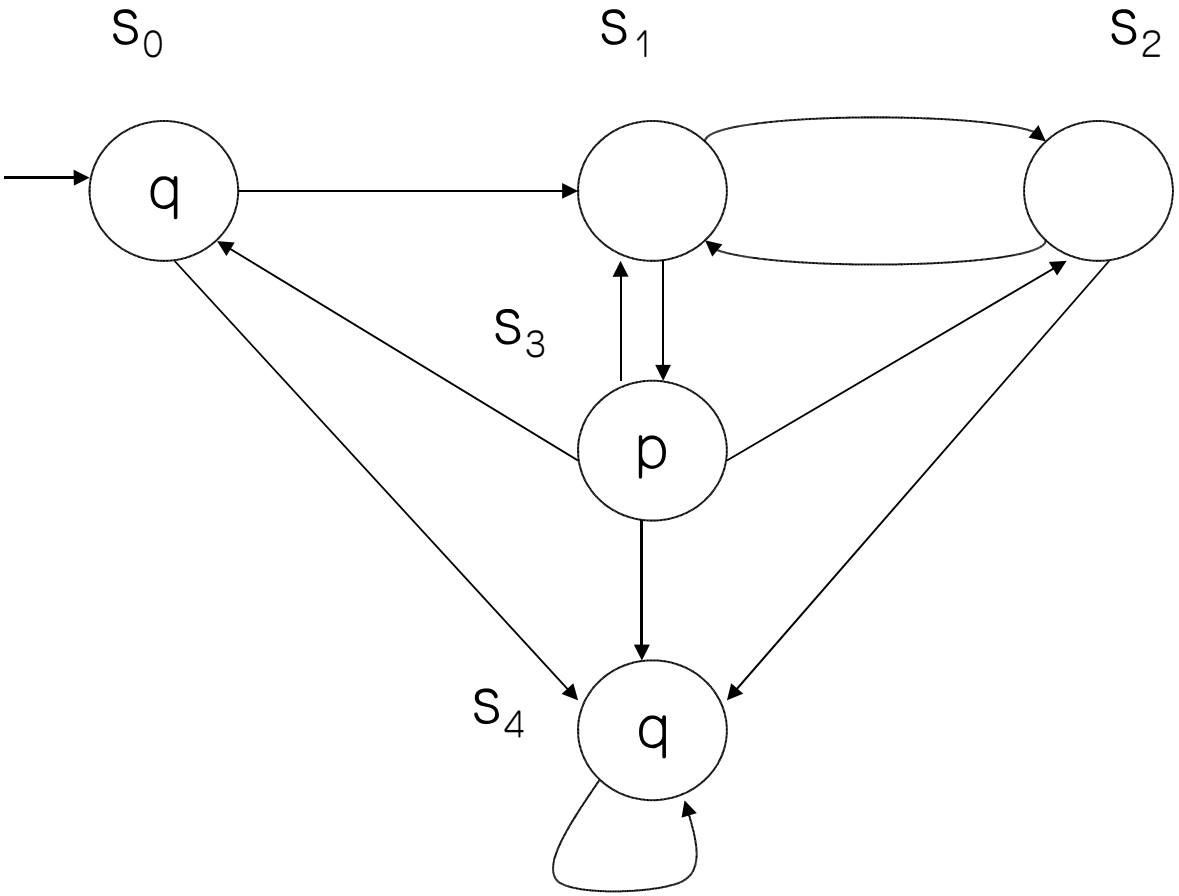
SMV Examples

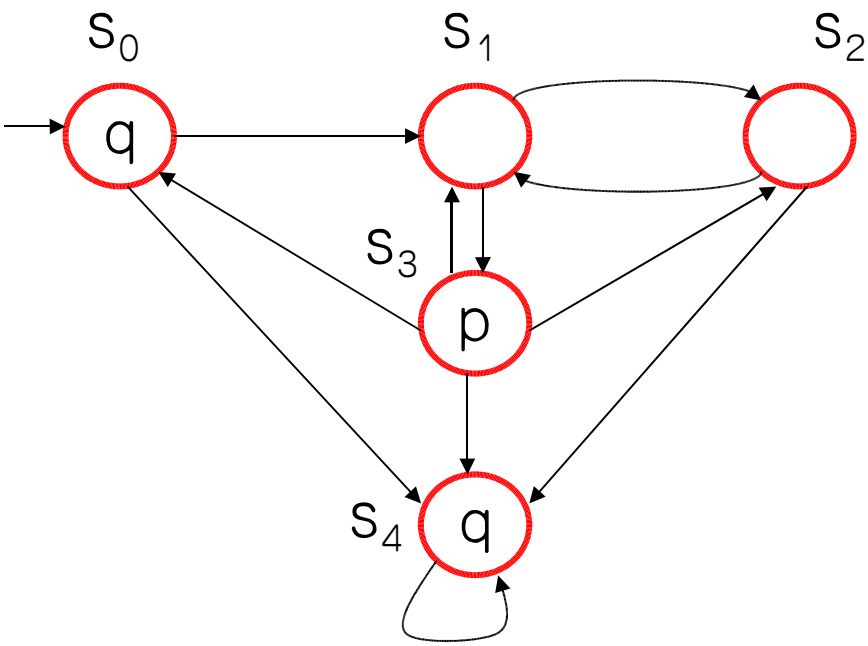
Bug Catching in 2006 Fall

Oct. 24, 2006

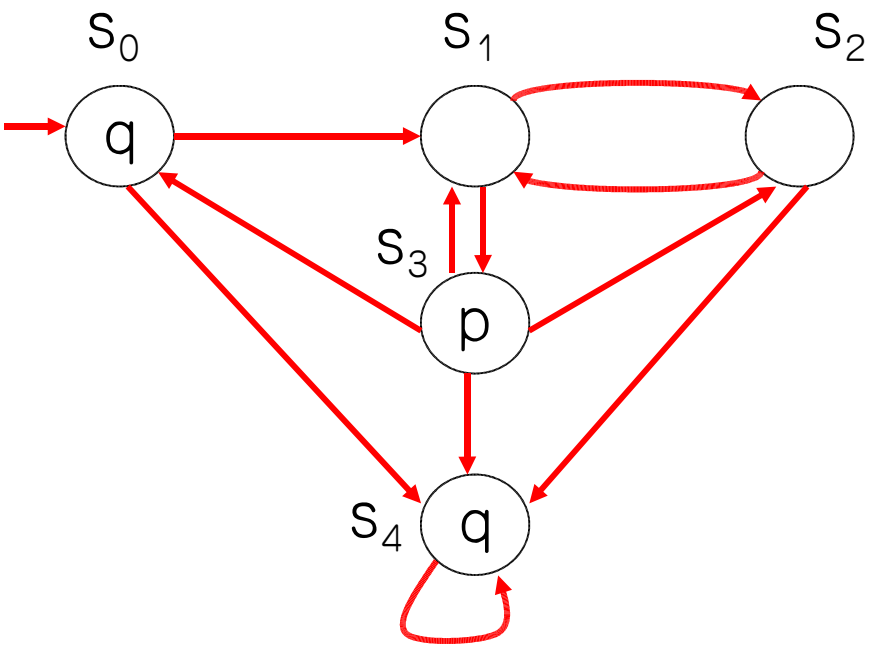
Introductory Example

EG q
AG q

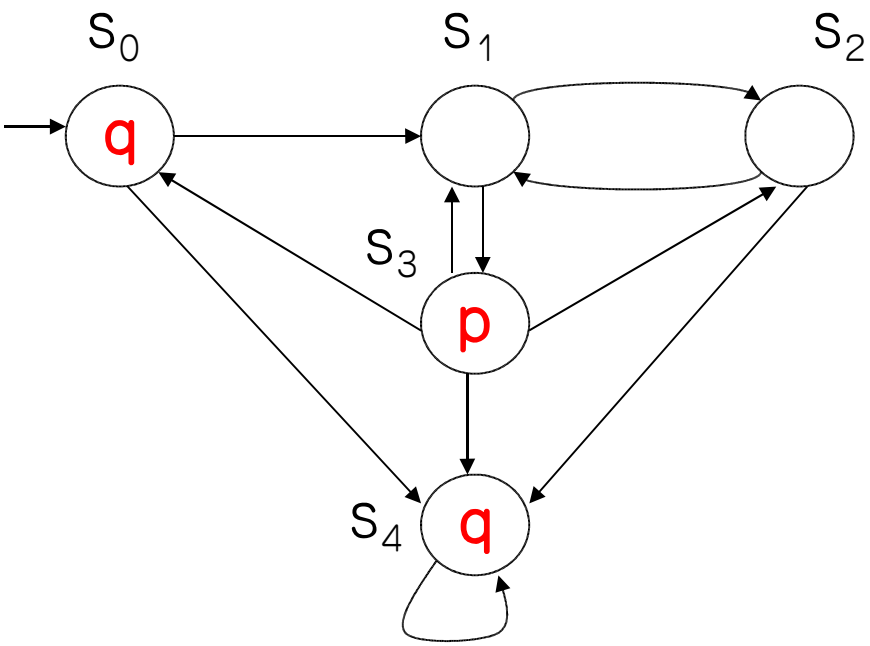




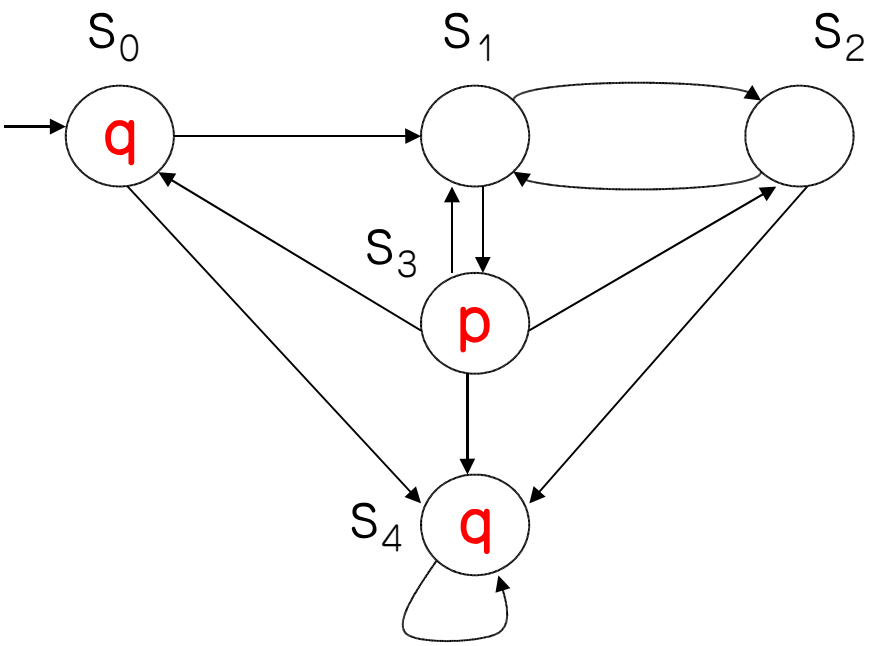

```
MODULE main
VAR
    state : {s0, s1, s2, s3, s4};
```



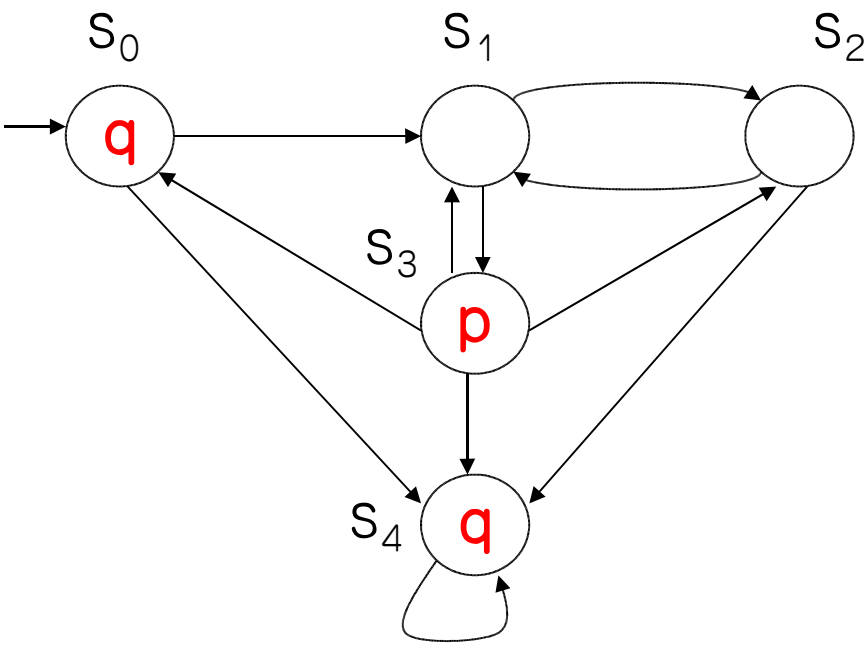

```
MODULE main
VAR
    state : {s0, s1, s2, s3, s4};
ASSIGN
    init (state) := s0;
    next(state) := case
            state = s0 : {s1, s4};
            state = s1 : {s2, s3};
            state = s2 : {s1, s4};
            state = s3 : {s0, s1, s2, s4};
            state = s4 : s4;
    esac;
```




```
MODULE main
VAR
    state : {s0, s1, s2, s3, s4};
ASSIGN
    init (state) := s0;
    next(state) := case
            state = s0 : {s1, s4};
            state = s1 : {s2, s3};
            state = s2 : {s1, s4};
            state = s3 : {s0, s1, s2, s4};
            state = s4 : s4;
    esac;
DEFINE
    p := (state = s3);
    q := (state = s0 | state = s4);
```




```
MODULE main
VAR
    state : {s0, s1, s2, s3, s4};
ASSIGN
    init (state) := s0;
    next(state) := case
            state = s0 : {s1, s4};
            state = s1 : {s2, s3};
            state = s2 : {s1, s4};
            state = s3 : {s0, s1, s2, s4};
            state = s4 : s4;
    esac;
DEFINE
    p := (state = s3);
    q := (state = s0 | state = s4);

SPEC   EG q
SPEC   AG q
```



