* **Code**

MODULE main

VAR

proc1 : process task(x);

proc2 : process task(x);

x : 0 .. 200;

ASSIGN

init(x) := 0;

---EF p = exists some path (E) that eventually in the future satisfies p.

---EG p = exists some path (E) that condition p is continuously true.

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

SPEC EG (x = 0)

SPEC EG (x = 1)

SPEC EG (x = 2)

SPEC EF (x = 0 & proc1.counter = 100 & proc2.counter = 100 )

SPEC EF (x = 1 & proc1.counter = 100 & proc2.counter = 100 )

SPEC EF (x = 2 & proc1.counter = 100 & proc2.counter = 100 )

SPEC EF (x = 100 & proc1.counter = 100 | proc2.counter = 100 )

SPEC EF (x = 200 & proc1.counter = 100 & proc2.counter = 100 )

SPEC EF (x = 201)

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

MODULE task(x)

VAR

counter : 0 .. 100;

ASSIGN

init(counter) := 0;

next(x) :=

case

(counter < 100) & (x < 200) : x+1;

TRUE : x;

esac;

next(counter) :=

case

(counter < 100) : counter + 1;

TRUE : counter;

esac;

* **Command**

NuSMV.exe hw2.smv

* **Results**

Result shows that in ***smv*** implementation of the problem, there is a chance of starvation but there is no chance of accessing the shared variable at the same time.

-- specification EG x = 0 is true

None of processes start running

-- specification EG x = 1 is false

-- specification EG x = 2 is false

-- specification EF ((x = 0 & proc1.counter = 100) & proc2.counter = 100) is false

-- specification EF ((x = 1 & proc1.counter = 100) & proc2.counter = 100) is false

-- specification EF ((x = 2 & proc1.counter = 100) & proc2.counter = 100) is false

The above lines show that mutual exclusion condition is always preserved.

-- specification EF ((x = 100 & proc1.counter = 100) | proc2.counter = 100) is true

It shows that starvation condition may happen.

-- specification EF ((x = 200 & proc1.counter = 100) & proc2.counter = 100) is true

It shows that the ideal condition of processing can be achieved.

-- specification EF x = 201 is false