



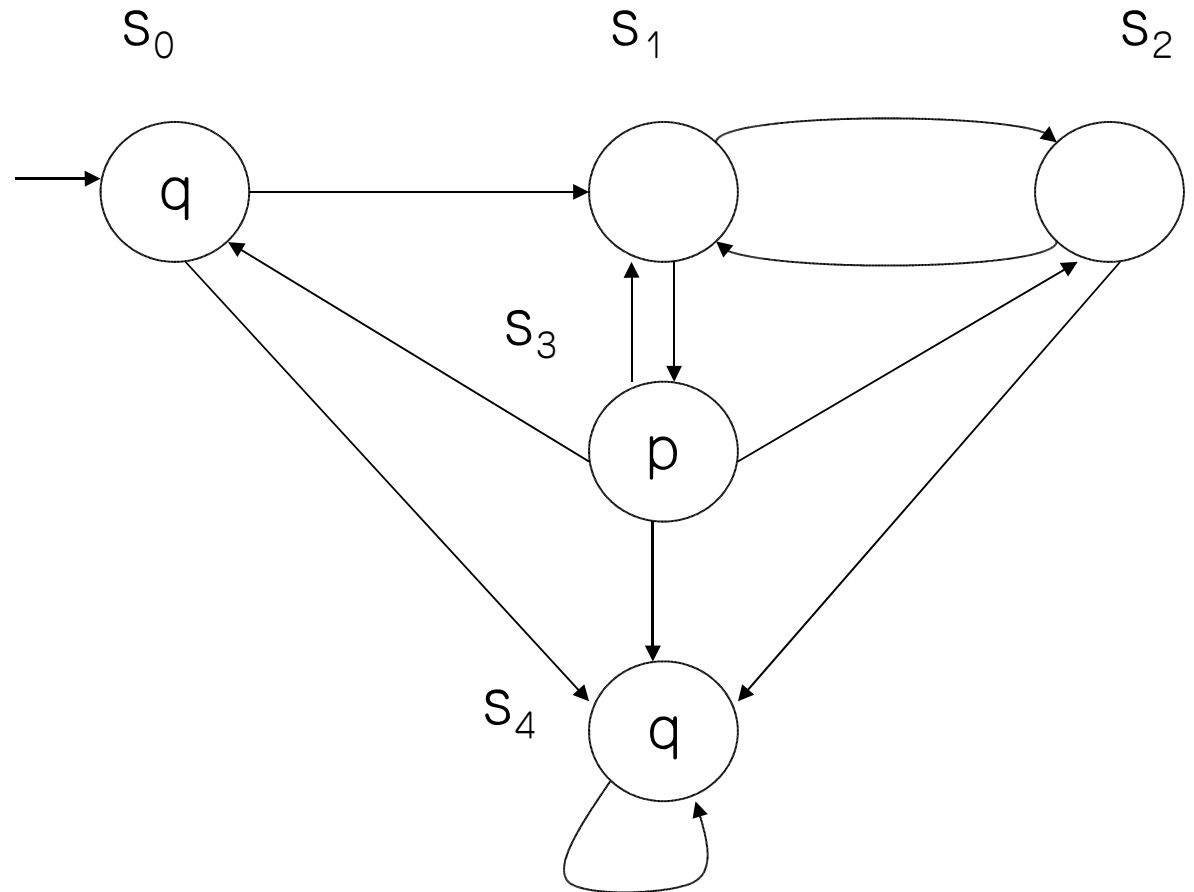
# SMV Examples

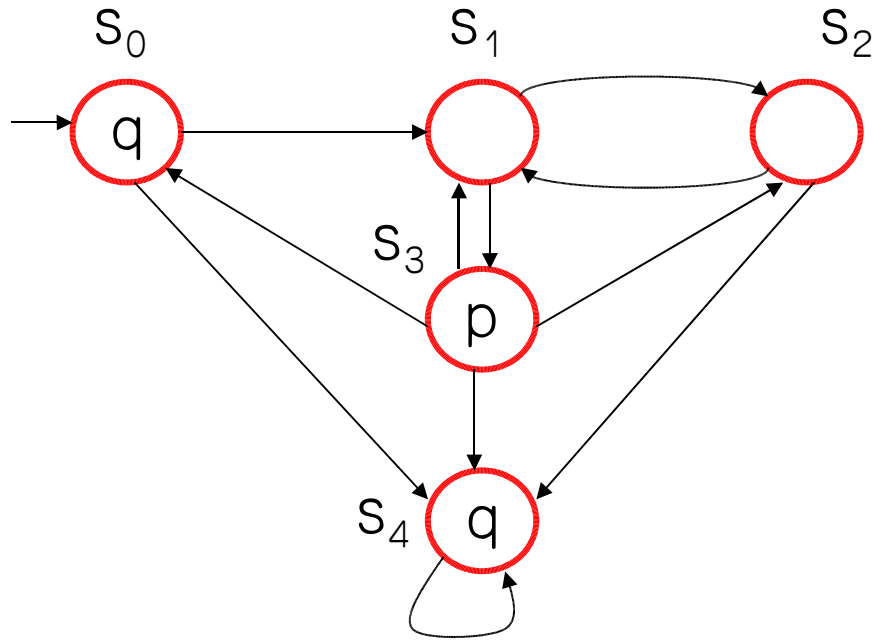
**Bug Catching** in 2006 Fall

Oct. 24, 2006

# Introductory Example

EG q  
AG q

