MODEL 1: A simple counter

MODULE Proc

VAR

  n : {1, 2, 3, 4};

ASSIGN

  init(n) := 1;

  next(n) := case

               n=4: 1;

               n<4: n+1;

               TRUE: n;

             esac;

MODULE main

VAR

  p : Proc;

The specifications found after running the above code in NuSmv can be found below:

Number of Input Variables: 0

Number of State Variables: 1

1: {1, 2, 3, 4}

Number of Frozen Variables: 0

p.n : {1, 2, 3, 4}

Number of bits: 2 (0 frozen, 0 input, 2 state)

system diameter: 4

reachable states: 4 (2^2) out of 4 (2^2)

------- State 1 ------

p.n = 4

------- State 2 ------

p.n = 2

------- State 3 ------

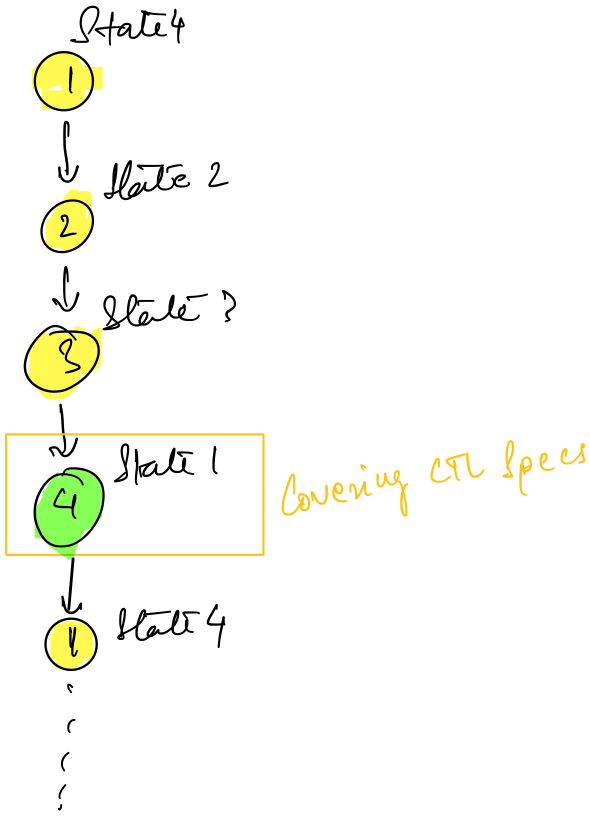
p.n = 3

------- State 4 ------

p.n = 1

-------------------------

TRANSITION DIAGRAM:



TRANSITION DIAGRAM FOR THE MODULE

The CTL specs at test are:

A screenshot of text

Description automatically generated

1. AG ! (p.n = 4)

Along all the possible paths, there can never be a state where the value of p.n = 4 holds true : This statement can be immediately falsified from the transition diagram above and the counter example is State: 2.4 <- p.n = 4.

2. AG (p.n = 4 -> p.n = 4)

Along all the possible paths there exists a state which has its property p.n = 4: This statement holds true and can be verified from the transition diagram above.

The screenshots of the procedure can be found below:

A screenshot of text

Description automatically generated

A screenshot of text

Description automatically generated

A screenshot of text

Description automatically generated

MODEL 2: Semaphores

MODULE user(lock)  
VAR  
 state: {idle, entering, critical, exiting};  
ASSIGN

     init(state) := idle;

     next(state) :=

           case

                state = idle : {idle, entering};

                state = entering & !lock: critical;

                state = critical : {critical, exiting};

                state = exiting : idle;

                TRUE : state;

           esac;

     next(lock) :=

           case

                state = entering : TRUE;

                state = exiting : FALSE;

                TRUE : lock;

           esac;

MODULE main

VAR

     semaphore : boolean;

     proc1 : process user(semaphore);

     proc2 : process user(semaphore);

ASSIGN

     init(semaphore) := FALSE;

The specifications found after running the above code in NuSmv can be found below:

