Exam Examples [*]

Patrick Trentin
patrick.trentin@unitn.it
http://disi.unitn.it/~trentin

Formal Methods Lab Class, May 19, 2015

Università degli Studi di
Trento

[*] These slides are derived from those by Stefano Tonetta, Alberto Griggio, Silvia Tomasi, Thi Thieu Hoa Le, Alessandra Giordani for FM lab 2005/14

Exam

Info:

- you will not be allowed to access internet
- you will have access to short manuals of both tools with essential syntax coverage
- the exam is an individual work, cheating is severely punished!

Exam

Info:

- you will not be allowed to access internet
- you will have access to short manuals of both tools with essential syntax coverage
- the exam is an individual work, cheating is severely punished!

Examples:

- cover Laboratory part **only**
- taken from last year (2013/14)
- warning: exams of this year are yet to be prepared
  - the number of exercises might vary
  - the type of exercise might vary
  - the difficulty in solving the exam should remain nearly the same

Example 1: nuXmv

- Implement **4-bit counter with reset** which counts **4 steps at a time** if the input ``reset'' is false, resetting to 0 if ``reset'' is true. Initially the counter is 0. Use a variable ``out'' to represent the output of the counter, ``reset'' for the reset input, and four variables ``b0'', ``b1'', ``b2'', ``b3'' to represent the bits, from the least-significative to the most-significative ones.
- Express the following properties, and have nuXmv verify them or have it find counter-examples.
  - In CTL:
    - it is never the case that the counter is 12;
    - it is necessarily always the case that, when reset is true, then necessarily at next step the value of the counter is 0.
  - In LTL:
    - infinitely often the value of the counter is 12;
    - It is always the case that, if the value of the counter is 8 and the counter is not reset, then at the next step the value of the counter is 12.

Example 1: nuXmv

```
MODULE main
VAR
  b0 : boolean; b1 : boolean; b2 : boolean; b3 : boolean;
  reset : boolean; out : 0..15;
ASSIGN
  init(b0) := FALSE; init(b1) := FALSE; init(b2) := FALSE; init(b3) := FALSE;
  next(b0) := FALSE; next(b1) := FALSE;
  next(b2) := case
      reset = TRUE : FALSE;
      reset = FALSE : !b2;
    esac;
  next(b3) := case
      reset : FALSE;
      TRUE : ((!b2 & b3) | (b2 & !b3));
    esac;
  out := toint(b0) + 2*toint(b1) + 4*toint(b2) + 8*toint(b3);

--- PROPERTIES
CTLSPEC AG !(out=12);
CTLSPEC AG (reset -> AX (out=0) );
LTLSPEC G F (out=12) ;
LTLSPEC G ((!reset & out=8) -> X out=12);
```

Example 1: spin

In a railway station **trains** are countinuously arriving and leaving. Goods are contained in some cargos and, depending on the weight, they are moved from/to either **trucks** or **vans**.
Write a Promela program that models this scenario considering **each cargo as a message** that should be sent/received through the right channel. Each **channel** (train, truck and van) can contain **16 cargos** as a maximum. The **maximum weight** of each cargo in a van is **128**.
You will need two processes:

- ``split'', that splits goods from the train channel, dividing them over the other two channels, truck and van, depending on the weight values attached
- ``merge'', that merges the two streams back into one, most likely in a different order, and writes it back into the train channel.

Here are the initial cargo weights on the train: $345, 12, 6777, 32, 0$;

Example 1: spin

```
#define MaxWeight 128
#define Size 16

chan train = [Size] of { short };
chan truck = [Size] of { short };
chan van = [Size] of { short };

proctype split()
{
  short cargo;
  do
    :: train?cargo ->
      if
        :: (cargo >= MaxWeight) ->
          truck!cargo
        :: (cargo < MaxWeight) ->
          van!cargo
      fi;
  od
}
```

```
proctype merge()
{
  short cargo;
  do
    ::
      if
        :: truck?cargo
        :: van?cargo
      fi;
      train!cargo;
  od
}

init
{
  train!345; train!12; train!6777;
  train!32; train!0;
  run split();
  run merge()
}
```

Example 2: nuXmv

Implement a **5-bit counter** that starts from 0 counts 1,3,7,15,31 and goes back to 0 (i.e: $0, 1, 3, 7, 15, 31, 0, 1, 3, 7, 15, 31, 0, ...$). Note that the next value is obtained multiplying by 2 and summing 1.
Use variable ``out'' to represent the output of the counter, and five bits to represent the bits. Express the following properties, and use nuXmv to check them.

