Tutorial - NuSMV

Reference: https://nusmv.fbk.eu/NuSMV/tutorial/v25/tutorial.pdf
Reference: https://nusmv.fbk.eu/examples/examples.html

Examples

Following examples will help you understand the basic features in NuSMV and how to utilize the full spectrum of its features to perform formal verification. (Note: For windows users, NuSMV should be used as NuSMV.exe)

1. Simple Verification

Code (example-0.smv)

```
MODULE main
VAR
  request : {Tr, Fa};
  state : {ready, busy};
ASSIGN
  init(state) := ready;
  next(state) := case
                 state = ready & (request = Tr): busy;
                 TRUE : {ready,busy};
                 esac;
SPEC
  AG((request = Tr) -> AF state = busy)
```

Command Line

run: NuSMV example-0.smv
output:
-- specification (AG (AF gate1.output) & AG (AF !gate1.output)) is true

2. Asynchronous three-bit Counter

Code (example-1.smv)

```
MODULE main
VAR
  gate1 : process inverter(gate3.output);
  gate2 : process inverter(gate1.output);
  gate3 : process inverter(gate2.output);
SPEC
  (AG AF gate1.output) & (AG AF !gate1.output)

MODULE inverter(input)
VAR
  output : boolean;
ASSIGN
  init(output) := FALSE;
  next(output) := !input;
FAIRNESS
  running
```

Command Line

run: NuSMV example-1.smv
output:
-- specification (AG (AF gate1.output) & AG (AF !gate1.output)) is true

3. Synchronous three-bit Counter

Code (example-2.smv)

```
MODULE main
VAR
  bit0 : counter_cell(TRUE);
  bit1 : counter_cell(bit0.carry_out);
  bit2 : counter_cell(bit1.carry_out);
SPEC
  AG AF bit2.carry_out

MODULE counter_cell(carry_in)
VAR
  value : boolean;
ASSIGN
  init(value) := FALSE;
  next(value) := value xor carry_in;
DEFINE
  carry_out := value & carry_in;
```

Command Line

run: NuSMV example-2.smv
output:
-- specification AG (AF bit2.carry_out) is true

4. Semaphore Example - Using CTL

Code (example-3.smv)

```
MODULE main
	VAR
		semaphore : boolean;
		proc1 : process user(semaphore);
		proc2 : process user(semaphore);
	ASSIGN
		init(semaphore) := FALSE;
	SPEC AG ! (proc1.state = critical & proc2.state = critical)
	SPEC AG (proc1.state = entering -> AF proc1.state = critical)
MODULE user(semaphore)
	VAR
		state : {idle, entering, critical, exiting};
	ASSIGN
		init(state) := idle;
		next(state) :=
			case
				state = idle : {idle, entering};
				state = entering & !semaphore : critical;
				state = critical : {critical, exiting};
				state = exiting : idle;
				TRUE : state;
			esac;
		next(semaphore) :=
			case
				state = entering : TRUE;
				state = exiting : FALSE;
				TRUE : semaphore;
			esac;
	FAIRNESS
		running
```

Command Line

run: NuSMV example-3.smv
output:
-- specification AG (!(proc1.state = critical & proc2.state = critical))
-- is true
-- specification AG (proc1.state = entering -> AF proc1.state = critical)
-- is false
-- as demonstrated by the following execution sequence
-> State: 1.1 <-
semaphore = FALSE
proc1.state = idle

```
proc2.state = idle
-> Input: 1.2 <-
_process_selector_ = proc1
-- Loop starts here
-> State: 1.2 <-
proc1.state = entering
-> Input: 1.3 <-
_process_selector_ = proc2
-> State: 1.3 <-
proc2.state = entering
-> Input: 1.4 <-
_process_selector_ = proc2
-> State: 1.4 <-
semaphore = FALSE
proc2.state = critical
-> Input: 1.5 <-
_process_selector_ = proc1
-> State: 1.5 <-
-> Input: 1.6 <-
_process_selector_ = proc2
-> State 1.6 <-
proc2.state = exiting
-> Input: 1.7 <-
_process_selector_ = proc2
-> State 1.7 <-
semaphore = FALSE
proc2.state = idle
```

5. Semaphore Example - Using LTL

Code (example-4.smv)

```
MODULE main
	VAR
		semaphore : boolean;
		proc1 : process user(semaphore);
		proc2 : process user(semaphore);
	ASSIGN
		init(semaphore) := FALSE;
	LTLSPEC G ! (proc1.state = critical & proc2.state = critical)
	LTLSPEC G (proc1.state = entering -> F proc1.state = critical)
MODULE user(semaphore)
	VAR
		state : {idle, entering, critical, exiting};
	ASSIGN
		init(state) := idle;
		next(state) :=
			case
				state = idle : {idle, entering};
				state = entering & !semaphore : critical;
				state = critical : {critical, exiting};
				state = exiting : idle;
				TRUE : state;
			esac;
		next(semaphore) :=
			case
				state = entering : TRUE;
				state = exiting : FALSE;
				TRUE : semaphore;
			esac;
	FAIRNESS
		running
```

