Introduction to NuSMV

Di Marco, Okwieka

Getting Started

Set up NuSMV

NuSMV has binary releases for Linux (libc6), Mac OS X and Windows

They work out of the box, just extract the archive!

Run ./bin/NuSMV -int <model (.smv)>

Set up NuSMV

We also provide a Docker container for Linux based on Alpine.

Call `./build-docker` to build the image, `./nusmv` to open the container shell.

NuSMV Model files (.smv) in ./models are bind-mounted to /models in container.

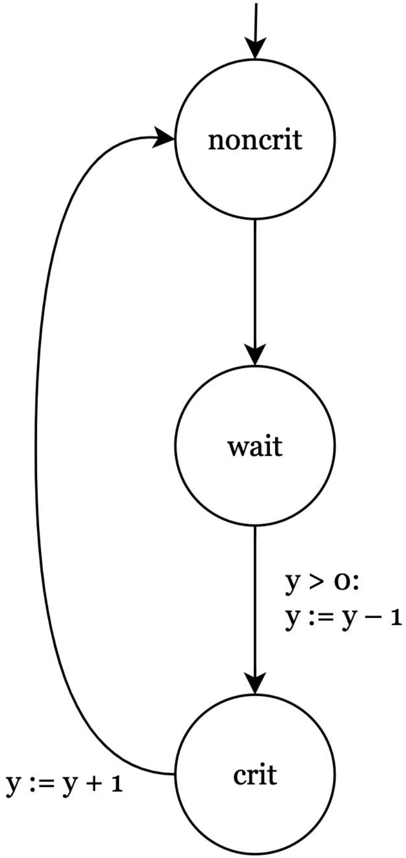
Run NuSMV

1. Start: NuSMV -int <file>

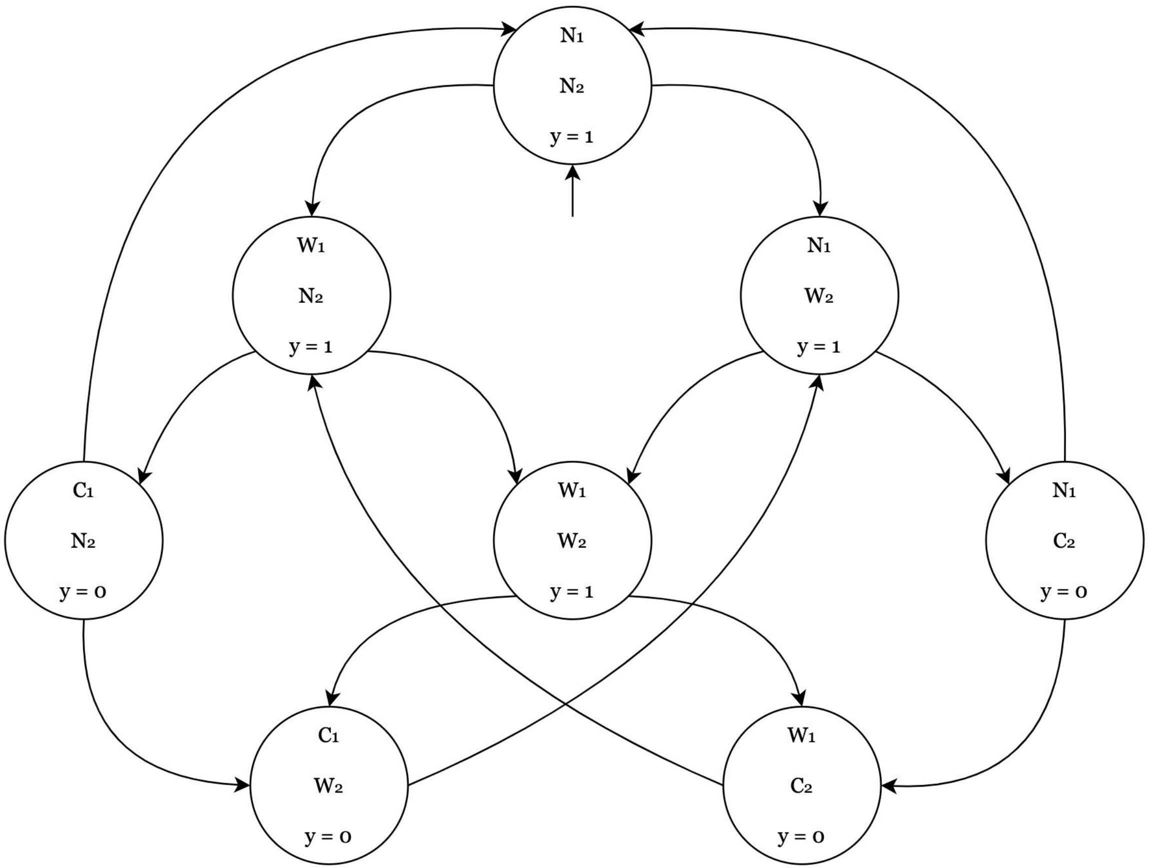
2. Build the model and BDDs: go

Modeling

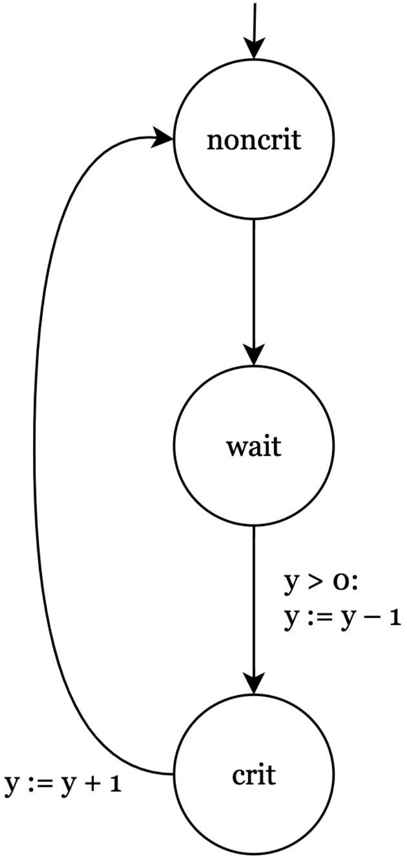
A Simple Mutual Exclusion Model



A Simple Mutual Exclusion Model

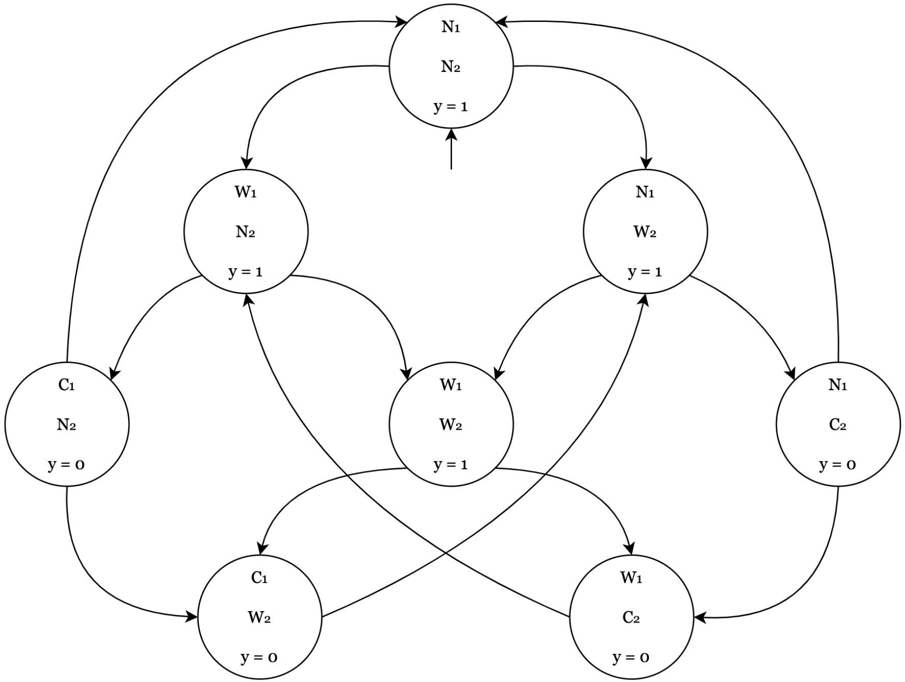


A Simple Mutual Exclusion Model



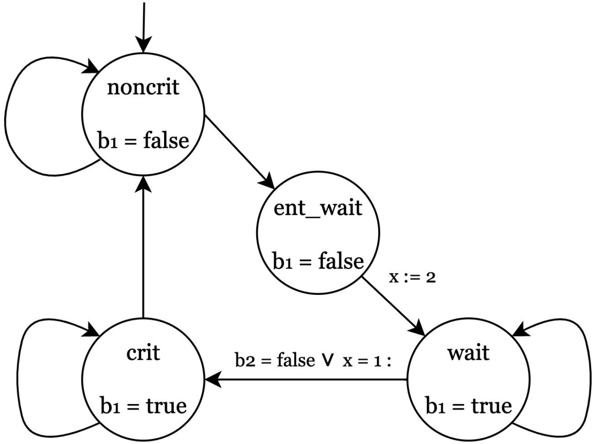

```
MODULE pg(y)
VAR
    state : { noncrit, wait, crit };
ASSIGN
    init(state) := noncrit;
    next(state) := case
            (state = noncrit)             : wait;
            (state = wait) & (y>0)        : crit;
            (state = crit)                : noncrit;