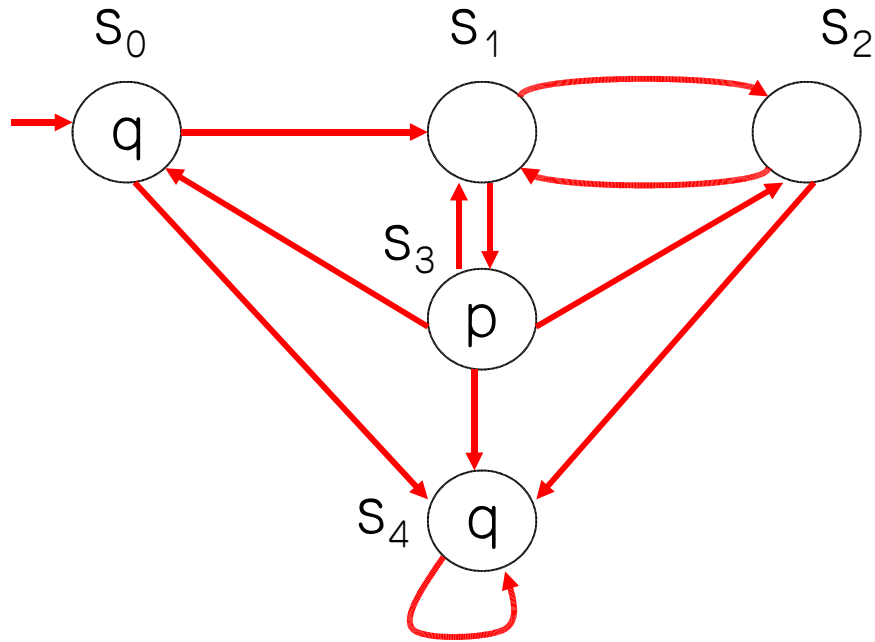




MODULE main

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};

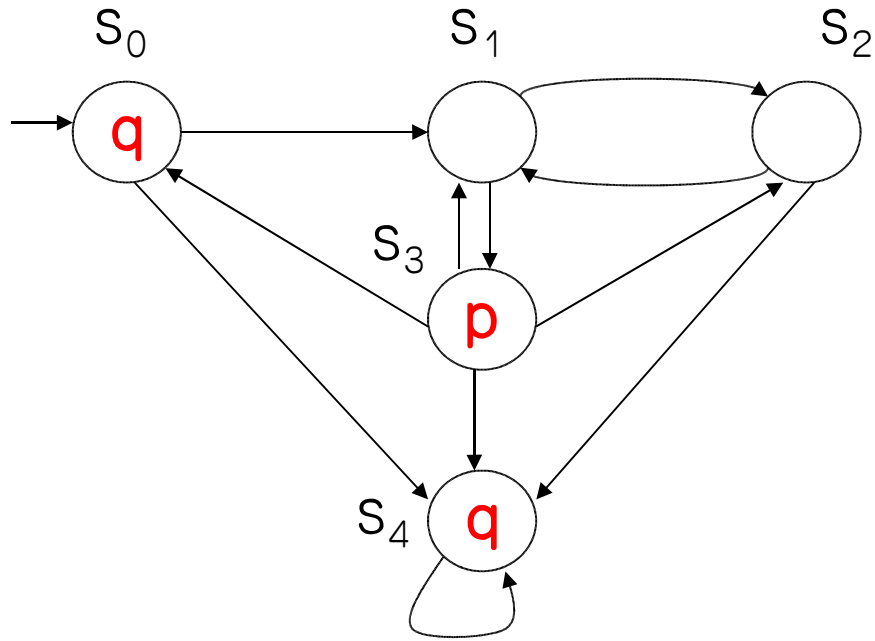
state = s1 : {s2, s3};

