nuXmv Bounded Model Checking

Enrico Magnago

University of Trento,
Fondazione Bruno Kessler

Bounded Model Checking

Bounded Model Checking

Idea

- look for a **counter-example** path of increasing length k
  - **bug oriented:** *is there a bad behaviour?*

Bounded Model Checking

Idea

- look for a **counter-example** path of increasing length k
  - **bug oriented:** *is there a bad behaviour?*

- for each k: build a boolean formula that is satisfiable iff there is a counter-example of length k
  (can be expressed using $k \cdot |\mathrm{s}|$ variables)

Bounded Model Checking

Idea

- look for a **counter-example** path of increasing length k
  - **bug oriented:** *is there a bad behaviour?*

- for each k: build a boolean formula that is satisfiable iff there is a counter-example of length k
  (can be expressed using $k \cdot |\mathrm{s}|$ variables)

- use of a *SAT procedure* to check the satisfiability of the boolean formula
  - can manage complex formulas on several 100K variables
  - returns satisfying assignment (i.e. a counter-example)

Commands for Bounded Model Checking

NuSMV/ nuXmv

- `go_bmc`: initializes the system for the BMC verification.
- `bmc_pick_state`, `bmc_simulate [-k]`: simulate the system
- `check_ltlspec_bmc` checks LTL specifications
- `check_invar_bmc` checks INVAR specifications

nuXmv only

- `go_msat`: initializes the system so as to use the **MathSAT 5** *SMT Solver* as back-end
- `msat_pick_state`, `msat_simulate [-k]`: simulate the system
- `msat_check_ltlspec_bmc`: checks LTL specifications
- `msat_check_invar_bmc`: checks INVAR specifications

Example: BMC simulation

modulo 8 counter

```
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);
```

```
NuSMV > read_model -i counter8.smv
NuSMV > bmc_go;
NuSMV > bmc_pick_state;
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 -> X out = 5)
```

- It is an example of **safety** property: *nothing bad ever happens*.
  - the counterexample is a **finite** trace (of length 4)
  - **important:** there are no counterexamples of length up to 3

