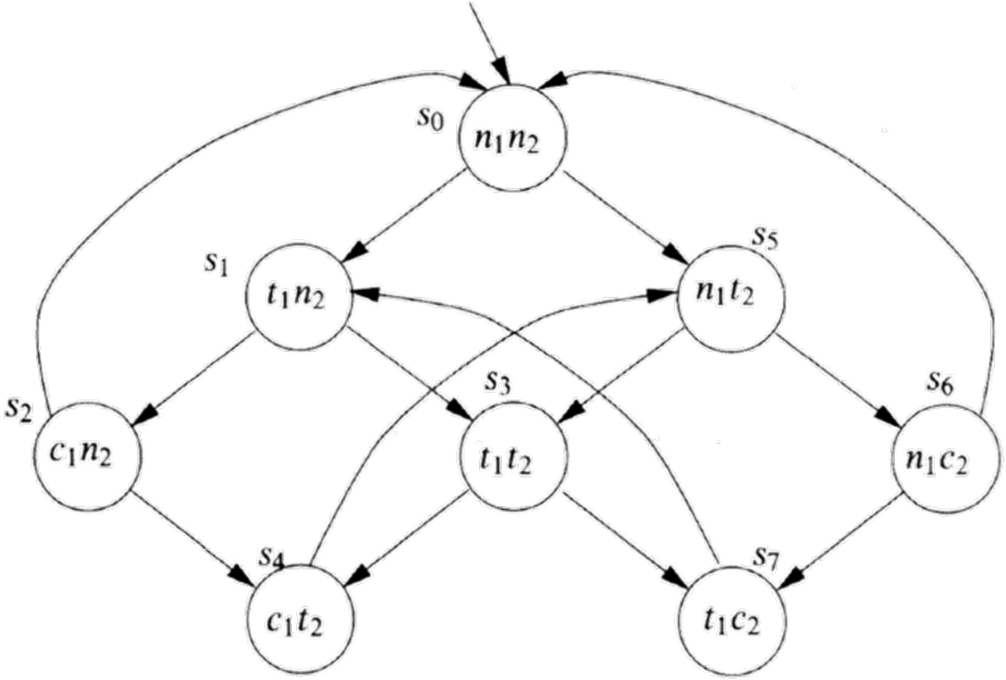
Temporal Logic
-NuSMV

Moonzoo Kim
CS Dept. KAIST

moonzoo@cs.kaist.ac.kr

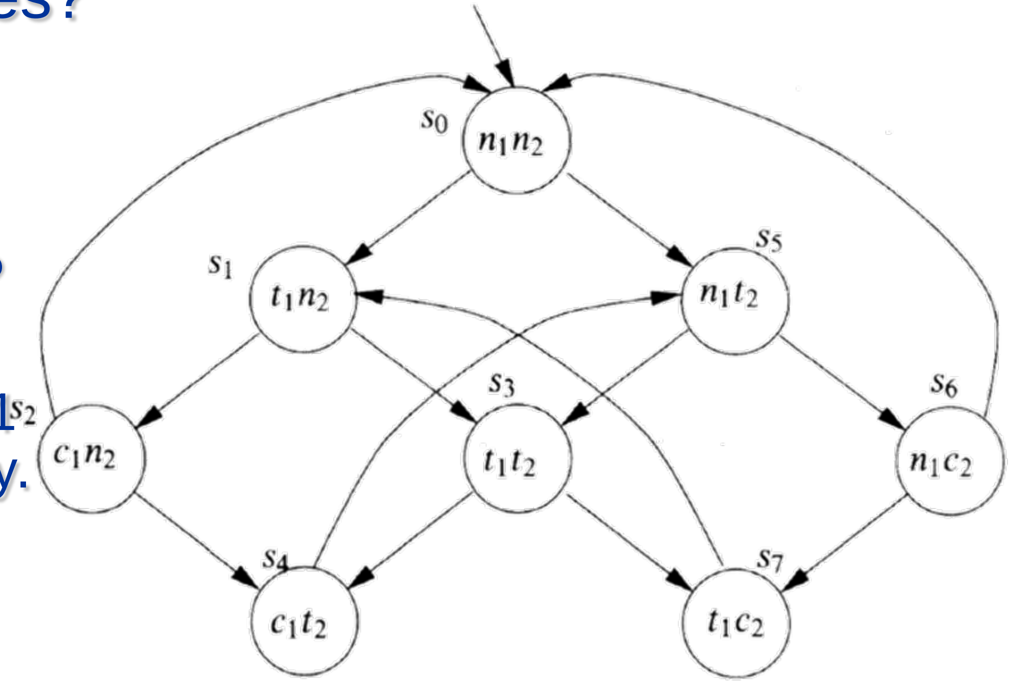
NuSMV specification of the 1st mutual exclusion (1/2)

```
MODULE main
VAR
  state : {nn,tn,cn,tt,ct,nt,nc,tc,cc} ;
ASSIGN
  init(state) := nn;
  next(state) :=
     case
         (state=nn)   : {tn, nt};
         (state=tn)   : {cn,tt};
         (state=cn)   : {nn,ct};
         (state=tt)   : {ct,tc};
         (state=ct)   : nt;
         (state=nt)   : {tt,nc};
         (state=nc)   : {nn,tc};
         (state=tc)   : tn;
           1 : state;
     esac;
-- safety
LTLSPEC G !(state=cc)
-- liveness
LTLSPEC G ((state=tn|state=tt|state=tc) -> F (state=cn|state=ct|state=cc))
```