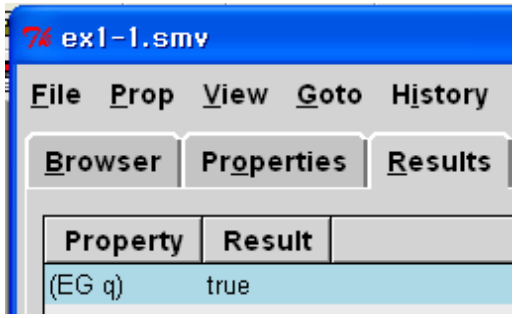
Model

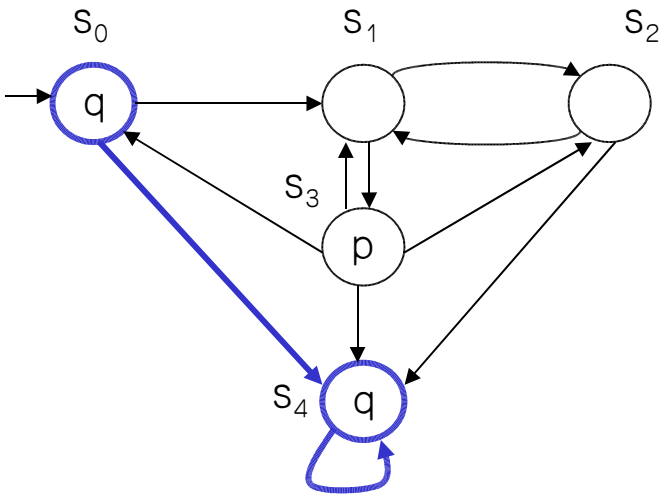
```
MODULE main
VAR
    state : {s0, s1, s2, s3, s4};
ASSIGN
    init (state) := s0;
    next(state) := case
            state = s0 : {s1, s4};
            state = s1 : {s2, s3};
            state = s2 : {s1, s4};
            state = s3 : {s0, s1, s2, s4};
            state = s4 : s4;
    esac;
DEFINE
    p := (state = s3);
    q := (state = s0 | state = s4);
```