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};

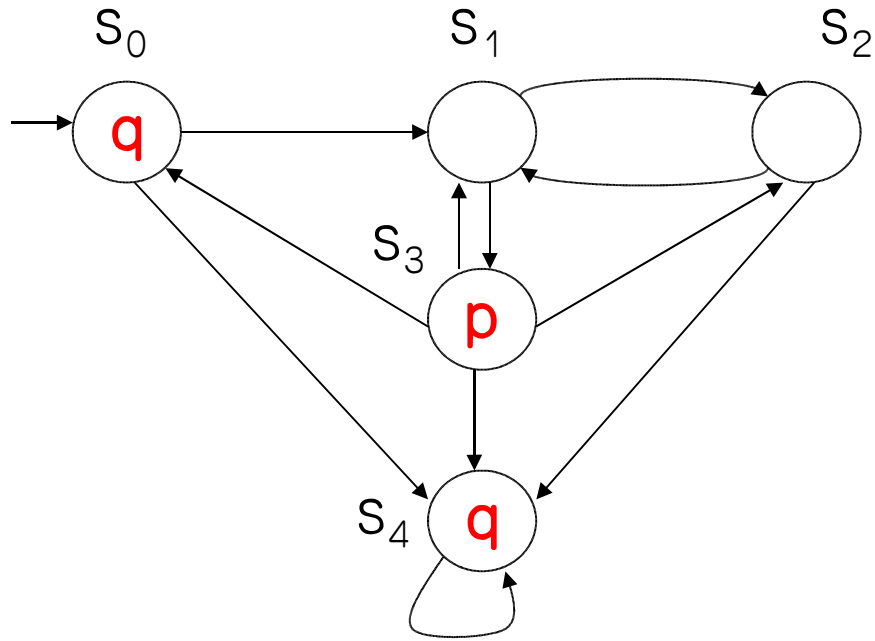
state = s4 : s4;

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;

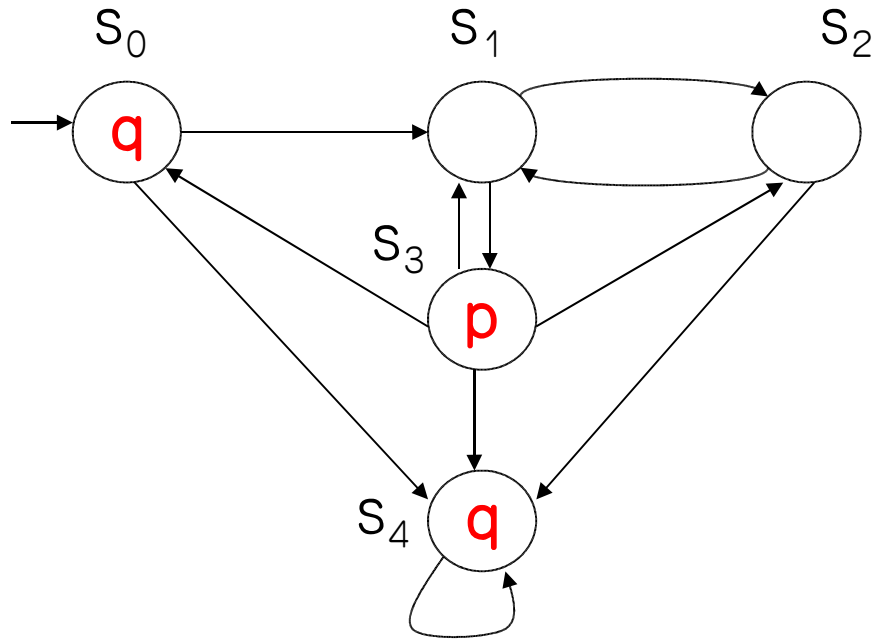
DEFINE

p := (state = s3);

q := (state = s0 | state = s4);

SPEC EG q

SPEC AG q



Model 

Property 

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);