            TRUE                          : state;
        esac;
    next(y) := case
            (state = wait) & (y>0)        : y - 1;
            (state = crit) & (y<2)        : y + 1;
            TRUE                          : y;
        esac;
```

A Simple Mutual Exclusion Model

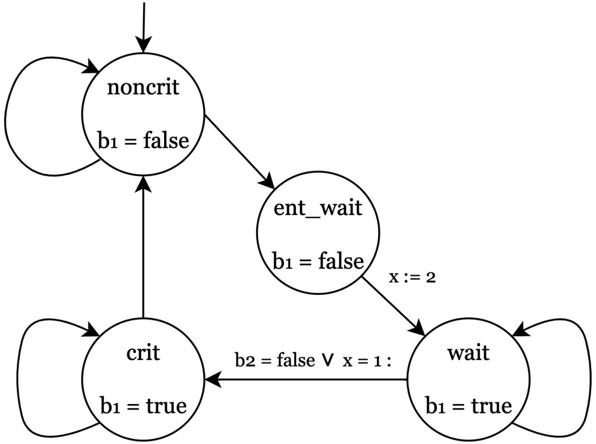



```
MODULE main
VAR
    y   : 0 .. 2;
    pg1 : process pg(y);
    pg2 : process pg(y);
ASSIGN
    init(y) := 1;
```

Peterson's Algorithm



Peterson's Algorithm




```
MODULE peterson(id, x, other_b)
VAR
    state : { noncrit, entering_wait, wait, crit };
ASSIGN
    init(state) := noncrit;
    next(state) := case
        (state = noncrit)                  : { noncrit,
                                              entering_wait };
        (state = entering_wait)            : wait;
        (state = wait)
            & ((id = x) | !(other_b))      : { wait, crit };
        (state = crit)                     : { crit, noncrit };
        TRUE                               : state;
    esac;
DEFINE
    b := (state = wait) | (state = crit);
```

Peterson's Algorithm

```
MODULE main
VAR
    x   : 1 .. 2;
    pg1 : peterson(1, x, pg2.b);
    pg2 : peterson(2, x, pg1.b);
ASSIGN
    next(x) := case
                 (pg1.state = entering_wait)  : 2;
                 (pg2.state = entering_wait)  : 1;
                 TRUE                         : x;
        esac;
```

Simulation

Simulation

Simulation generates (finite) traces, which are numbered

States of a trace also numbered and labelled

<Trace Number>.**<State Number>**

e.g. 1.**42**, 1.**43**, 1.**44**, …

Retained in memory during session

Can go to earlier state and branch off new traces

→ Tree shape

Simulation

Three different modes for choosing the next state

- Deterministic: first in set of possible states
- Random: nondeterministic
- Interactive: user chooses

What if there are too many options in interactive mode?

→ User specifies constraint formula *only for that choice*

Simulation

Overall process:

1. Pick initial state: pick_state -r OR pick_state -i
   OR continue from another trace: goto_state <state label>
2. Simulate k steps:

   simulate -k <k> -r OR -i

3. Display trace fragment:

   show_traces <trace number>[.from:to]

Both pick_state and simulate take the -c “constraint” option.

