Course "Formal Methods"
Lab Test

Roberto Sebastiani
DISI, Università di Trento, Italy

January 13^{th}, 2016

769857918

[COPY WITH SOLUTIONS]

1 nuXmv

Implement a 3-bit counter which counts the number of times an input boolean variable "bin" changes value from FALSE to TRUE. Use three boolean variables "b0", "b1", "b2" to represent the bits of the counter, from the least-significant to the most-significant one. Use an output variable "out" to represent the value of the counter. Use a variable "overflow", with values in the set {NO, YES}, to keep track of a counter overflow event. Use a variable "obin" to keep track of the previous value of the input variable "bin", and an output variable "rise" to express the fact that "bin" changed value from FALSE to TRUE in the current step. Use an input boolean variable "reset" to reset the value of "b0", "b1", "b2", "obin" and "reset" to their initial value. Initially, "b0", "b1", "b2", "bin" and "obin" should be set to FALSE, while "overflow" should evaluate 'NO'. Implement, using the assign-syntax, the following transitions:

- "obin" is set to FALSE if "reset" is TRUE, and to "bin" otherwise
- "b0" is set to FALSE if "reset" is TRUE, it is set to "!b0" if "rise" is TRUE, and keeps its value otherwise
- "b1" is set to FALSE if "reset" is TRUE, it is set to "!b1" if "rise & b0" is TRUE, and keeps its value otherwise
- "b2" is set to FALSE if "reset" is TRUE, it is set to "!b2" if "rise & b0 & b1" is TRUE, and keeps its value otherwise
- "overflow" is set to 'NO' if "reset" is TRUE, it is set to 'YES' if "rise & b0 & b1 & b2" is TRUE, and keeps its value otherwise

Manually verify that the simulation works as intended. Express the following properties, and have nuXmv verify that all properties are FALSE.

- CTL: it is necessarily always the case that infinitely often the counter is 0
- CTL: it is necessarily always the case that eventually the counter is always different than 0
- CTL: it is necessarily always the case that , if "overflow" is 'YES' in a given state then it also holds that "overflow" is 'YES' until "reset"
- CTL: it is necessarily always the case that when "b0", "b1" and "b2" are TRUE then from the next state eventually the value of counter will go back to 0
- LTL: if "rise" is TRUE infinitely often, then "overflow" is 'YES' infinitely often as well
- Bonus Point: explain why the latter formula is verified if CTL is used instead of LTL.

Solution:

```
MODULE main
VAR
  bin: boolean; obin: boolean; reset: boolean;
  b0: boolean; b1: boolean; b2: boolean;
  overflow: {NO, YES};

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

DEFINE
  rise := bin & !obin;

ASSIGN
  init(b0) := FALSE; init(b1) := FALSE; init(b2) := FALSE;
  init(bin) := FALSE; init(obin) := FALSE; init(reset) := FALSE;
  init(overflow) := NO;

  next(obin) := case
    reset : FALSE;
    TRUE  : bin;
  esac;

  next(b0) := case
    reset : FALSE;
    rise  : !b0;
    TRUE  : b0;
  esac;

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

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

  next(overflow) := case
    reset               : NO;
    rise & b0 & b1 & b2 : YES;
    TRUE                : overflow;
```

```
  esac;

-- it is necessarily always the case that infinitely often the counter is 0
CTLSPEC AG AF (out = 0)

-- it is necessarily always the case that eventually the counter is
-- always different than 0
CTLSPEC AF EG !(out = 0)

-- it is necessarily always the case that, if ‘‘overflow’’ is ’YES’ in a given
-- state then it also holds that ‘‘overflow’’ is ’YES’ until ‘‘reset’’
CTLSPEC AG (overflow = YES -> A[overflow = YES U reset])

-- it is necessarily always the case that when ‘‘b0’’, ‘‘b1’’ and ‘‘b2’’ are true
-- then from the next state eventually the value of counter will go back to 0
CTLSPEC AG (out=7 -> AX AF(out=0))

-- if ‘‘rise’’ is TRUE infinitely often, then ‘‘overflow’’ is ’YES’ infinitely often
-- as well (LTL)
LTLSPEC (G F rise) -> (G F overflow = YES)

-- Bonus Point: explain why the latter formula is verified if CTL is used instead of LTL.
In CTL, the implication premise is trivially false.
```

2 Spin

Write a Promela program P that initializes two integer variables ‘‘`sum`’’ and ‘‘`v`’’ to 1 and 0 respectively. Then, the program P enters a loop from which it can exit only if ‘‘`sum`’’ is equal to zero. Inside the loop, P computes a random positive value included in $\{1, 3\}$ and assigns it to ‘‘`v`’’. Then, it updates the value of ‘‘`sum`’’ in the following way:

- if ‘‘`sum`’’ is positive valued, it subtracts ‘‘`v`’’ to its value
- otherwise, it adds ‘‘`v`’’ to its value

Verify that there exists at least one execution in which the program terminates by appending `assert(false);` to the source code of P and by generating a verifier from the Promela program:

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

Solution:

```
active [1] proctype P ()
{
  int sum = 1;
  int v = 0;
  do
    :: 0 == sum ->
      break;
    :: else ->
      if
        :: true -> v = 1;
        :: true -> v = 2;
        :: true -> v = 3;
      fi;
      if
        :: sum > 0 ->
          sum = sum - v
          printf("- v %d\n", v);
        :: sum < 0 ->
          sum = sum + v
          printf("+ v %d\n", v);
      fi;
  od;
  printf("exit\n");
  assert(false);
}
```