

# nuXmv: bounded model checking \*

Patrick Trentin

`patrick.trentin@unitn.it`

`http://disi.unitn.it/~trentin`

Formal Methods Lab Class, Apr 28, 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

# Bounded Model Checking

## Key ideas:

- looks for counter-example paths of increasing length  $k$  (called *bound*) (i.e. path consisting of  $k + 1$  states)
  - oriented to finding bugs: *is there a bad behaviour?*
- for each  $k$ , builds a boolean formula that is satisfiable iff there is a counter-example of length  $k$ 
  - can be expressed using  $k \cdot |s|$  variables
- satisfiability of the boolean formulas is checked using a *SAT procedure*
  - can manage complex formulas on several 100K variables
  - returns satisfying assignment (i.e., a counter-example)

# Example: modulo 8 counter [1/2]

## Example:

```
MODULE main
VAR
  b0    : boolean;
  b1    : boolean;
  b2    : boolean;
ASSIGN
  init(b0) := FALSE;
  init(b1) := FALSE;
  init(b2) := FALSE;
  next(b0) := !b0;
  next(b1) := (!b0 & b1) | (b0 & !b1);
  next(b2) := ((b0 & b1) & !b2) | (!(b0 & b1) & b2);
DEFINE
  out := toint(b0) + 2*toint(b1) + 4*toint(b2);
```

## Example: modulo 8 counter [2/2]

Initializing command: `go_bmc`

Picking initial state command: `bmc_pick_state`

Simulating command: `bmc_simulate`

```
NuSMV > bmc_simulate -k 3 -p
```

```
-> State: 1.1 <-
```

```
  b0 = FALSE
```

```
  b1 = FALSE
```

```
  b2 = FALSE
```

```
  out = 0
```

```
-> State: 1.2 <-
```

```
  b0 = TRUE
```

```
  out = 1
```

```
-> State: 1.3 <-
```

```
  b0 = FALSE
```

```
  b1 = TRUE
```

```
  out = 2
```

```
-> State: 1.4 <-
```

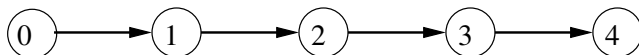
```
  b0 = TRUE
```

```
  out = 3
```

# Checking LTL specifications

The following specification is false:

LTLSPEC  $G (out = 3 \rightarrow X out = 5)$



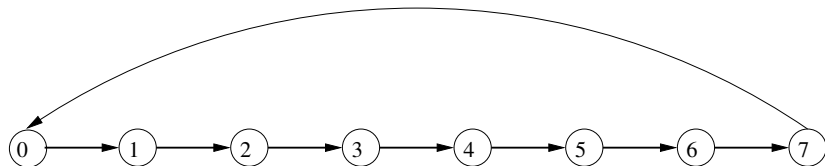
- It is an example of *safety* property ( $\rightarrow$  “nothing bad ever happens”)
  - the counterexample is a *finite* trace (of length 4)
  - there are no counterexamples of length up to 3
- LTL properties can be checked via the `check_ltlspec_bmc` and `check_ltlspec_bmc_onepb` commands

# Checking LTL specifications

```
NuSMV > check_ltlspec_bmc -p "G (out = 3 -> X out = 5)"
-- no counterexample found with bound 0 for specification ...
-- no counterexample found with bound 1 for specification ...
-- no counterexample found with bound 2 for specification ...
-- no counterexample found with bound 3 for specification ...
-- specification G (out = 3 -> X out = 5) is false
-- as demonstrated by the following execution sequence
-> State 1.1 <-
    ...
    out = 0
-> State 1.2 <-
    ...
-> State 1.4 <-
    ...
    out = 3
-> State 1.5 <-
    ...
    out = 4
```