Checking LTL specifications

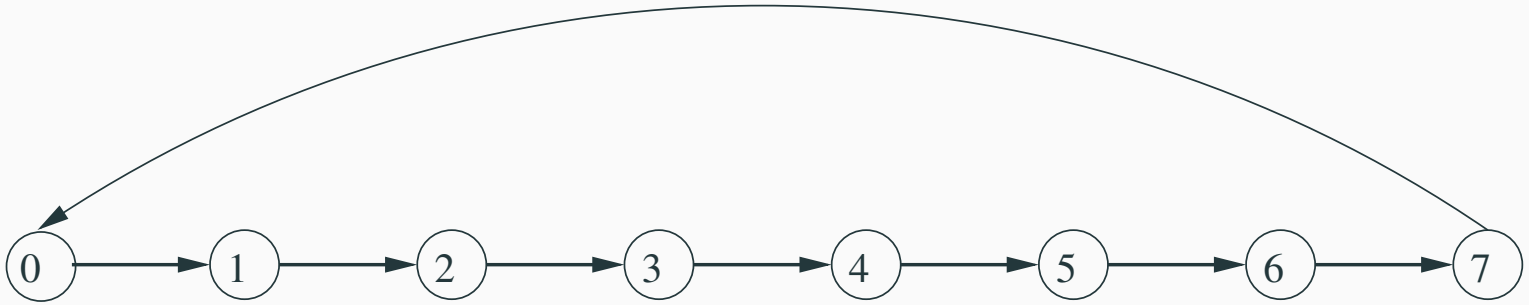
Output

```
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**:

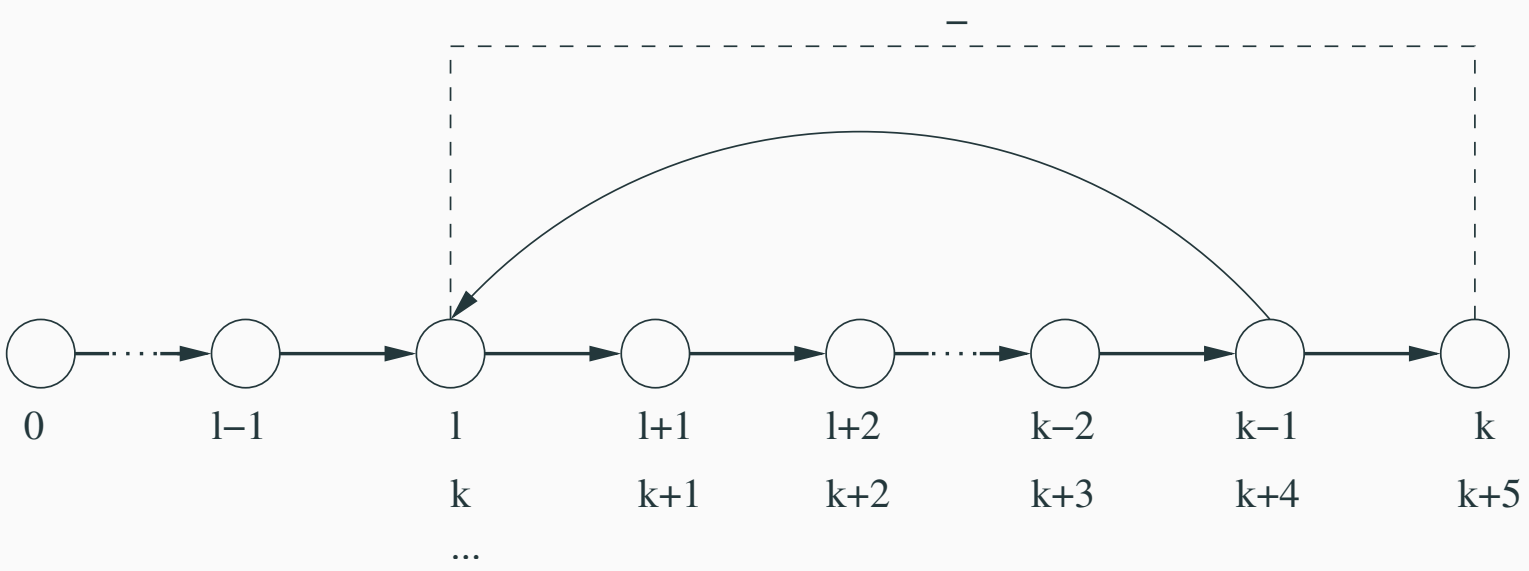
```
LTLSPEC ! G ( F (out = 2));
LTLSPEC F ( G ! (out = 2));
```



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

Bounded Model Checking: counter-examples

Looping counterexample



prefix : assignments from 0 to $l-1$,

loop : infinitely repeat assignments l to $k-1$,

loop-back : k^{th} assignment, always identical to l^{th} assignment.

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_loopback = '*'`, `bmc_length = 10`
- 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 ...
```

