Course "Formal Methods"
Lab Test

Roberto Sebastiani
DISI, Università di Trento, Italy

February 11^{th}, 2016

769857918

[COPY WITH SOLUTIONS]

1 nuXmv

Implement a 4-bit counter that counts all odd numbers starting from 1 *(e.g. 1, 3, 5, 7, 9, 11, 13, 15, 1, 3, ...)* when the "`reset`" input is FALSE. The counter shall always be immediately set to 1 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 TRUE
- "`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	1	1
0	0	1	1	3
0	1	0	1	5
0	1	1	1	7
1	0	0	1	9
1	0	1	1	11
1	1	0	1	13
1	1	1	1	15

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-example.

- CTL Properties:
  - it is never the case that "`b0`" is FALSE
  - it is necesarily always the case that when reset is true or the number is 15, then necessarily at the next step the value of the counter is 1
  - 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 1
  - infinitely often the value of the counter is 3
  - if reset is always FALSE, then infinitely often the value of the counter is 3

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) := TRUE;
  init(b1) := FALSE;
  init(b2) := FALSE;
  init(b3) := FALSE;

  next(b0) := TRUE;
  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 even
CTLSPEC AG !(b0 = FALSE)

-- it is necesarily always the case that when reset is true or the number
-- is 15, then necessarily at the next step the value of the counter is 1
CTLSPEC AG ((reset | out=15) -> AX (out=1))

-- 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 1
LTLSPEC G F (out = 1)

-- infinitely often the value of the counter is 3
LTLSPEC G F (out = 3)

-- if reset is always false, then infinitely often the value of the counter is 3
LTLSPEC (G !reset) -> (G F (out = 3))
```

2 Spin

Write a promela program defining a process ‘‘`fact(n, p)`’’ to calculate recursively the factorial of ‘‘`n`’’, communicating the result via a **channel message** to its parent process ‘‘`p`’’. In the init function, use that process to compute ‘‘`fact(k)`’’ and verify that it is greater than 2^k for $k > 3$. (e.g., try with $k = 10$).

Solution:

```
proctype fact(int n; chan p)
{
  int result;
  chan child = [1] of { int };

  if
    :: (n <= 1) -> p!1
    :: (n >= 2) ->
      run fact(n-1, child);
      child?result;
      p!n*result
  fi
}

init
{
  int result;
  chan child = [1] of { int };
  run fact(10, child);
  child?result;
  printf("result: %d\n", result);
}
```