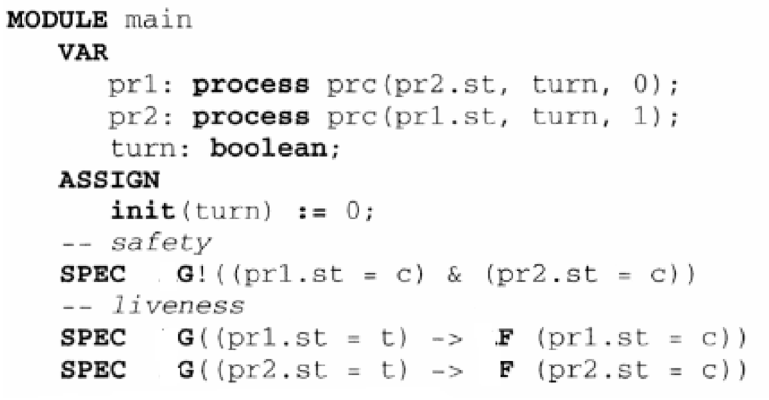
NuSMV specification of the 1st mutual exclusion (2/2)

- What if there are 3 processes?
  - We need a more realistic compositional model
- Does this way of modeling reflect real implementation?
  - There might be no global scheduler, which allows only 1 process to execute 1 step only.
  - For software process, asynchronous interleaving is more realistic

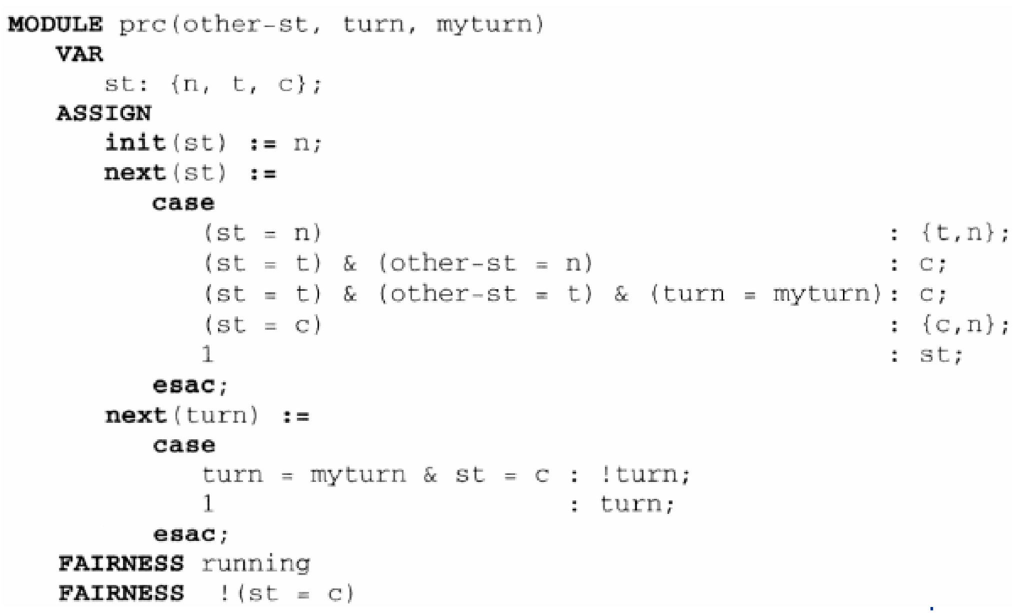


Revised mutual exclusion model in NuSMV (1/2)

- This code consists of two modules, `main` and `prc`
  - `main`
    - `turn` determines whose turn it is to enter the critical section if both are trying to enter
  - `prc`
    - `st`: the status of a process
    - `other-st`: the status of the other
- FAIRNESS ϕ restrict search tree to execution paths along which ϕ is infinitely often true
  - i.e., limit the search space
  - `FAIRNESS running` enforces that the process should execute infinitely often