Property

```
SPEC   EG q
SPEC   AG q
```

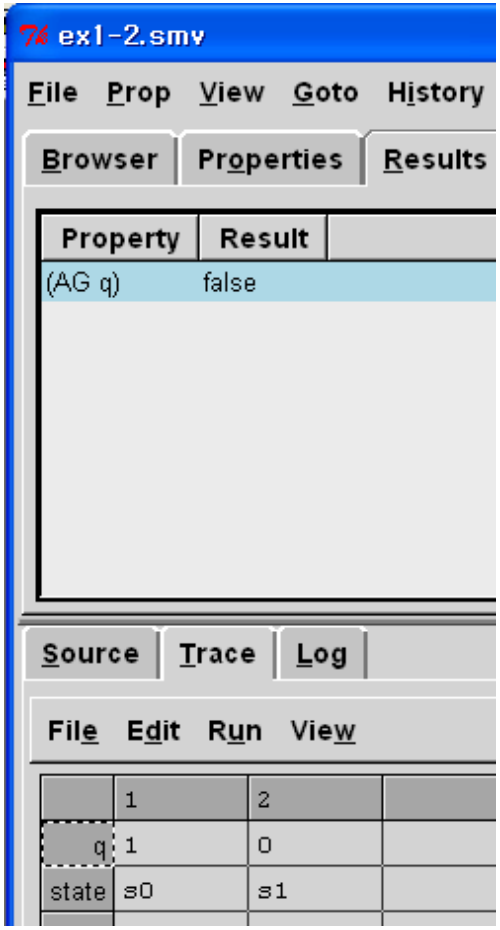
EG q is ture

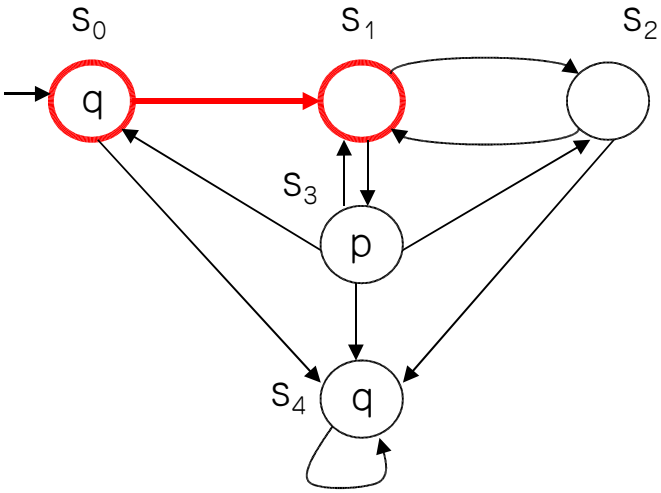




witness

AG q is false

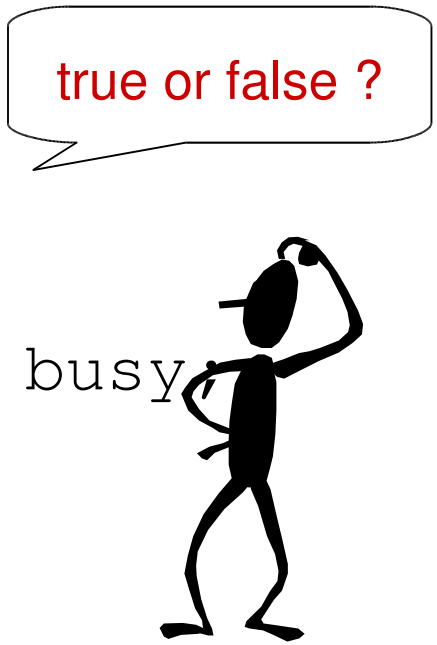




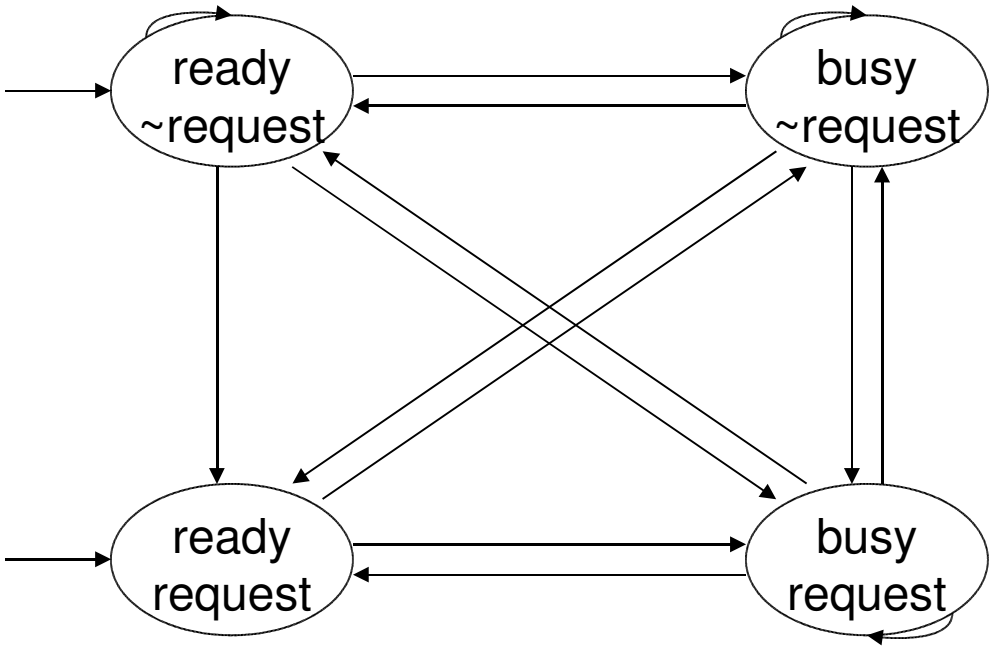
counterexample

A Simple Example

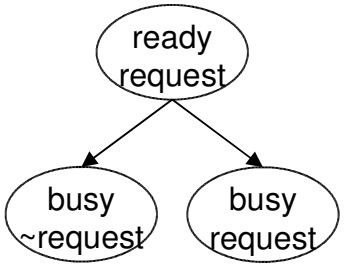
```
MODULE main
VAR
   request : boolean;
   state   : {ready,busy};
ASSIGN
   init(state) := ready;
   next(state) := case
     state = ready & request : busy;
     1 : {ready,busy};
   esac;

SPEC  AG(request -> AF state = busy)
```



