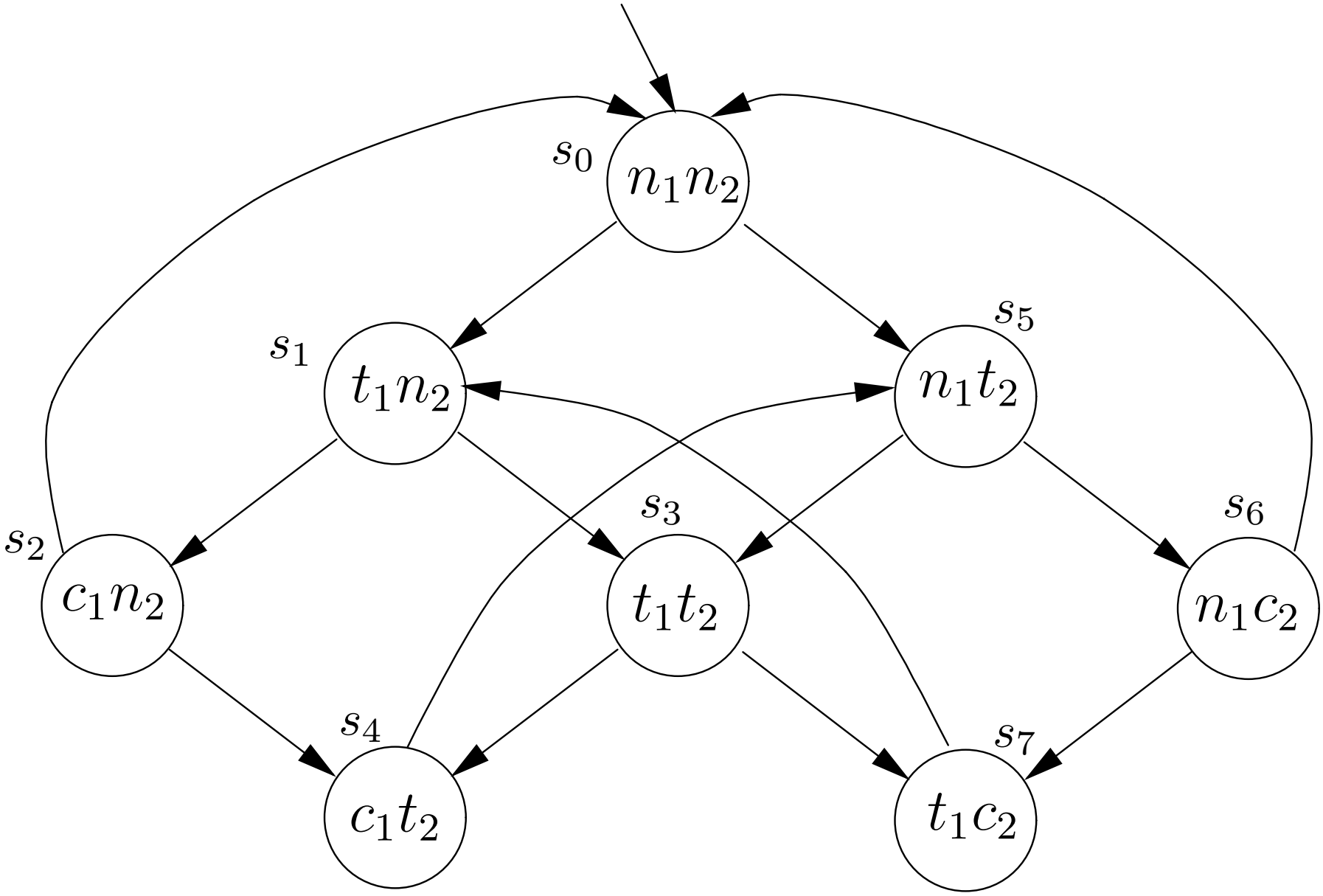


Figure 3.7. A first-attempt model for mutual exclusion.