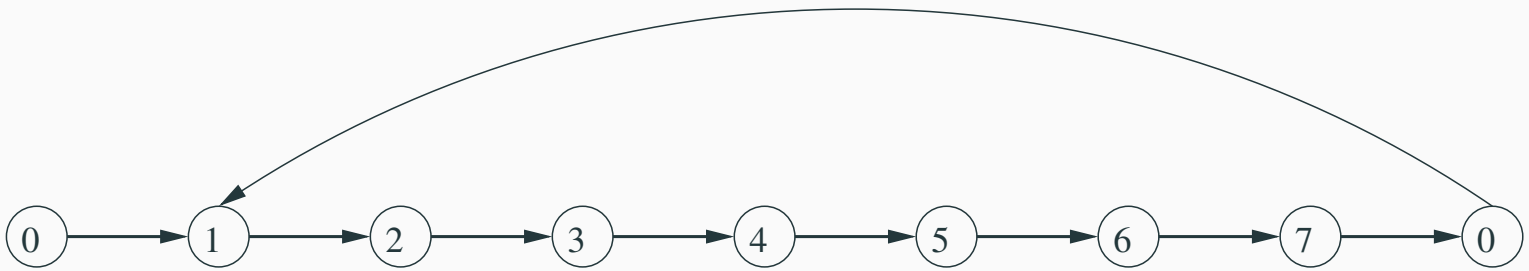
Checking LTL specifications

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 `!G ( F (out =2))`

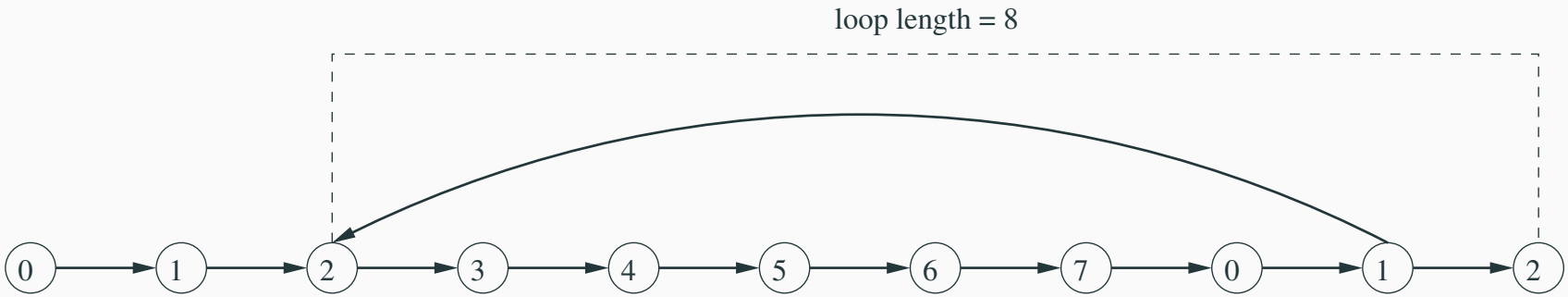
```
NuSMV > check_ltlspec_bmc_onepb -k 9 -l X -p "! G ( F (out =2))"
-- no counterexample found with bound 9 and no loop for specification
```

