  b2: boolean;
  b3: boolean;
  reset: boolean;

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

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

  next(b0) :=             FALSE;
  next(b1) := case
                reset :   FALSE;
                TRUE :    !b1;
              esac;
  next(b2) := case
                reset :   FALSE;
                b1 :      !b2;
                TRUE :    b2;
              esac;
  next(b3) := case
                reset :   FALSE;
                b1 & b2 : !b3;
                TRUE :    b3;
              esac;

-- it is never the case that the counter is odd
CTLSPEC AG !(b0 = TRUE)

-- it is necesarily always the case that when reset is true or the number
-- is 14, then necessarily at the next step the value of the counter is 0
CTLSPEC AG ((reset | out=14) -> AX (out=0))

-- it is always the case that if reset is FALSE, then the  next value of b1 is !b1
CTLSPEC AG ((!reset&b1 -> AX !b1) &
            (!reset&!b1 -> AX b1))
```

```
-- it is always the case that, if both b1 and b2 are TRUE,
-- then the next value of b3 is equal to !b3
CTLSPEC AG ((b1&b2&b3 -> AX !b3) &
            (b1&b2&!b3 -> AX b3))

-- infinitely often the value of the counter is 0
LTLSPEC G F (out = 0)

-- infinitely often the value of the counter is 2
LTLSPEC G F (out = 2)

-- if reset is always false, then infinitely often the value of the counter is 2
LTLSPEC (G ! reset) -> (G F (out = 2))
```

1 Spin

Write a Promela program that initializes an array of 10 integers and computes the product of the values in the array. The initialization procedure should non-deterministically assign a random value in the range [0..9] to each array location. Both the content of the array and the final product should be printed on screen. Execute a simulation of the program and visually check that the computed value is correct.

Solution:

```
#define N 10

init {
    int a[N];
    int i = 0;
    int product = 1;

    do
        :: (i >= N) -> break;
        :: else ->
            if
                :: true -> a[i] = 0
                :: true -> a[i] = 1
                :: true -> a[i] = 2
                :: true -> a[i] = 3
                :: true -> a[i] = 4
                :: true -> a[i] = 5
                :: true -> a[i] = 6
                :: true -> a[i] = 7
                :: true -> a[i] = 8
                :: true -> a[i] = 9
            fi;
            printf("a[%d] = %d\n", i, a[i]);
            i++;
    od;

    i = 0;

    do
        :: (i >= N) -> break
        :: else ->
            product = product * a[i];
            i++;
    od;
    printf("The product is: %d\n", product);
}
```