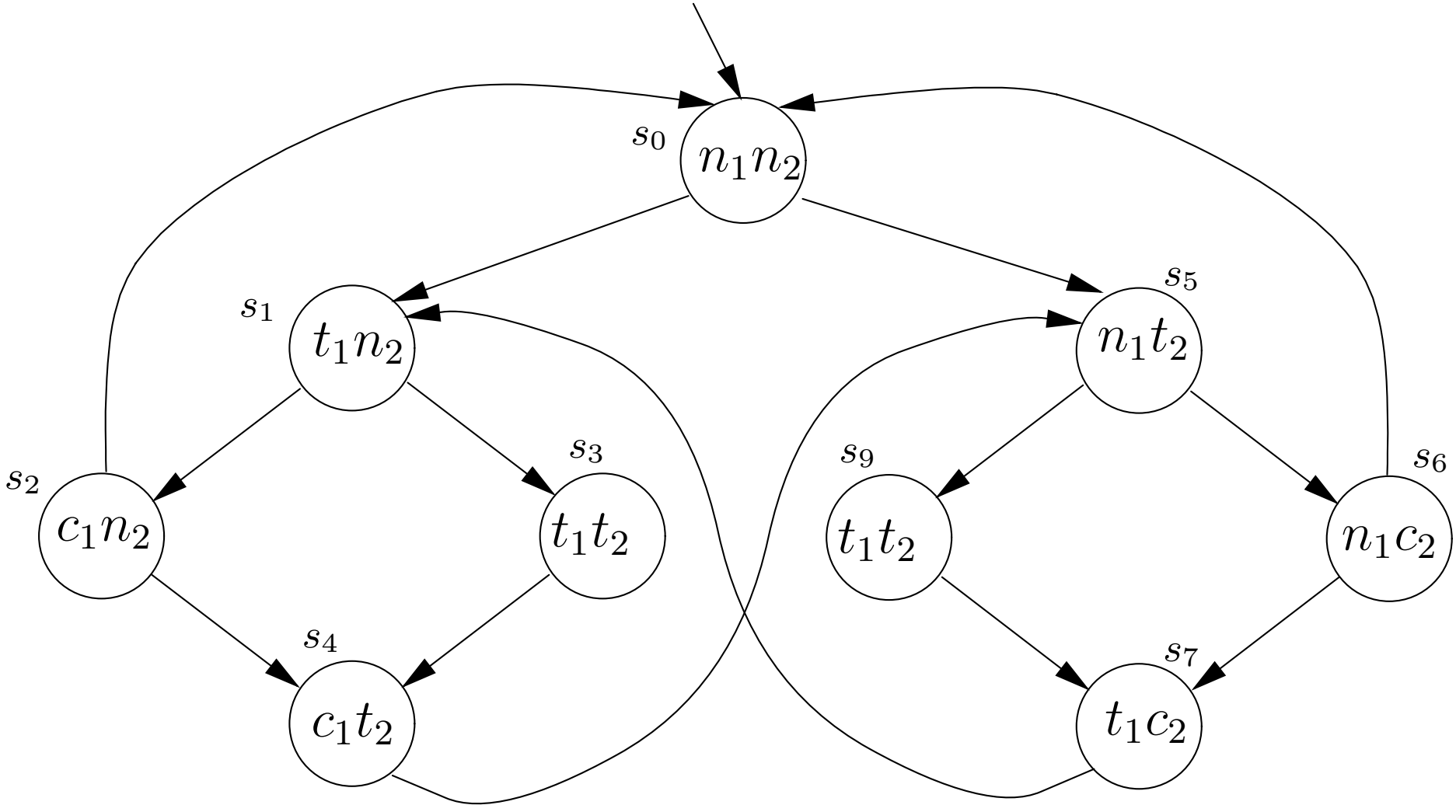
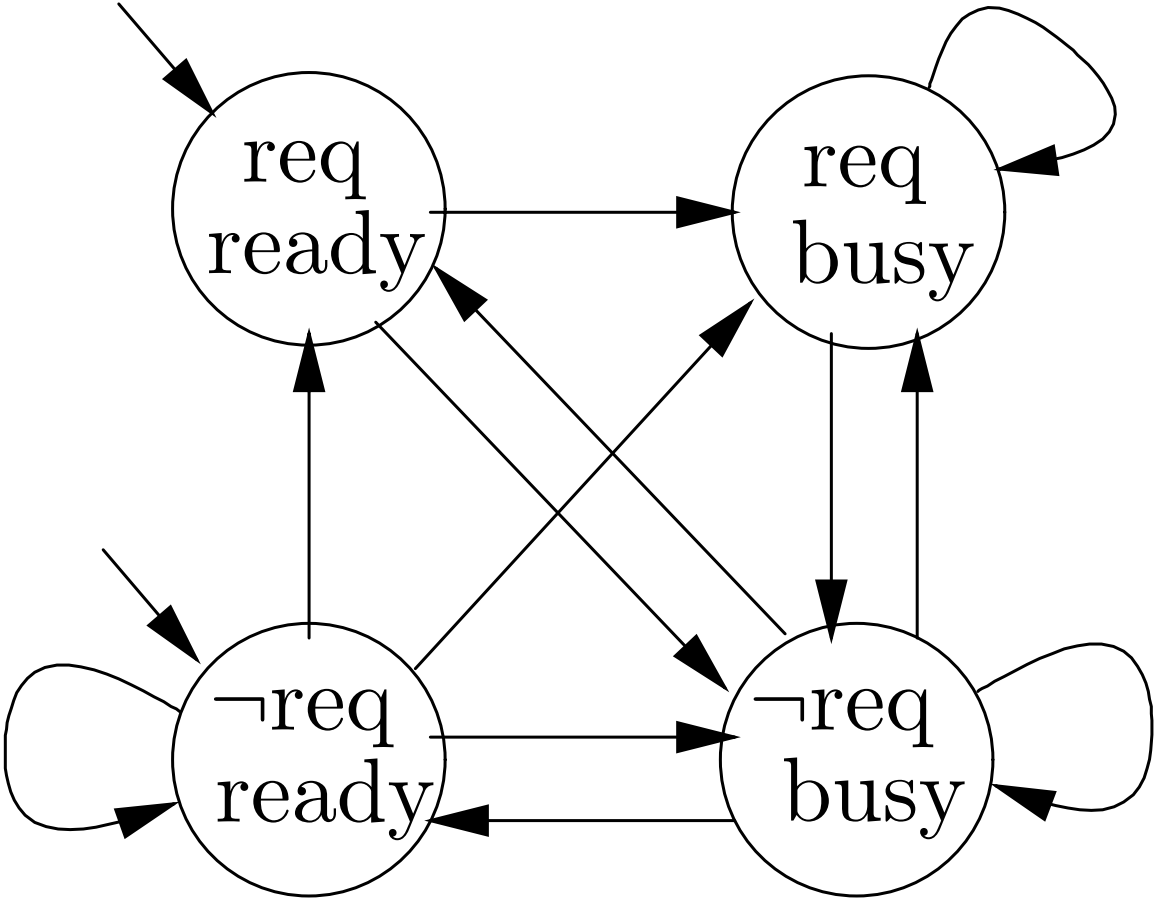