# Checking LTL specifications

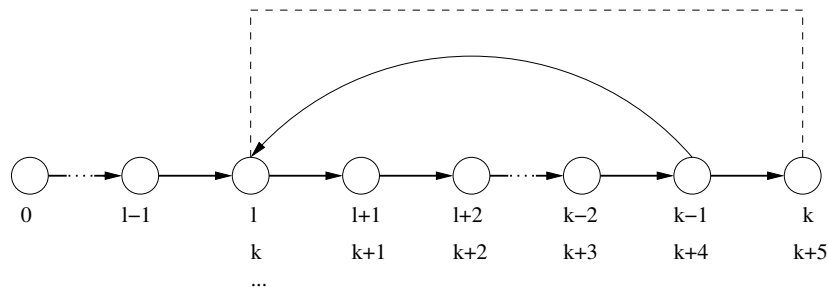
The following specification is false:

$$\text{LTLSPEC } ! G ( F ( \text{out} = 2 ) ) \quad \dashv\vdash \quad F ( G ! ( \text{out} = 2 ) )$$


- It is an example of *liveness* property ( $\rightarrow$  “something desirable will eventually happen”)
  - the counterexample is an *infinite* trace (with a *loop* of length 8)
  - since the state where `out = 2` is entered infinitely often, the property is false

# Bounded Model Checking: counterexamples

The general form of counterexamples found by BMC is the following:



- The counterexample is composed of
  - a *prefix* part (times from 0 to 1-1)
  - a *loop* part (indefinitely from 1 to k-1)
  - as the loop is always backward, it is called *loopback*



# Length and loopback condition

- `check_ltlspec_bmc` looks for counterexamples of length up to  $k$ .
- `check_ltlspec_bmc_onepb` looks for counterexamples of length  $k$ .
- To set the loopback conditions use: `-l bmc_loopback`.
  - `bmc_loopback >=0` : loop to a precise time point
  - `bmc_loopback < 0` : loop length
  - `bmc_loopback = 'X'`: no loopback
  - `bmc_loopback = '*'`: all possible loopbacks
- To set the bounded length use: `-k bmc_length`.
- Default values: `bmc_length = 10`, `bmc_loopback = '*'`
- Default values can be changed using:
  - set `bmc_length k` sets the length to  $k$
  - set `bmc_loopback l` sets the loopback to  $l$

# Checking LTL specifications

Let us consider again the specification ! G ( F (out =2))

```
NuSMV > check_ltlspec_bmc_onepb -k 9 -l 0 -p "! G ( F (out =2))"  
-- no counterexample found with bound 9 and loop at 0 for specification ...
```

# Checking LTL specifications

Let us consider again the specification ! G ( F (out =2))

```
NuSMV > check_ltlspec_bmc_onepb -k 9 -l 0 -p "! G ( F (out =2))"  
-- no counterexample found with bound 9 and loop at 0 for specification ...
```

```
NuSMV > check_ltlspec_bmc_onepb -k 8 -l 1 -p "! G ( F (out =2))"  
-- no counterexample found with bound 8 and loop at 1 for specification ...
```

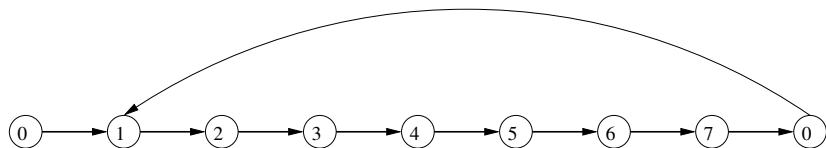
# Checking LTL specifications

Let us consider again the specification ! G ( F ( out =2))

```
NuSMV > check_ltlspec_bmc_onepb -k 9 -l 0 -p "! G ( F (out =2))"  
-- no counterexample found with bound 9 and loop at 0 for specification ...
```

```
NuSMV > check_ltlspec_bmc_onepb -k 8 -l 1 -p "! G ( F (out =2))"  
-- no counterexample found with bound 8 and loop at 1 for specification ...
```

```
NuSMV > check_ltlspec_bmc_onepb -k 9 -l 1 -p "! G ( F (out =2))"  
-- specification ! G F out = 2 is false  
-- as demonstrated by the following execution sequence  
...
```