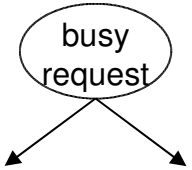
Kripke structure



Computation tree

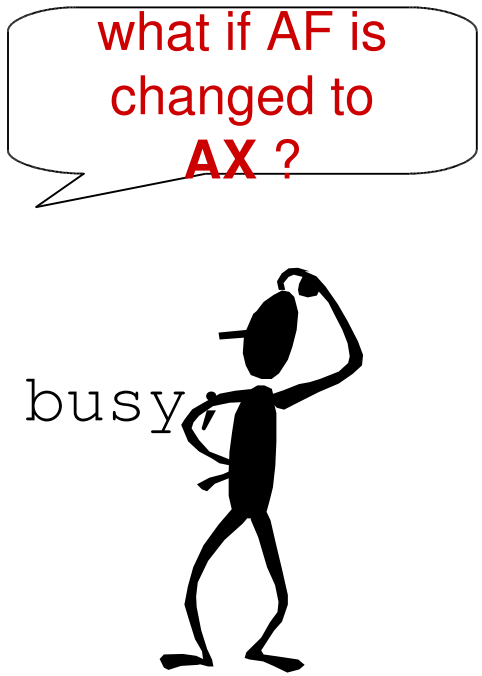


property hold

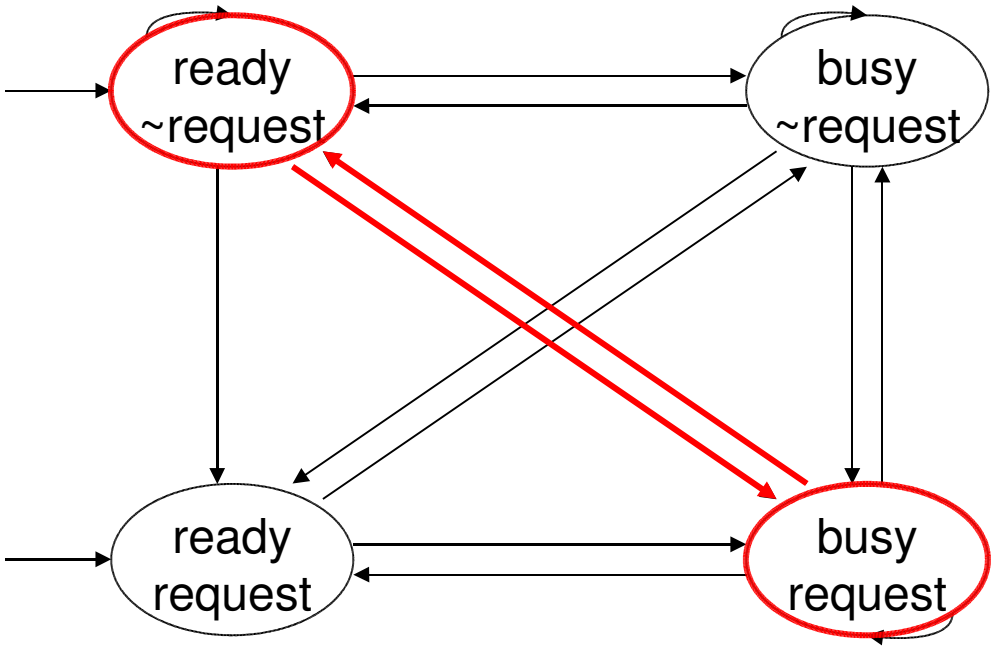


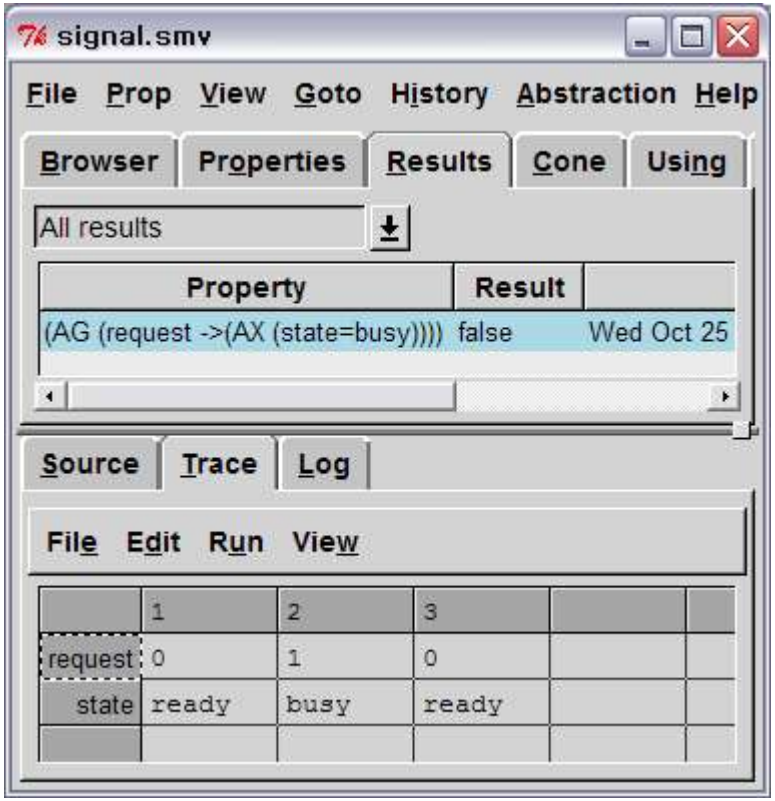
property already hold

```
MODULE main
VAR
   request : boolean;
   state   : {ready,busy};
ASSIGN
   init(state) := ready;
   next(state) := case
     state = ready & request : busy;
     1 : {ready,busy};
   esac;

SPEC  AG(request -> AX state = busy)
```