Command Line

run: NuSMV example-4.smv
possible output:
-- specification G !(proc1.state = critical & proc2.state = critical) is true
-- specification G (proc1.state = entering -> F proc1.state = critical) is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
 -> State: 1.1 <-

```
        semaphore = FALSE
        proc1.state = idle
        proc2.state = idle
-> Input: 1.2 <-
        _process_selector_ = proc1
        running = FALSE
        proc2.running = FALSE
        proc1.running = TRUE
-> State: 1.2 <-
-> Input: 1.3 <-
        _process_selector_ = proc2
        proc2.running = TRUE
        proc1.running = FALSE
-> State: 1.3 <-
-> Input: 1.4 <-
        _process_selector_ = proc1
        proc2.running = FALSE
        proc1.running = TRUE
-- Loop starts here
-> State: 1.4 <-
        proc1.state = entering
-> Input: 1.5 <-
        _process_selector_ = proc2
        proc2.running = TRUE
        proc1.running = FALSE
-- Loop starts here
-> State: 1.5 <-
-> Input: 1.6 <-
-- Loop starts here
-> State: 1.6 <-
-> Input: 1.7 <-
-> State: 1.7 <-
        proc2.state = entering
-> Input: 1.8 <-
-> State: 1.8 <-
        semaphore = TRUE
        proc2.state = critical
-> Input: 1.9 <-
        _process_selector_ = proc1
        proc2.running = FALSE
        proc1.running = TRUE
-> State: 1.9 <-
-> Input: 1.10 <-
```

```
        _process_selector_ = proc2
        proc2.running = TRUE
        proc1.running = FALSE
-> State: 1.10 <-
-> Input: 1.11 <-
-> State: 1.11 <-
        proc2.state = exiting
-> Input: 1.12 <-
-> State: 1.12 <-
        semaphore = FALSE
        proc2.state = idle
```

6. Mutex Example

Code (example-5.smv)

```
MODULE main
      VAR
            s0: {noncritical, trying, critical};
            s1: {noncritical, trying, critical};
            turn: boolean;
            pr0: process prc(s0, s1, turn, FALSE);
            pr1: process prc(s1, s0, turn, TRUE);
      ASSIGN
            init(turn) := FALSE;
      FAIRNESS
            !(s0 = critical)
      --FAIRNESS
            --!(s1 = critical)
      SPEC EF((s0 = critical) & (s1 = critical))
      SPEC AG((s0 = trying) -> AF (s0 = critical))
      SPEC AG((s1 = trying) -> AF (s1 = critical))
      SPEC AG((s0 = critical) -> A[(s0 = critical) U (!(s0 =
      critical) & A[!(s0 = critical) U (s1 = critical)])])
      SPEC AG((s1 = critical) -> A[(s1 = critical) U (!(s1 =
      critical) & A[!(s1 = critical) U (s0 = critical)])])

MODULE prc(state0, state1, turn, turn0)
      ASSIGN
            init(state0) := noncritical;
            next(state0) :=
            case
            (state0 = noncritical) : {trying,noncritical};
            (state0 = trying) & (state1 = noncritical): critical;
            (state0 = trying) &
            (state1 = trying) & (turn = turn0):  critical;
            (state0 = critical) : {critical,noncritical};
            TRUE : state0;
            esac;
            next(turn) :=
            case
            turn = turn0 & state0 = critical: !turn;
            TRUE : turn;
            esac;
FAIRNESS
running
```

Command Line

run: NuSMV example-5.smv
possible output:

```
-- specification EF (s0 = critical & s1 = critical)  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
  -> State: 1.1 <-
        s0 = noncritical
        s1 = noncritical
        turn = FALSE
-- specification AG (s0 = trying -> AF s0 = critical)  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
  -> State: 2.1 <-
        s0 = noncritical
        s1 = noncritical
        turn = FALSE
  -> Input: 2.2 <-
        _process_selector_ = pr1
        running = FALSE
        pr1.running = TRUE
        pr0.running = FALSE
  -> State: 2.2 <-
        s1 = trying
  -> Input: 2.3 <-
  -> State: 2.3 <-
        s1 = critical
  -> Input: 2.4 <-
        _process_selector_ = pr0
        pr1.running = FALSE
        pr0.running = TRUE
  -- Loop starts here
  -> State: 2.4 <-
        s0 = trying
  -> Input: 2.5 <-
        _process_selector_ = pr1
        pr1.running = TRUE
        pr0.running = FALSE
  -- Loop starts here
```

```
  -> State: 2.5 <-
  -> Input: 2.6 <-
        _process_selector_ = pr0
        pr1.running = FALSE
        pr0.running = TRUE
  -- Loop starts here
  -> State: 2.6 <-
  -> Input: 2.7 <-
        _process_selector_ = main
        running = TRUE
        pr0.running = FALSE
  -> State: 2.7 <-
-- specification AG (s1 = trying -> AF s1 = critical)  is true
-- specification AG (s0 = critical -> A [ s0 = critical U (!(s0 = critical) & A [ !(s0 =
critical) U s1 = critical ] ) ] )  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
  -> State: 3.1 <-
        s0 = noncritical
        s1 = noncritical
        turn = FALSE
  -> Input: 3.2 <-
        _process_selector_ = pr0
        running = FALSE
        pr1.running = FALSE
        pr0.running = TRUE
  -> State: 3.2 <-
        s0 = trying
  -> Input: 3.3 <-
  -> State: 3.3 <-
        s0 = critical
  -> Input: 3.4 <-
  -> State: 3.4 <-
        s0 = noncritical
        turn = TRUE
-- specification AG (s1 = critical -> A [ s1 = critical U (!(s1 = critical) & A [ !(s1 =
critical) U s0 = critical ] ) ] )  is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
  -- Loop starts here
  -> State: 4.1 <-
```

```
        s0 = noncritical
        s1 = noncritical
        turn = FALSE
-> Input: 4.2 <-
        _process_selector_ = pr1
        running = FALSE
        pr1.running = TRUE
        pr0.running = FALSE
-> State: 4.2 <-
        s1 = trying
-> Input: 4.3 <-
-> State: 4.3 <-
        s1 = critical
-> Input: 4.4 <-
-> State: 4.4 <-
        s1 = noncritical
```