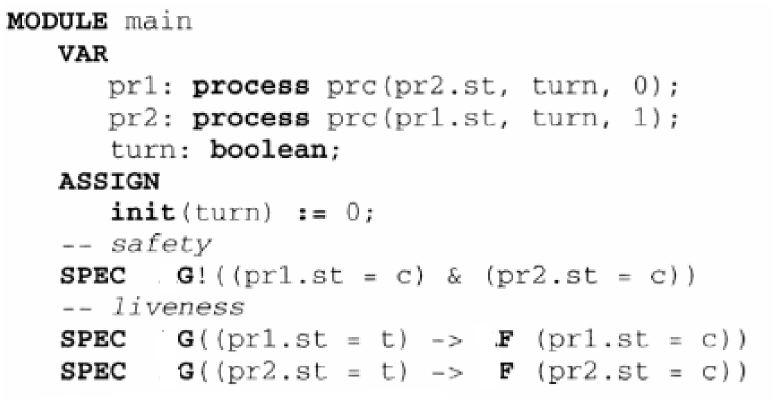
```
MODULE main
   VAR
      pr1: process prc(pr2.st, turn, 0);
      pr2: process prc(pr1.st, turn, 1);
      turn: boolean;
   ASSIGN
      init(turn) := 0;
   -- safety
   SPEC   G!((pr1.st = c) & (pr2.st = c))
   -- liveness
   SPEC   G((pr1.st = t) -> F (pr1.st = c))
   SPEC   G((pr2.st = t) -> F (pr2.st = c))
```



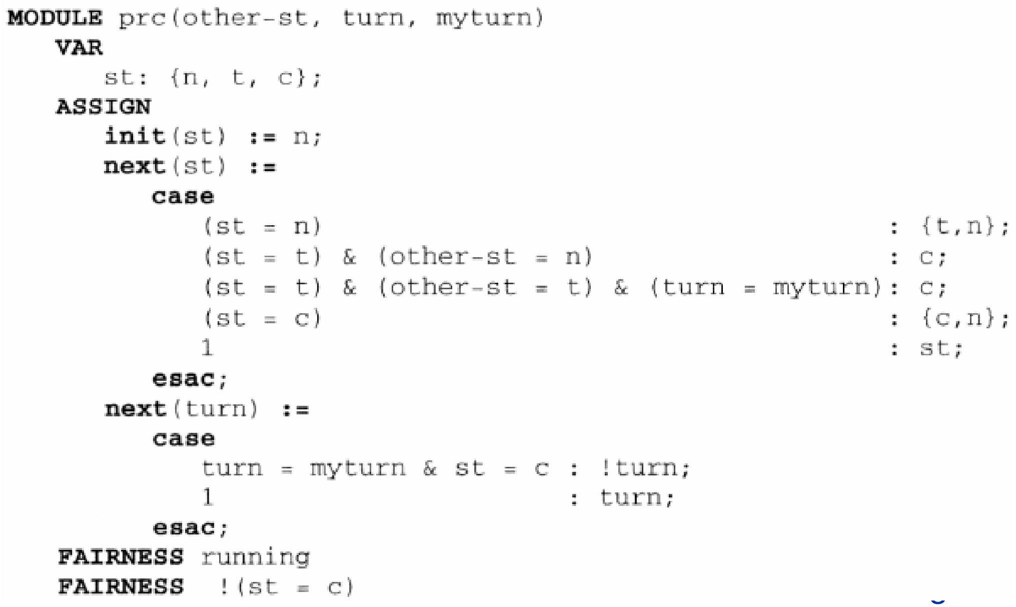
```
MODULE prc(other-st, turn, myturn)
   VAR
      st: {n, t, c};
   ASSIGN
      init(st) := n;
      next(st) :=
         case
            (st = n)                                      : {t,n};
            (st = t) & (other-st = n)                     : c;
            (st = t) & (other-st = t) & (turn = myturn): c;
            (st = c)                                      : {c,n};
            1                                             : st;
         esac;
      next(turn) :=
         case
            turn = myturn & st = c  : !turn;
            1                       : turn;
         esac;
   FAIRNESS running
   FAIRNESS  !(st = c)
```

Revised mutual exclusion model in NuSMV (2/2)

- `FAIRNESS !(st=c)`
  - This prevents a process from staying at critical section forever
    - this prevents to detects silly violation of liveness property due to such situation
- `FAIRNESS running`
  - This prevents a process from executing all the time
    - this prevents to detects silly violation of liveness property due to such situation