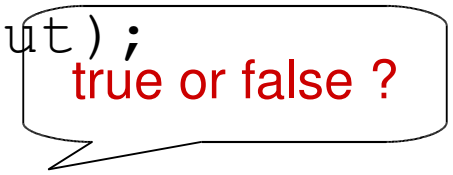
the property is false



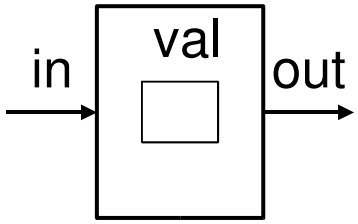


A Three Bit Counter

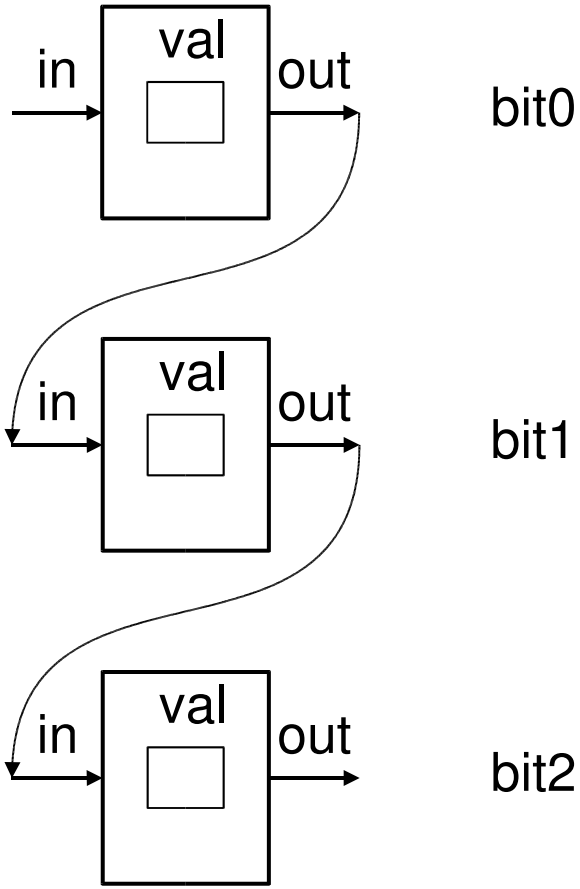
```
MODULE main
VAR
  bit0 : counter_cell(1);
  bit1 : counter_cell(bit0.carry_out);
  bit2 : counter_cell(bit1.carry_out);

SPEC  AG AF bit2.carry_out

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



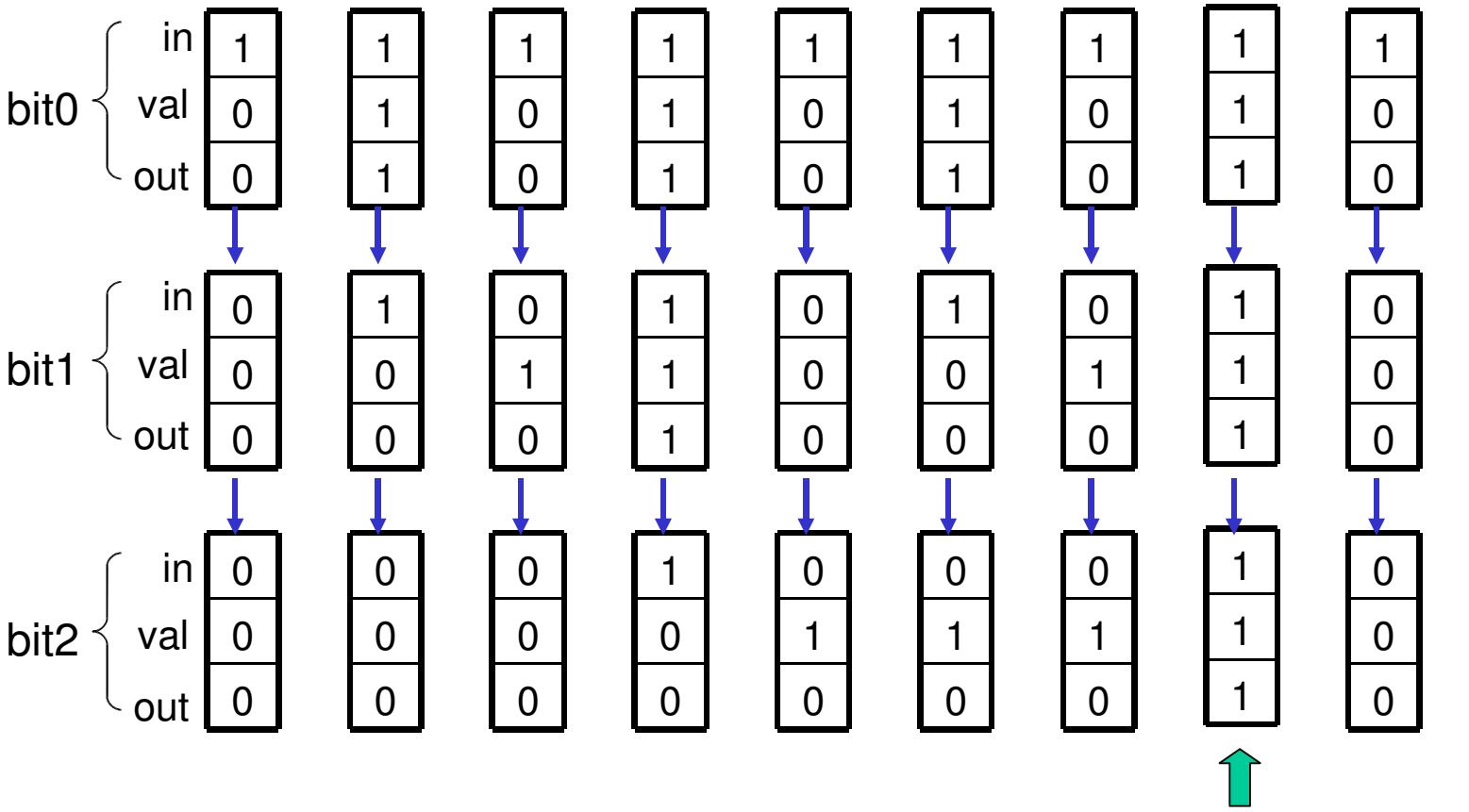
module declaration



module instantiations

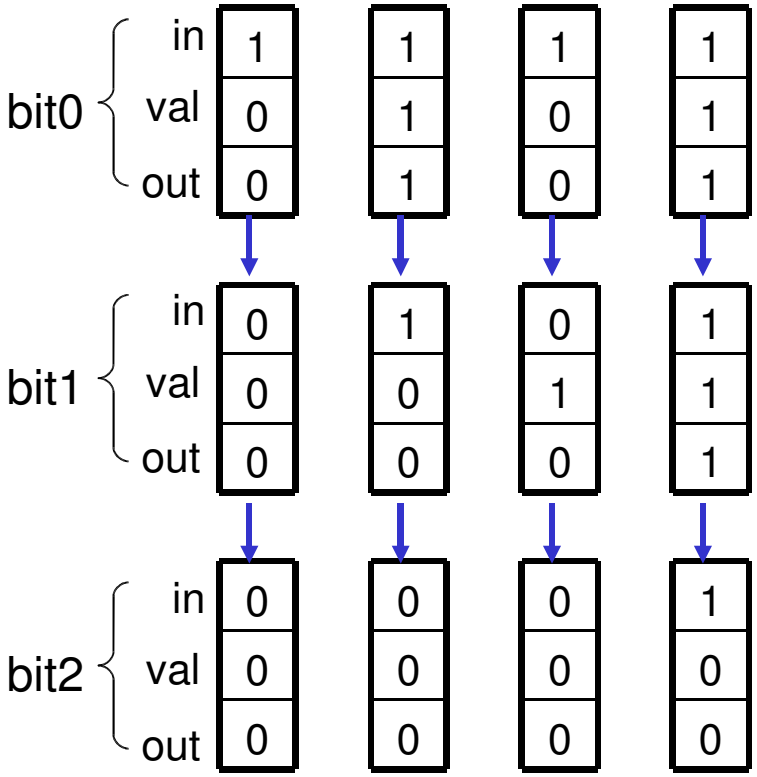


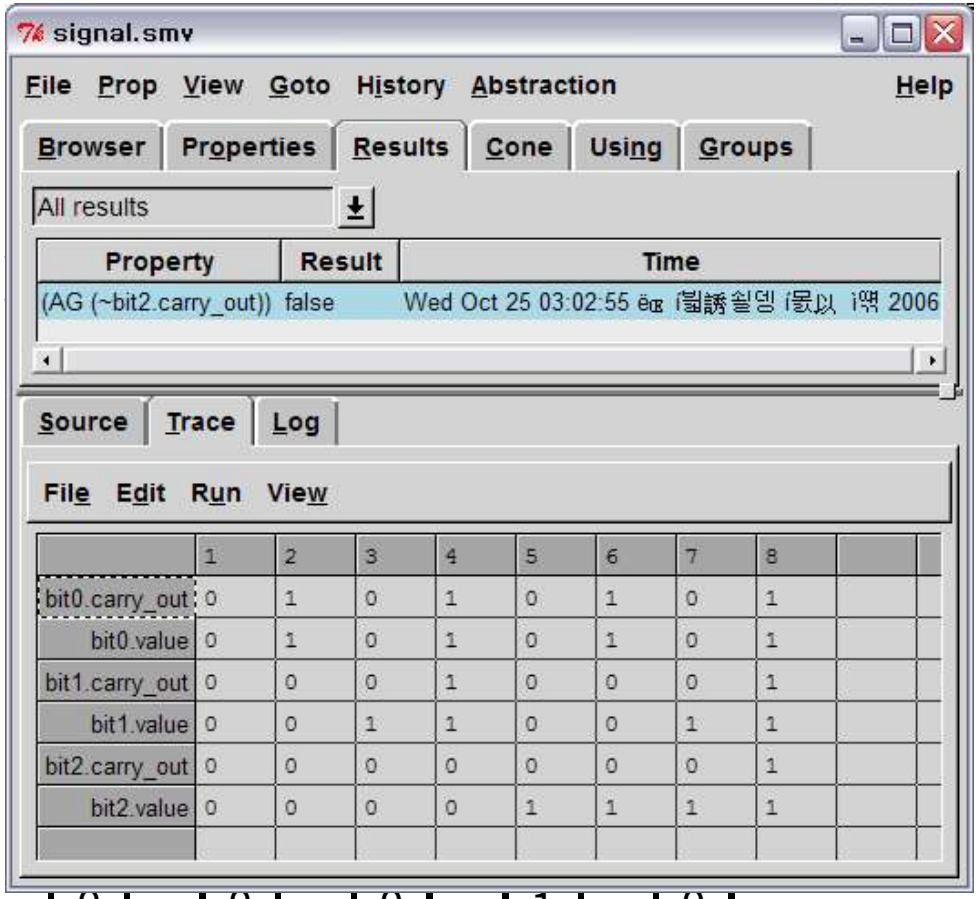
AG AF bit2.carry_out
is true



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

SPEC  AG (!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;
```