Simulation

(CLI Example of Tic-Tac-Toe Model)

Temporal Logic

CTL in NuSMV

NuSMV	English
`AG p`	Always Globally *p*
`EG p`	Exists Globally *p*
`AF p`	Always Future *p*
`EF p`	Exists Future *p*
`AX p`	Always Next *p*
`EX p`	Exists Next *p*
`A [p U q]`	Always *p* Until *q*
`E [p U q]`	Exists *p* Until *q*

LTL in NuSMV

NuSMV	English
`G p`	Globally *p*
`F p`	Future *p*
`X p`	Next *p*
`p U q`	*p* Until *q*

Past Temporal Operators

NuSMV	English	Definition	Past version of...
`H p`	Historically *p*	*p* holds always in <u>all</u> past states	Globally (**`G`**)
`O p`	Once *p*	*p* holds in <u>at least one</u> state in the past	Future (**`F`**)
`Y p`	Yesterday *p*	*p* holds in the previous state	Next (**`X`**)
`p S q`	*p* Since *q*	*p* holds up to some states in the past and then `H q` holds.	Until (**`U`**)

Model Checking

Safety properties

"it is never the case that both processes are in the critical section at the same time"

$$\mathbf{G}\,\neg(crit_1 \wedge crit_2)$$

Safety properties

"it is never the case that both processes are in the critical section at the same time"

$$\mathbf{G}\,\neg(crit_1 \wedge crit_2)$$

```
LTLSPEC G !(pg1.state = crit & pg2.state = crit)
```

Safety properties

"it is never the case that both processes are in the critical section at the same time"

$$\mathbf{G}\,\neg(crit_1 \wedge crit_2)$$

```
LTLSPEC G !(pg1.state = crit & pg2.state = crit)
```

```
> read_model -i peterson.smv
> go
> process_model
```

Safety properties

"it is never the case that both processes are in the critical section at the same time"

$$\mathbf{G}\, \neg(crit_1 \wedge crit_2)$$

```
LTLSPEC G !(pg1.state = crit & pg2.state = crit)
```

```
> read_model -i peterson.smv
> go
> process_model
```

```
-- specification G !(pg1.state = crit & pg2.state = crit) is
true
```

Liveness properties

"Both processes get into the critical section infinitely often"

$$\mathbf{GF}\, crit_1 \land \mathbf{GF}\, crit_2$$

```
LTLSPEC G ( F (pg1.state = crit)) & G ( F (pg2.state = crit))
```

Counterexamples

```
-- specification (G (F pg1.state = crit) & G (F pg2.state = crit)) is
false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
  -- Loop starts here
  -> State: 1.1 <-
    x = 1
    pg1.state = noncrit
    pg2.state = noncrit
    pg1.b = FALSE
    pg2.b = FALSE
  -> State: 1.2 <-
```

Fairness

```
FAIRNESS
    pg1.state = crit
FAIRNESS
    pg2.state = crit
```

Fairness

```
FAIRNESS
    pg1.state = crit
FAIRNESS
    pg2.state = crit
```

```
-- specification (G (F pg1.state = crit) & G (F pg2.state = crit)) is
true
```

Fairness

```
JUSTICE
    pg1.state = crit
JUSTICE
    pg2.state = crit
```

```
COMPASSION
    ( pg1.state = wait, pg1.state = crit )
COMPASSION
    ( pg2.state = wait, pg2.state = crit )
```

Past properties

"If a process reaches the critical state then it must have requested to enter."

$$\mathbf{G}\,(crit_1 \rightarrow \mathbf{O}\ wait_1)$$

```
LTLSPEC G (pg1.state=crit -> O (pg1.state=wait)) & G (pg2.state=crit -> O
(pg2.state=wait))
```

Past properties

"If a process reaches the critical state then it must have requested to enter."

$$\mathbf{G}\,(crit_1 \rightarrow \mathbf{O}\ wait_1)$$

```
LTLSPEC G (pg1.state=crit -> O (pg1.state=wait)) & G (pg2.state=crit -> O
(pg2.state=wait))
```

```
-- specification G (pg1.state = crit -> O (pg1.state = wait)) &
G (pg2.state = crit -> O (pg2.state = wait)) is true
```

Fin.