- In CTL:
  - it is always the case that, when the number is even, the value of out is zero
  - after 3 iterations the number is 7
  - it is always the case that, if all the bits are set to TRUE then at the next step all the bits set to FALSE
- In LTL:
  - it is never the case that out is 31
  - it is never the case that out is greater than 31

Example 2: nuXmv

```
MODULE main
VAR
  b0 : boolean; b1 : boolean; b2 : boolean;
  b3 : boolean; b4 : boolean; out : 0..31;
ASSIGN
  init(b0) := FALSE; init(b1) := FALSE; init(b2) := FALSE;
  init(b3) := FALSE; init(b4) := FALSE;
  next(b0) := case b4 = TRUE : FALSE; TRUE : TRUE; esac;
  next(b1) := case b4 = TRUE : FALSE; TRUE : b0;   esac;
  next(b2) := case b4 = TRUE : FALSE; TRUE : b1;   esac;
  next(b3) := case b4 = TRUE : FALSE; TRUE : b2;   esac;
  next(b4) := case b4 = TRUE : FALSE; TRUE : b3;   esac;
  out := toint(b0) + 2*toint(b1) + 4*toint(b2) + 8*toint(b3) + 16*toint(b4);

--- PROPERTIES
CTLSPEC AG (!b0 -> out=0)
CTLSPEC AX(AX (AX (out=7)))
CTLSPEC AG ((b4 & b3 & b2 & b1 & b0) -> AX (!b4 & !b3 & !b2 & !b1 & !b0));
LTLSPEC G !(out = 31)
LTLSPEC G !(out>31)
```

Example 2: spin

Procedures in Promela can be modeled as processes, even recursive ones. Write a program defining a process **factorial(n, p)** to calculate recursively the **factorial** of n, communicating the result via a 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 $k = 10$).

Example 2: spin

```
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;
  assert(result > 1024);
  printf("result: %d\n", result)
}
```

Example 3: nuXmv

Implement a **5-bit shifter** that **divides** the integer number **by two** (i.e $21, 10, 5, 2, 1, 0, 0, 0..$), by shifting to the right each bit. Use a variable ``out'' to represent the output of the counter and five boolean variables to represent the bits of the number. Define variables ``next_out'' to represent the number divided by two and ``remainder'' to save the remainder if out is odd (i.e. $21 = 10 \cdot 2 + 1$)

Express the following properties, and check them with nuXmv:

- it is necessarily always the case that, when the number is even, the next value of mod should be zero
- it is always the case that, given that out evaluates to ZERO, all future divisions by 2 will evaluate to ZERO, mod included
- after 5 iterations the number should be 0
- it is always the case that the number divided by 2 is less than the current number

Example 3: nuXmv

```
MODULE main
VAR
  b0 : boolean; b1 : boolean; b2 : boolean;
  b3 : boolean; b4 : boolean; out : 0..31;
  next_out : 0..15; remainder : 0..1;
ASSIGN
  init(b0) := FALSE; init(b1) := FALSE; init(b2) := FALSE;
  init(b3) := FALSE; init(b4) := FALSE;
  next(b0) := b1; next(b1) := b2; next(b2) := b3;
  next(b3) := b4; next(b4) := FALSE;

  out := toint(b0) + 2*toint(b1) + 4*toint(b2) + 8*toint(b3) + 16*toint(b4)
  next_out := out/2;
  remainder := out mod 2;

--- PROPERTIES
CTLSPEC AG ((!b0) -> AX (remainder=0));
CTLSPEC AG (out=0 -> AX (next_out=0 & remainder=0));
CTLSPEC AX (AX (AX (AX (AX (out=0)))));
LTLSPEC G (next_out<out);
```

Example 3: spin

In each sentence (string hereafter) the number of the characters composing the string is greater or equal than the number of the words contained in the sentence. A word is characterized by delimiters:

- space ' '
- tabulation '\t'
- endline '\n'

Write a spin function **count()** that perfoms property-based slicing of a string channel, counts the number of characters **nc** and the number of words **nw** and checks if the property $nc >= nw$ is always true.

Use the init function to pass to count() a string (remember that you can model a string as a channel of integers corresponding to ascii characters).

Example 3: spin

```
chan text = [40] of { short };
int c, nw, nc;

proctype count()
{
  bool inword = false;
  do
    :: text?c ->
      printf("%c",c);
      if
        :: c != ’\n’ -> nc++
        :: else /* do nothing */
      fi;
      if
        :: c == ’ ’ || c == ’\t’ ->
          inword = false
        :: c == ’\n’ ->
          break;
        :: else ->
          if
            :: !inword ->
              nw++; inword = true
            :: else /* do nothing */
          fi
      fi
  od;
  assert(nc >= nw);
  printf("%d\t%d\n", nw, nc)
}

init
{
  text!’I’;
  text!’ ’;
  text!’d’;
  text!’o’;
  text!’\n’;
  run count();
}
```