AG (!bit2.carry_out
is false





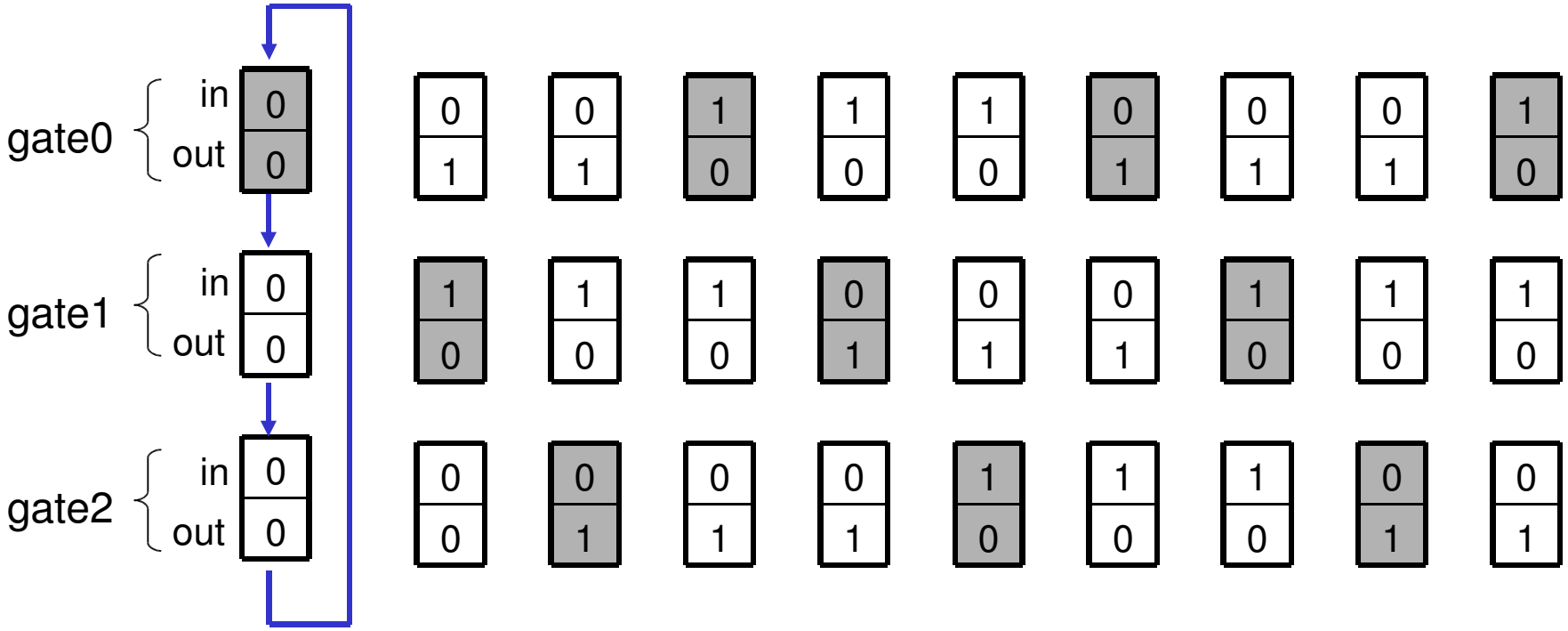
bit2.carry_out is ture

Inverter Ring

```
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) := 0;
   next(output) := !input;

FAIRNESS
   running
```

true or false ?

In asynchronous composition, a step of the computation is a step by exactly one component. The process to execute is assumed to choose gate0, gate1, and gate2 repeatedly.

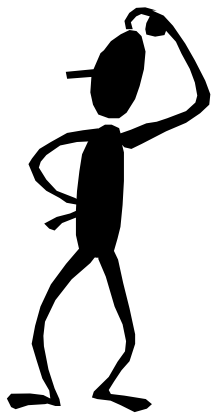



```
(AG AF gate1.output) & (AG AF !
      gate1.output) is true
```