Number of Input Variables: 1

1: {main, proc2, proc1}

Number of State Variables: 3

2: {idle, entering, critical, exiting}

1: boolean

Number of Frozen Variables: 0

semaphore : boolean

proc1.state : {idle, entering, critical, exiting}

proc2.state : {idle, entering, critical, exiting}

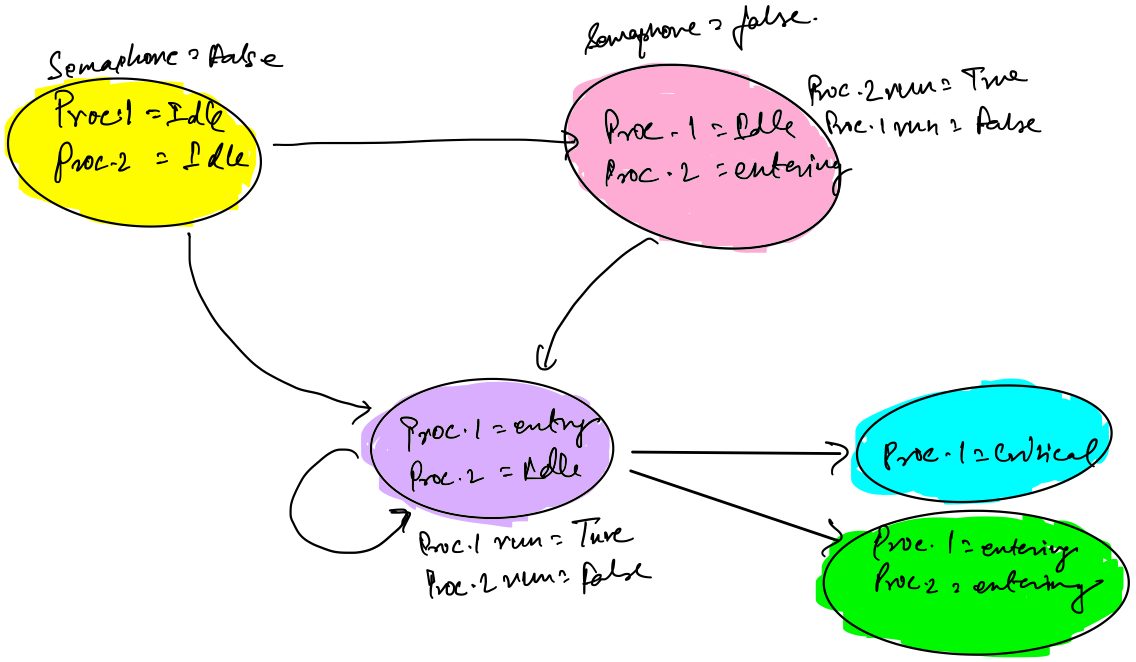
\_process\_selector\_ : {main, proc2, proc1}

Number of bits: 7 (0 frozen, 2 input, 5 state)

system diameter: 5

reachable states: 12 (2^3.58496) out of 32 (2^5)

TRANSITION DIAGRAM:



TRANSITION DIAGRAM FOR THE MODULE

The CTL specs at test are:

A screenshot of text

Description automatically generated

1. AG !(proc1.state = critical & proc2.state = critical)

Among all the possible paths, there can never be a state when both the proc1 and proc2 exist in the critical state: This statement is true as there can never be any state which holds both proc1 and proc2 together in critical state.

2. AG(proc1.state = entering -> AF (proc1.state = critical))

In all the possible paths in the transition system if the proc1.state is at the entering state, this implies that among all the paths in the future the state of proc1 will be critical following the entering state.

The screenshots of the procedure can be found below:

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of text

Description automatically generated

A screenshot of text

Description automatically generated

MODEL 3: Spin

MODULE main

VAR

     request : boolean;

     state : {ready, busy};

ASSIGN

     init(state) := ready;

     next(state) := case

           state = ready & request : busy;

           TRUE : {ready , busy};

           esac;

The specifications found after running the above code in NuSmv can be found below:

Number of Input Variables: 0

Number of State Variables: 2

1: {ready, busy}

1: boolean

Number of Frozen Variables: 0

request : boolean

state : {ready, busy}

Number of bits: 2 (0 frozen, 0 input, 2 state)

system diameter: 2

reachable states: 4 (2^2) out of 4 (2^2)