Figure 3.8. A second-attempt model for mutual exclusion. There are now two states representing t_1t_2, namely s_3 and s_9.

```
MODULE main
VAR
  request : boolean;
  status : {ready,busy};
ASSIGN
  init(status) := ready;
  next(status) := case
                    request : busy;
                    1 : {ready,busy};
                  esac;
LTLSPEC
  G(request -> F status=busy)
```


req
ready
req
busy
¬req
ready
¬req
busy

```
MODULE main
VAR
  bit0 : counter_cell(1);
  bit1 : counter_cell(bit0.carry_out);
  bit2 : counter_cell(bit1.carry_out);
LTLSPEC
  G F bit2.carry_out

MODULE counter_cell(carry_in)
VAR
  value : boolean;
ASSIGN
  init(value) := 0;
  next(value) := (value + carry_in) mod 2;
DEFINE
  carry_out := value & carry_in;
```

```
MODULE main
   VAR
      pr1: process prc(pr2.st, turn, 0);
      pr2: process prc(pr1.st, turn, 1);
      turn: boolean;
   ASSIGN
      init(turn) := 0;
   -- safety
   LTLSPEC  G!((pr1.st = c) & (pr2.st = c))
   -- liveness
   LTLSPEC  G((pr1.st = t) -> F (pr1.st = c))
   LTLSPEC  G((pr2.st = t) -> F (pr2.st = c))
   -- 'negation' of strict sequencing (desired to be false)
   LTLSPEC G(pr1.st=c -> ( G pr1.st=c | (pr1.st=c U
             (!pr1.st=c & G !pr1.st=c | ((!pr1.st=c) U 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)
```

Figure 3.10. SMV code for mutual exclusion. Because W is not supported by SMV, we had to make use of equivalence (3.3) to write the no-strict-sequencing formula as an equivalent but longer formula involving U.

