Course "Formal Methods"
Lab Test

Roberto Sebastiani
DISI, Università di Trento, Italy

May 27^{th}, 2016

769857918

[COPY WITH SOLUTIONS]

1 nuXmv

Encode the following implementation of Insertion Sort for arrays of length 5 in nuXmv:

```
void isort(arr) {
      // init: i = 1, j = 1;
l1:   while (i < 5) {
l2:     j = i;
l3:     while (j > 0 & array[j] < array[j-1]) {
l4:       swap(array[j], array[j-1]);
l5:       j--;
        }
l6:     i++;
      }
l7:   // done!
```

Hints:

- use ‘pc’ to keep track of the possible state values { l1, l2, l3, l4, l5, l6, l7 }
- declare ‘i’ in 1..5, initialize 1
- declare ‘j’ in 0..4, initialize 1
- ensure that the content of ‘arr‘ does never change when ‘pc != l4’
- ensure that the content of ‘arr‘ that is **not** involved in a ‘swap’ operation does not change even when ‘pc = l4’
- *(easier?)* encode the constraints over ‘arr’ with constrained-style modelling
- *(easier?)* encode the evolution of ‘pc’, ‘i’ and ‘j’ with assignment-style modelling

Add the following code to initialize the system

```
MODULE main
VAR
  arr    : array 0..4 of 1..10;
  sorter : isort(arr);

INIT
  arr[0] = 9 & arr[1] = 7 & arr[2] = 5 &
  arr[3] = 3 & arr[4] = 1;
```

and verify that all the following properties are found **TRUE** by nuXmv:

- the algorithm always terminates
- eventually in the future, the array will be sorted forever
- the algorithm is not done (pc = l7) until the array is sorted

Solution:

```
MODULE main
VAR
  arr    : array 0..4 of 1..10;
  sorter : isort(arr);

INIT
  arr[0] = 9 & arr[1] = 7 & arr[2] = 5 &
  arr[3] = 3 & arr[4] = 1;

MODULE isort(arr)
VAR
  pc : { l1, l2, l3, l4, l5, l6, l7 };
  i  : 1..5; j  : 0..4;

ASSIGN
  init(pc) := l1;
  init(i)  := 1;
  init(j)  := 1;

  next(i) := case
    pc = l6 & i < 5 : i + 1;
    TRUE            : i;
  esac;

  next(j) := case
    pc = l2 & i != 5 : i;
    pc = l5 & j > 0  : j - 1;
    TRUE             : j;
  esac;

DEFINE
  done := pc = l7;

-- FSM
TRANS
  pc = l1 -> ((i < 5  -> next(pc) = l2) &
              (i >= 5 -> next(pc) = l7))
TRANS
  pc = l2 -> next(pc) = l3
TRANS
  pc = l3 -> ((((j = 1 & arr[1] < arr[0]) | (j = 2 & arr[2] < arr[1]) |
                (j = 3 & arr[3] < arr[2]) | (j = 4 & arr[4] < arr[3])
               ) -> next(pc) = l4
              ) & (j <= 0 |
```

```
              ((j = 1 & arr[1] >= arr[0]) | (j = 2 & arr[2] >= arr[1]) |
               (j = 3 & arr[3] >= arr[2]) | (j = 4 & arr[4] >= arr[3])
              ) -> next(pc) = l6
            ))
TRANS
  pc = l4 -> (next(pc) = l5
              & (j = 1 -> next(arr[0]) = arr[1] & next(arr[1]) = arr[0])
              & (j = 2 -> next(arr[1]) = arr[2] & next(arr[2]) = arr[1])
              & (j = 3 -> next(arr[2]) = arr[3] & next(arr[3]) = arr[2])
              & (j = 4 -> next(arr[3]) = arr[4] & next(arr[4]) = arr[3])
              & (j != 1  -> next(arr[0]) = arr[0])
              & (j != 1 & j != 2 -> next(arr[1]) = arr[1])
              & (j != 2 & j != 3 -> next(arr[2]) = arr[2])
              & (j != 3 & j != 4 -> next(arr[3]) = arr[3])
              & (j != 4 -> next(arr[4]) = arr[4])
             )
TRANS
  pc = l5 -> next(pc) = l3
TRANS
  pc = l6 -> next(pc) = l1
TRANS
  pc = l7 -> next(pc) = l7

-- prevent content of array to change non-deterministically
TRANS
  pc != l4 -> (next(arr[0]) = arr[0]
               & next(arr[1]) = arr[1]
               & next(arr[2]) = arr[2]
               & next(arr[3]) = arr[3]
               & next(arr[4]) = arr[4])

-- The algorithm always terminates
CTLSPEC AF AG done

-- Eventually in the future, the array will be sorted forever
CTLSPEC AF AG (arr[0] <= arr[1] & arr[1] <= arr[2] &
               arr[2] <= arr[3] & arr[3] <= arr[4])

-- The algorithm is not done until the array is sorted
CTLSPEC A[!done U (arr[0] <= arr[1] & arr[1] <= arr[2] &
                   arr[2] <= arr[3] & arr[3] <= arr[4])]
```

2 Spin

Write a Promela model for the "Prisoners' Dilemma". Two **prisoners** processes independently send a `CONFESS` or `DENY` message through a pair of **synchronous channels** to a **policeman** process. Each prisoner chooses the content of his message in a **random** fashion and independently from the other. The policeman receives the messages, decides an adequate penalty and sends back to each prisoner a `SENTENCE` message with the number of years he is supposed to spend in detention, using the same channel from which he received the initial message. The penalty is decided as follows:

- if both confess, then both spend 5 years each in prison.
- if one confesses and the other denies, then the former is free while the latter spends 20 years in prison.
- if both deny, then both spend 1 year each in prison.

Simulate the model and (visually) verify that it matches the description.
(optional) what is the best strategy for the prisoners?

Solution:

```
mtype { SENTENCE, DENY, CONFESS };
chan rooms[2] = [0] of { mtype, byte };

proctype prisoner(byte i)
{
  mtype v;
  byte years;
  if
    :: rooms[i] ! CONFESS, 0 ->
      printf("Prisoner %d confessed.\n", i);
    :: rooms[i] ! DENY, 0;
      printf("Prisoner %d denied.\n", i);
  fi;
  rooms[i] ? SENTENCE, years;
  printf("Prisoner %d was sentenced to %d years of detention.\n", i, years);
}

proctype policeman()
{
  mtype m1, m2;
  byte v1, v2;
  rooms[0] ? m1, 0;
  rooms[1] ? m2, 0;
  if
    :: m1 == CONFESS && m2 == CONFESS ->
      v1 = 5; v2 = 5;
    :: m1 == CONFESS && m2 == DENY ->
      v1 = 0; v2 = 20;
    :: m1 == DENY && m2 == CONFESS ->
      v1 = 20; v2 = 0;
    :: m1 == DENY && m2 == DENY ->
      v1 = 1; v2 = 1;
  fi
  atomic {
    printf("Sentence decided.\n")
    rooms[0] ! SENTENCE, v1;
    rooms[1] ! SENTENCE, v2;
  }
}

init {
  run policeman(2); run prisoner(0); run prisoner(1);
}
```