# Checking LTL specifications

Let us consider again the specification  $\neg G ( F (out = 2))$

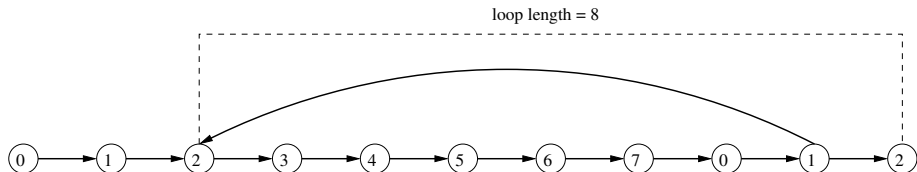
```
NuSMV > check_ltlspec_bmc_onepb -k 9 -l X -p "! G ( F (out =2))"  
-- no counterexample found with bound 9 and no loop for specification ...
```

# Checking LTL specifications

Let us consider again the specification  $\neg G ( F ( \text{out} = 2 ) )$

```
NuSMV > check_ltlspec_bmc_onepb -k 9 -l X -p "! G ( F (out =2))"  
-- no counterexample found with bound 9 and no loop for specification ...
```

```
NuSMV > check_ltlspec_bmc_onepb -k 10 -l -8 -p "! G ( F (out =2))"  
-- specification  $\neg G F \text{out} = 2$  is false  
-- as demonstrated by the following execution sequence  
...
```



# Checking invariants

- Bounded model checking can be used also for checking invariants
- Invariants are checked via the `check_invar_bmc` command
- Invariants are checked via an inductive reasoning, i.e.  
BMC tries to prove that:
  - the property holds in every initial state
  - the property holds in every state reachable from any state where it holds

# Checking invariants

- Consider the following example:

```
MODULE main
VAR
  out : 0..15;
ASSIGN
  init(out) := 0;
TRANS
  case
    out = 7 : next(out) = 0;
    TRUE    : next(out) = ((out + 1) mod 16);
  esac
INVARSPEC out in 0..10
INVARSPEC out in 0..7
```



# Checking invariants

```
NuSMV > check_invar_bmc
-- cannot prove the invariant out in (0 .. 10) : the induction fails
-- as demonstrated by the following execution sequence
-> State 1.1 <-
    out = 10
-> State 1.2 <-
    out = 11
-- invariant out in (0 .. 7)   is true
```

- The invariant `out in 0..10` is true. However, the induction fails because a state in which `out=11` can be reached from a state in which `out=10`
- If an invariant cannot be proved by inductive reasoning, it does not necessarily mean that the formula is false
- The stronger invariant `out in 0..7` is proved true by BMC, therefore also the invariant `out in 0..10` is true

# Exercise: cannibals

## Exercise:

Three missionaries and three cannibals want to cross a river but they have only one boat that holds two. If the cannibals ever outnumber the missionaries on either bank, the missionaries will be eaten. The boat cannot cross the river by itself with no people on board. The problem consists of finding a strategy to make them cross the river safely.

## Goals:

- model the problem in SMV
- use nuXmv or NuSMV to prove that there exists a solution to the planning problem

# Optional Exercise: frogs

## Optional Exercise:

The puzzle involves seven rocks and six frogs. The seven rocks are laid out in a horizontal line and the six frogs are evenly divided into a green trio and a brown trio. The green frogs sit on the rocks on the right side and the brown frogs sit on the rocks on the left side. The rock in the middle is vacant. Can you move the frogs to the opposite side? Notice that you can only move one frog at a time, and they can only move forward to an empty rock or jump over one, and only one frog to an empty rock.

## Goals:

- model the problem in SMV
- use nuXmv or NuSMV to prove that there exists a solution to the planning problem