------- State 1 ------

request = TRUE

state = busy

------- State 2 ------

request = TRUE

state = ready

------- State 3 ------

request = FALSE

state = busy

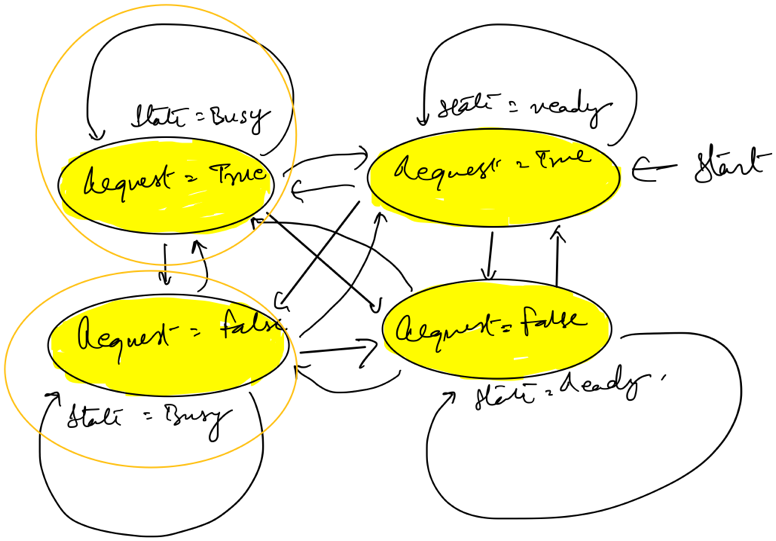
------- State 4 ------

request = FALSE

state = ready

-------------------------

Transition diagram:



Transition diagram for the module

The specs at test are:

A close up of a newspaper

Description automatically generated

1. AG (request -> AF state = busy)

Every time if there is a request, somewhere down the line from that request there will be a state which will be busy:

This statement is indeed true and same can be verified from the transition diagram above.

2. G (request -> AF state = busy)

This spec was provided as LTL spec but it was incorrect as it was a mix of LTL and CTL formulas, so further improvements in this formula was made as described below:

a.) AG (request -> AF state = busy):

Whenever there is a request, some state in the future is busy:

This statement is true and an observation of the transition diagram above provides the proof of correctness.

b.) EG (request -> AF state = busy):

There exists a state globally that whenever a request takes place, all the paths originating from that state will eventually hold busy at some time in future:

This statement is true and can be easily verified from the transition system above.

3. G (request -> F state = busy)

This spec was given incorrect as CTL spec but it is a CTL spec instead. This spec states that whenever there is a request down the path, some state in the future will be busy.

This spec holds true as a corollary of previous specs and can be verified from the transition system.

4. EG (request -> AG state = busy)

This spec states that there exists globally a state where whenever a request is present, all the states from that particular state is busy.

This CTL formula is wrong as we see from the transition and the output that state 1.1 is in request as well as ready.

The screenshots of the procedure can be found below:

A picture containing bird

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of text

Description automatically generated

A close up of a newspaper

Description automatically generated

MODEL 4: Traffic Lights

MODULE light (other)

VAR

     state: {r,y,g};

ASSIGN

     init(state) := r;

     next(state) := case

           state = r & other = r : {r, y};    -- either r or y

           state = y : g;

           state = g : {g, r};

           TRUE : state;

          esac;

MODULE main

VAR

-- tl1:light(t12.state);tl2:light(tl1.state);synch,two G possible

     tl1: process light (tl2.state); -- asynch

     tl2: process light (tl1.state);

The specifications found after running the above code in NuSmv can be found below:

Number of Input Variables: 1

1: {main, tl2, tl1}

Number of State Variables: 2

2: {r, y, g}

Number of Frozen Variables: 0

tl1.state : {r, y, g}

tl2.state : {r, y, g}

\_process\_selector\_ : {main, tl2, tl1}

Number of bits: 6 (0 frozen, 2 input, 4 state)

system diameter: 3

reachable states: 5 (2^2.32193) out of 9 (2^3.16993)

The specs at test are:

A close up of text on a white background

Description automatically generated

1.) ! F (tl1.state = g & tl2.state = g)

This spec states that both the states tl1 and tl2 can't be green together at the same time:

This statement is indeed true as there are not any two states which will be green together at same time.

2.) !F !(tl1.state = g & tl2.state = g)

This spec is a counter example of the first spec and basically states that both the states can be green together at a same time:

This statement being the counter statement of first can't hold true.

The screenshots of the procedure can be found below:

A close up of a newspaper

Description automatically generated

A screenshot of text

Description automatically generatedA screenshot of a cell phone

Description automatically generatedA close up of text on a white background

Description automatically generated