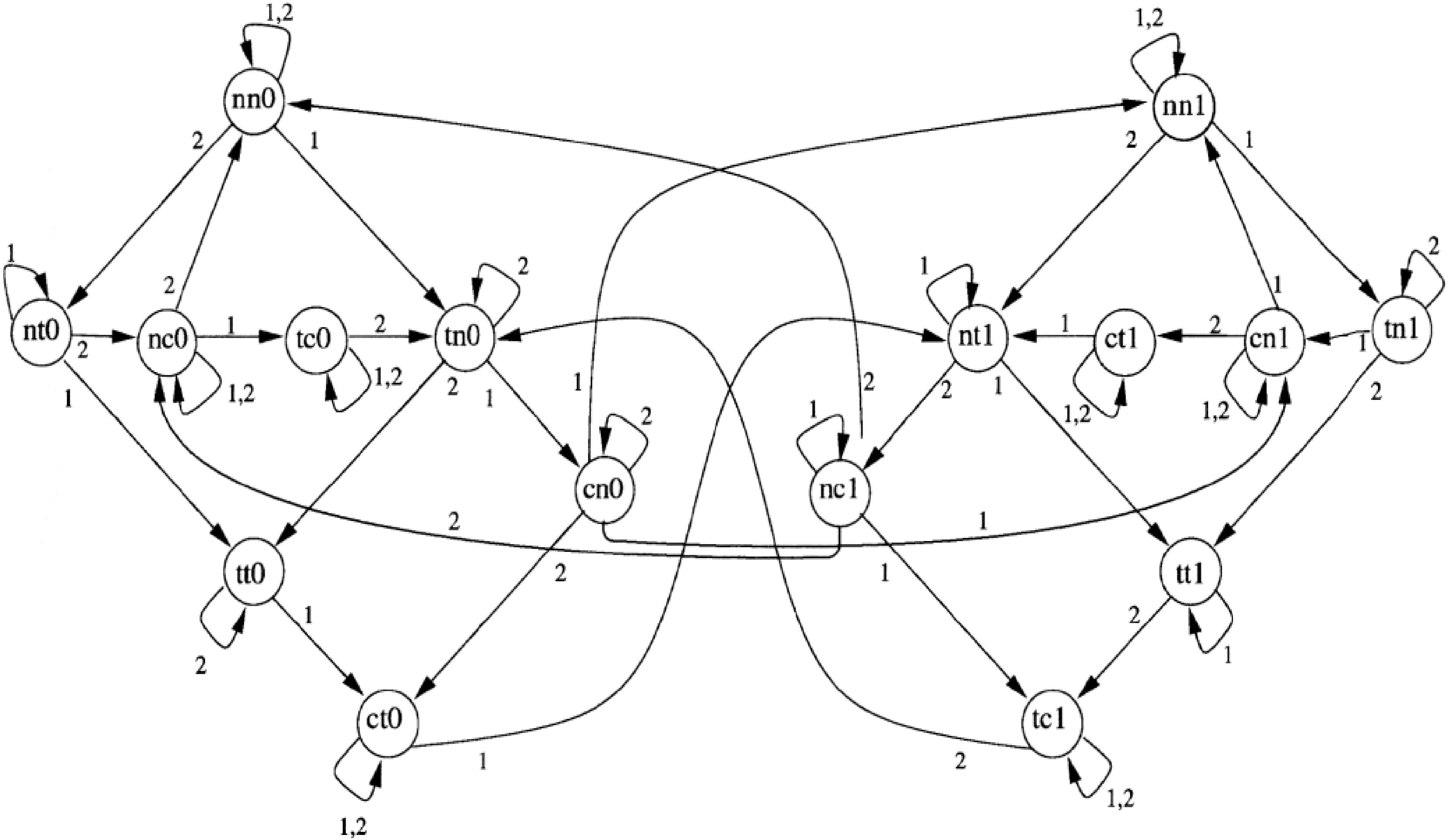


```
MODULE main
   VAR
      pr1: process prc(pr2.st, turn, 0);
      pr2: process prc(pr1.st, turn, 1);
      turn: boolean;
   ASSIGN
      init(turn) := 0;
   -- safety
   SPEC   G!((pr1.st = c) & (pr2.st = c))
   -- liveness
   SPEC   G((pr1.st = t) ->  F (pr1.st = c))
   SPEC   G((pr2.st = t) ->  F (pr2.st = c))
```



```
MODULE prc(other-st, turn, myturn)
   VAR
      st: {n, t, c};
   ASSIGN
      init(st) := n;
      next(st) :=
         case
            (st = n)                                        : {t,n};
            (st = t) & (other-st = n)                       : c;
            (st = t) & (other-st = t) & (turn = myturn): c;
            (st = c)                                        : {c,n};
            1                                               : st;
         esac;
      next(turn) :=
         case
            turn = myturn & st = c : !turn;
            1                      : turn;
         esac;
   FAIRNESS running
   FAIRNESS  !(st = c)
```

Transition system


nn0
nt0
nc0
tc0
tn0
cn0
tt0
ct0
nn1
nt1
ct1
cn1
tn1
nc1
tt1
tc1
1,2
1
2