Checking LTL specifications

Let us consider again the specification `!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
```

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

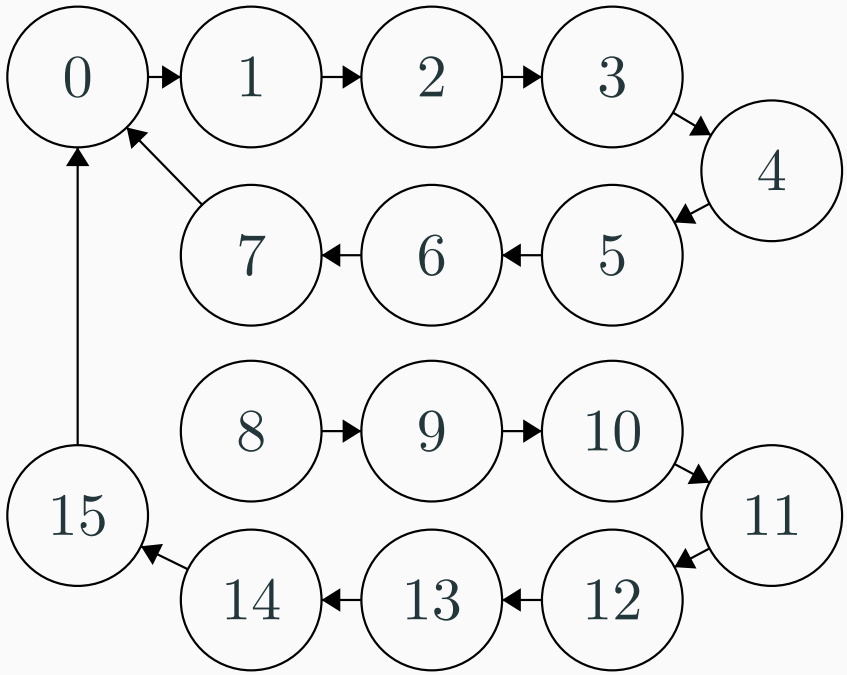


Checking invariants

- 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. nuXmv tries to prove that:
  - the property **holds in** every **initial state**
  - the property **holds in** every state that is **reachable from** another state in which the property 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**, but the the induction **fails** because a state in which `out=11` can be reached from a state in which `out=10`
- **Thus:** 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

Exercises

Exercises [1/3]

Cannibals

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.

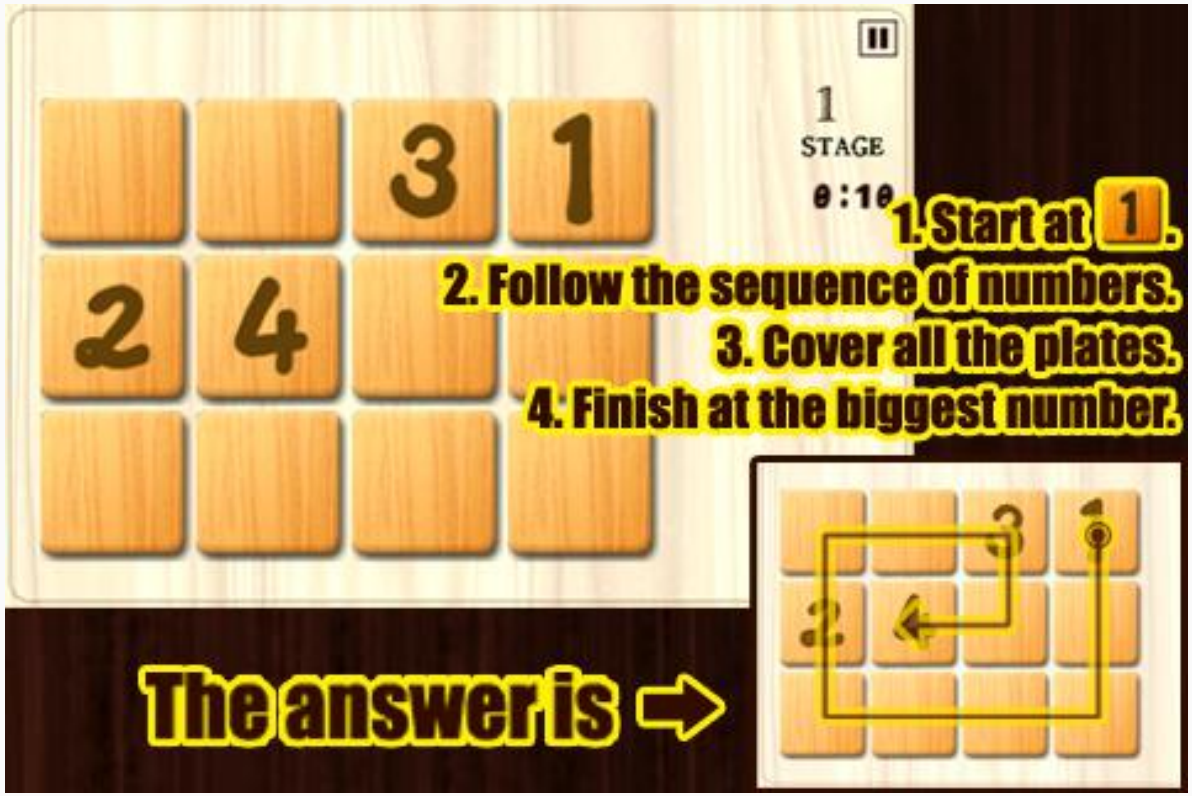
Goals

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

Exercises [2/3]

Numbers Paranoia

Encode and solve the following puzzle as a planning problem using nuXmv or NuSMV



Exercises [3/3]

Leaping frogs

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 reach an empty rock.

Goals

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