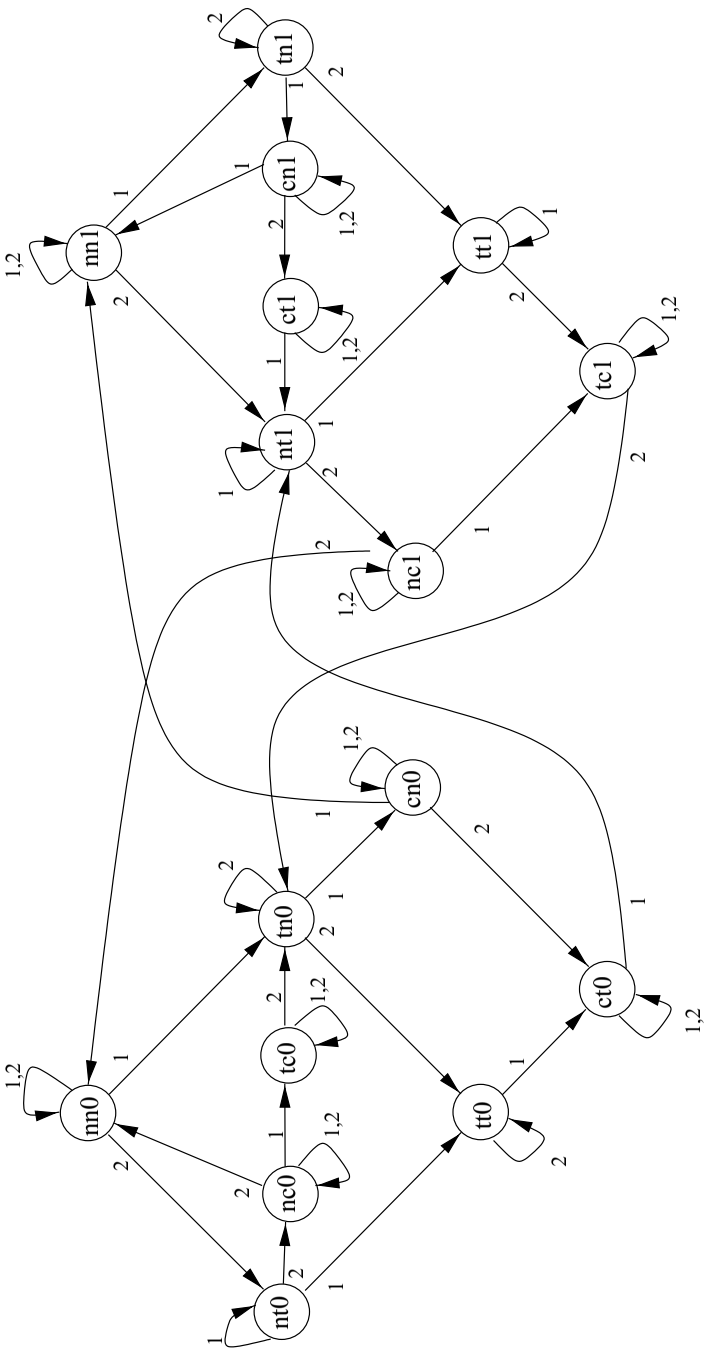


Figure 3.11. The transition system corresponding to the SMV code in Figure 3.10. The labels on the transitions denote the process which makes the move. The label $1, 2$ means that either process could make that move.

```
MODULE main
 VAR
  ferryman : boolean;
  goat     : boolean;
  cabbage  : boolean;
  wolf     : boolean;
  carry    : {g,c,w,0};
ASSIGN
 init(ferryman) := 0; init(goat)     := 0;
 init(cabbage)  := 0; init(wolf)     := 0;
 init(carry)    := 0;

 next(ferryman) := 0,1;

 next(carry) := case
                   ferryman=goat : g;
                   1             : 0;
                esac union
                case
                   ferryman=cabbage : c;
                   1                : 0;
                esac union
                case
                   ferryman=wolf : w;
                   1             : 0;
                esac union 0;

 next(goat) := case
   ferryman=goat  & next(carry)=g : next(ferryman);
   1                              : goat;
 esac;
 next(cabbage) := case
   ferryman=cabbage & next(carry)=c : next(ferryman);
   1                                : cabbage;
 esac;
 next(wolf) := case
   ferryman=wolf & next(carry)=w : next(ferryman);
   1                             : wolf;
 esac;

LTLSPEC !((   (goat=cabbage | goat=wolf) -> goat=ferryman)
            U (cabbage & goat & wolf & ferryman))
```

Figure 3.12. NuSMV code for the ferryman planning problem.