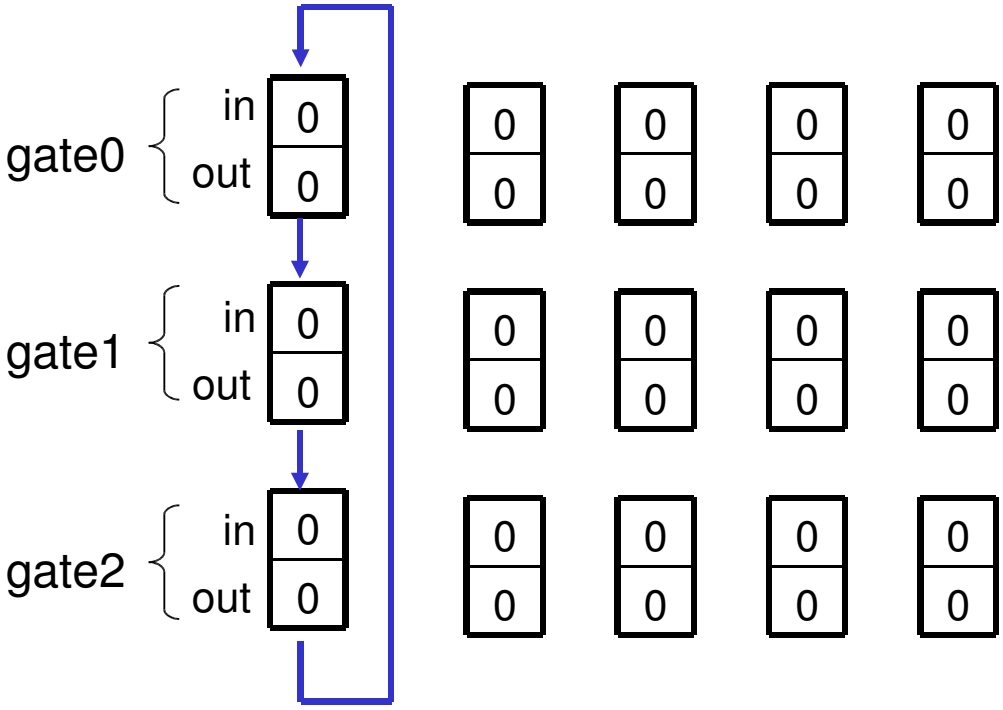
Inverter Ring

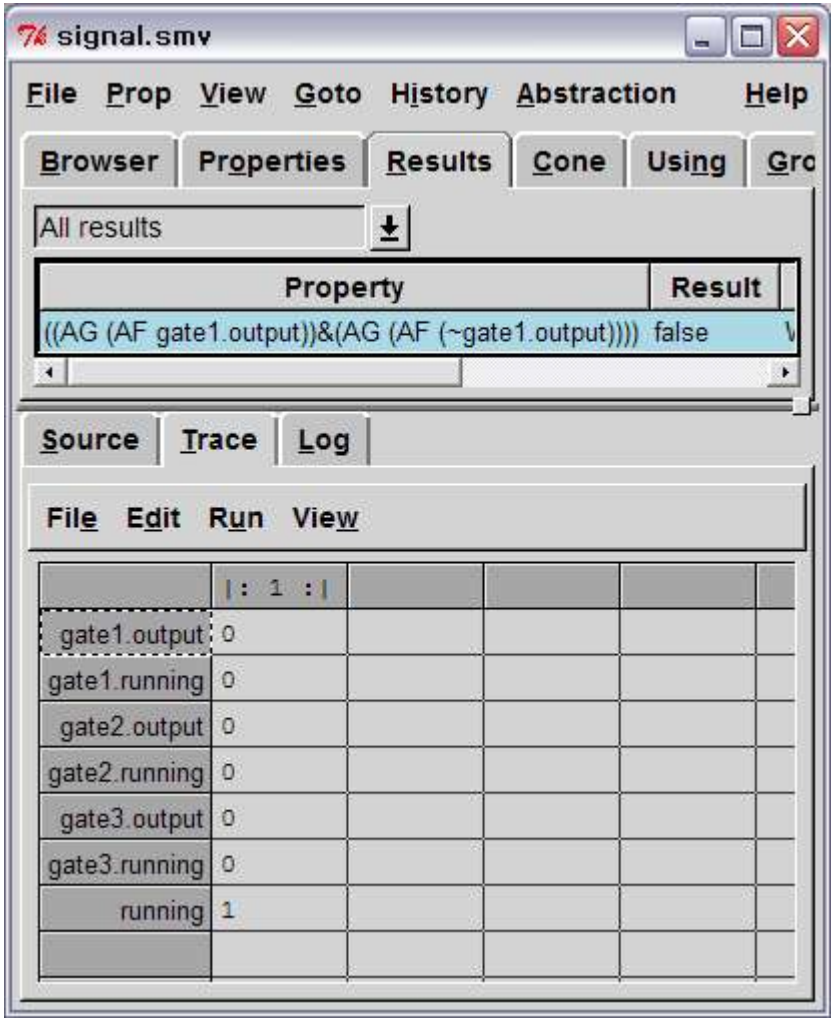
```
MODULE main
VAR
  gate1 : process inverter(gate3.output);
  gate2 : process inverter(gate1.output);
  gate3 : process inverter(gate2.output);

SPEC  (AG AF gate1.output) & (AG AF !
  gate1.output)

MODULE inverter(input)
VAR
  output : boolean;
ASSIGN
  init(output) := 0;
  next(output) := !input;

FAIRNESS
  running
```

what if FAIRNESS is deleted ?



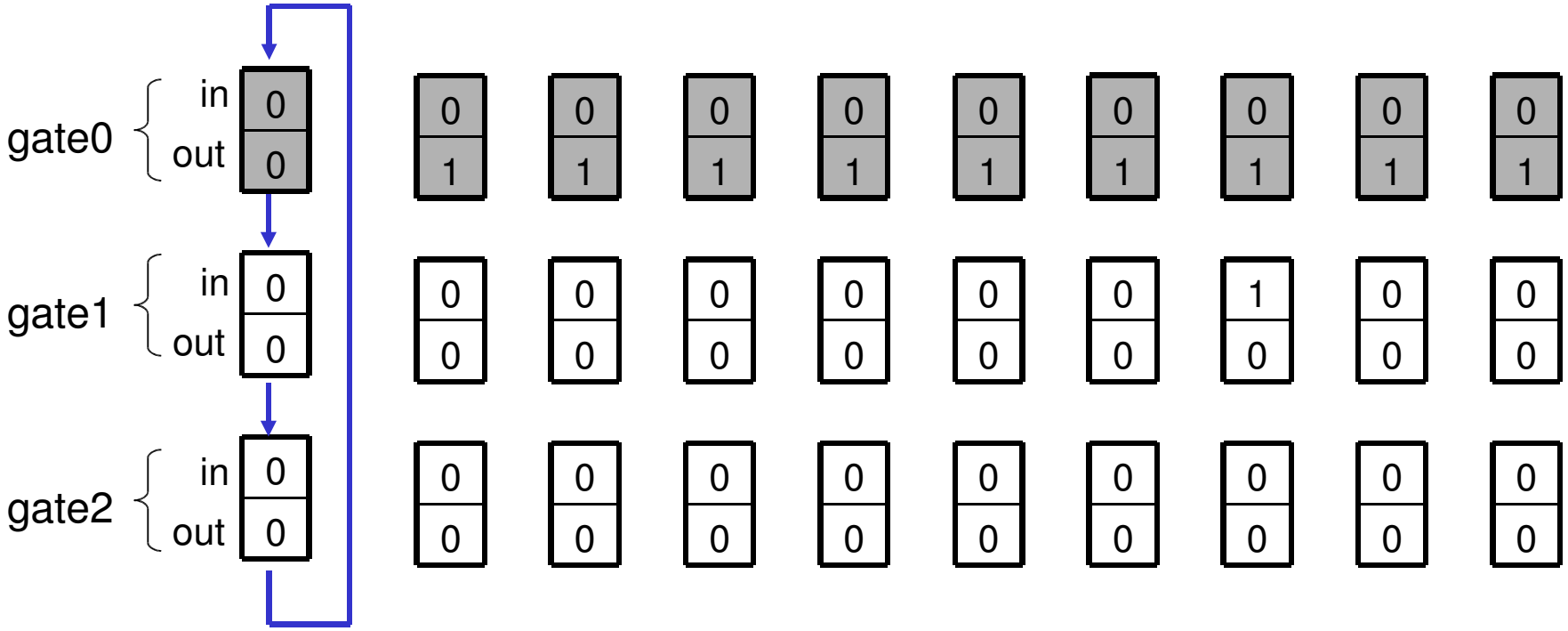
Without fairness constraint, there is a possibility that no processes are chosen.





(AG AF gate1.output) & (AG AF ! gate1.output) is false

Without fairness constraint, there is a possibility that the process gate0 is chosen forever.



`(AG AF gate1.output) & (AG AF ! gate1.output)` is false