Course "Formal Methods"
Lab Test

Roberto Sebastiani
DISI, Università di Trento, Italy

July 13^{th}, 2017

769857918

[COPY WITH SOLUTIONS]

1 Spin

The *infinite monkey theorem* states that a monkey hitting keys at random on a typewriter keyboard for an infinite amount of time will almost surely type a given text, such as the complete works of William Shakespeare.

Model, using *Promela*, a system comprised by 26 *monkeys* and one human *reviewer*.

Each **monkey** sits in front of a table with only one button, each of which is assigned a **unique** lower-case character in the alphabet (i.e. a random character from 'a' to 'z' in the *ASCII* table, note that there are precisely 26 characters in this interval). Whenever the monkey pushes the button, the corresponding character is sent to the human operator to be reviewed. The system must be designed in such a way that no other monkey can send a new character to the human operator before the latter has finished reviewing the last received character. Each monkey is also instructed to stop doing any activity as soon as a red light-bulb, placed in front of the room, is turned on.

The human **reviewer** is given the task of checking the incoming sequence of characters against a famous quote from the *Hamlet "to be or not to be"*, ignoring spaces and punctuation marks which are not being typed by any monkey. As soon as there is a complete match of the full sentence, the human operator turns on the red light-bulb in the room and terminates. Otherwise, it keeps checking the incoming characters for a matching sequence.

Use a global, shared, channel `typewriter` to send characters from the *monkeys* to the the human *reviewer*.

Write an *LTL* property s.t. the corresponding counter-example found by spin is an execution trace matching the sequence of characters `tobeornottobe`, and use *Spin* to find it.

Solution:

```
#define NUM_MONKEYS ('z' - 'a' + 1)

chan typewriter = [0] of { byte };
bool matched = false;

active [NUM_MONKEYS] proctype monkey()
{
    do
        :: typewriter!('a' + _pid)
        :: matched -> break;
    od
}

active proctype reviewer()
{
    byte specimen[13] = { 't', 'o', 'b', 'e', 'o', 'r', 'n', 'o', 't', 't', 'o', 'b', 'e' };
    byte idx, c;

    do
        :: typewriter?c ->
            if
                :: specimen[idx] == c ->
                    printf("Match: %c -- %c\n", specimen[idx], c);
                    idx++;
                    if
                        :: idx == 13 ->
                            matched = true;
                            break;
                        :: else ->
                            skip;
                    fi
                :: else ->
                    idx = 0;
            fi
    od
}

ltl p1 { [] ! matched };
```

2 nuXmv

Encode the following *sorting algorithm* for an array of length 4 in NuSMVor nuXmv as a **module** `selectionSort(arr, len)`:

```
    procedure selectionSort(arr, len):
l0:     while (j < len - 1):
l1:         iMin = j
            i    = j + 1
l2:         while (i < len):
l3:             if (arr[i] < arr[iMin]):
l4:                 iMin = i
l5:             i++
l6:         if (iMin != j):
l7:             swap(a[j], a[iMin])
l8:         j++
l9:    return                    # self-loop here!
```

Initialize, inside the module `selectionSort`, the variables `i`, `j`, and `iMin` to be equal 0 and `pc` to be equal `l0`.

Declare, inside the **main** module, a variable `arr` and a variable `sorter`:

- `arr` is an array of 4 elements with domain in $[1, 10]$, initialised to { `9, 7, 5, 3` }
- `sorter` is an instance of `selectionSort(arr, len)` taking as input `arr` and its corresponding length, 4.

Verify that the following properties are `true`:

- the algorithm always terminates
- eventually in the future, the array will be sorted forever
- eventually the array is sorted, and the algorithm is not done until the array is sorted

Solution:

```
MODULE main()
VAR
    arr : array 0..3 of 1..10;
    sorter : selectionSort(arr, 4);

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

MODULE selectionSort(arr, len)
VAR
    i : 0..(len + 1);     iMin : 0..(len + 1);     j : 0..len;
    pc : { l0, l1, l2, l3, l4, l5, l6, l7, l8, l9 };

INIT i = 0 & j = 0 & iMin = 0 & pc = l0;

ASSIGN
    next(pc) := case
        pc = l8                                              : l0;
        pc = l0 & (j < len - 1)                              : l1;
        pc = l1 | pc = l5                                    : l2;
        pc = l2 & (i < len)                                  : l3;
        pc = l3 & (i < len & iMin < len & arr[i] < arr[iMin]) : l4;
        pc = l3 | pc = l4                                    : l5;
        pc = l2                                              : l6;
        pc = l6 & (iMin != j)                                : l7;
        pc = l6 | pc = l7                                    : l8;
        TRUE                                                 : l9;
    esac;

    next(i) := case
        pc = l1                 : j + 1;
        pc = l5 & (i < len + 1) : i + 1;
        TRUE                    : i;
    esac;

    next(j) := case
        pc = l8 & j < len : j + 1;
        TRUE              : j;
    esac;

    next(iMin) := case
        pc = l1 : j;
        pc = l4 : i;
        TRUE    : iMin;
    esac;

DEFINE
    done := pc = l9;

TRANS pc != l7 ->
          (next(arr[0]) = arr[0] & next(arr[1]) = arr[1] &
           next(arr[2]) = arr[2] & next(arr[3]) = arr[3]);
```

```
TRANS (pc = l7 & iMin = 0 & j < len) -> next(arr[0]) = arr[j];
TRANS (pc = l7 & iMin = 1 & j < len) -> next(arr[1]) = arr[j];
TRANS (pc = l7 & iMin = 2 & j < len) -> next(arr[2]) = arr[j];
TRANS (pc = l7 & iMin = 3 & j < len) -> next(arr[3]) = arr[j];

TRANS (pc = l7 & j = 0 & iMin < len) -> next(arr[0]) = arr[iMin];
TRANS (pc = l7 & j = 1 & iMin < len) -> next(arr[1]) = arr[iMin];
TRANS (pc = l7 & j = 2 & iMin < len) -> next(arr[2]) = arr[iMin];
TRANS (pc = l7 & j = 3 & iMin < len) -> next(arr[3]) = arr[iMin];

TRANS (pc = l7 & j != 0 & iMin != 0) -> next(arr[0]) = arr[0];
TRANS (pc = l7 & j != 1 & iMin != 1) -> next(arr[1]) = arr[1];
TRANS (pc = l7 & j != 2 & iMin != 2) -> next(arr[2]) = arr[2];
TRANS (pc = l7 & j != 3 & iMin != 3) -> next(arr[3]) = arr[3];


-- 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]);

-- Eventually the array is sorted, and 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])] ;
```