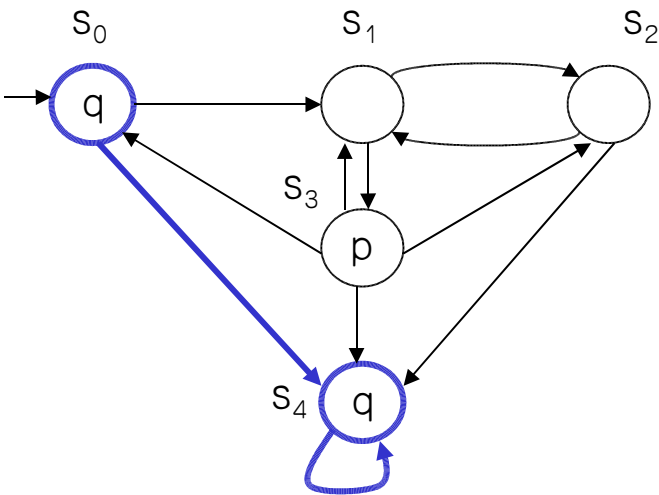
SPEC EG q

SPEC AG q

**EG q is true**

74 ex1-1.smv

Property	Result
(EG q)	true



witness

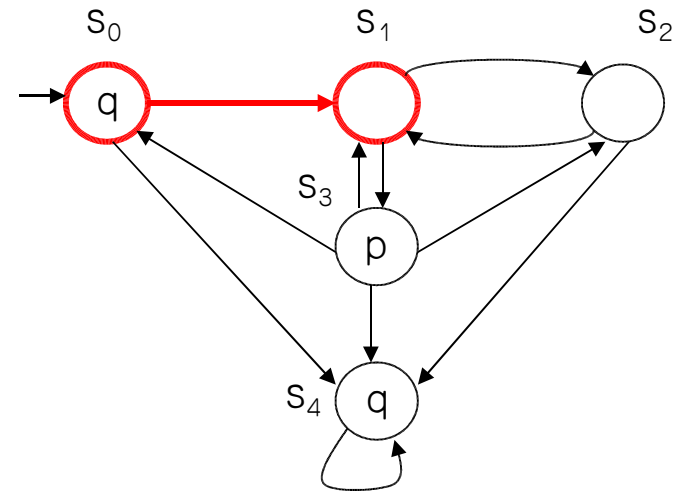
**AG q is false**

74 ex1-2.smv

Property	Result
(AG q)	false

	1	2
q	1	0
state	s0	s1



counterexample

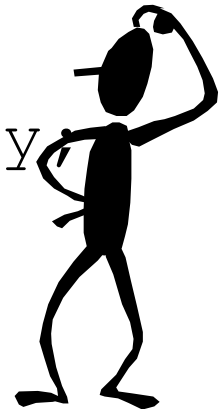


# A Simple Example

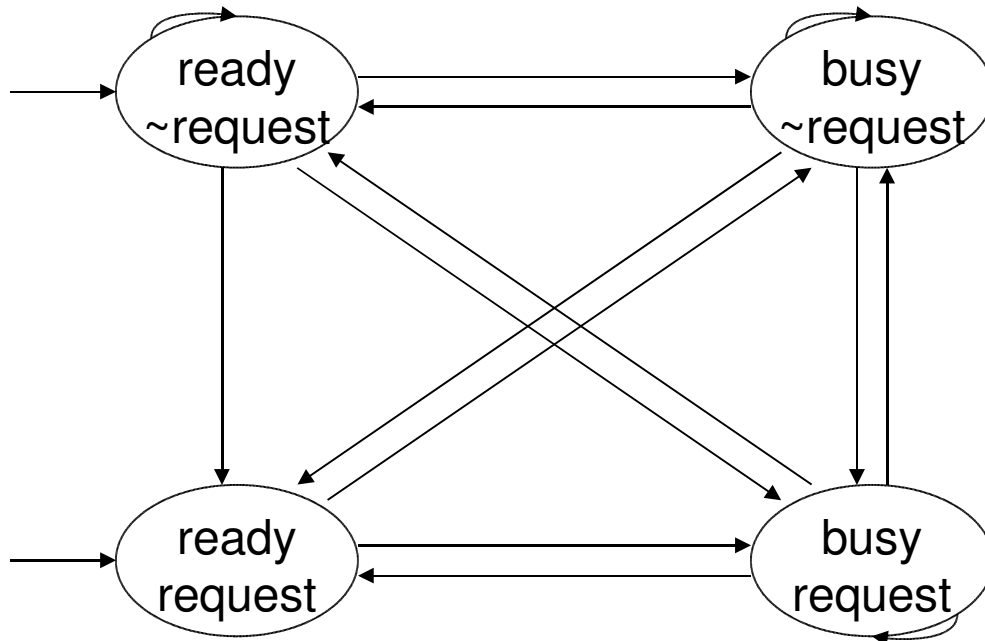
```
MODULE main
VAR
  request : boolean;
  state   : {ready,busy};
ASSIGN
  init(state) := ready;
  next(state) := case
    state = ready & request : busy;
    1 : {ready,busy};
  esac;

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

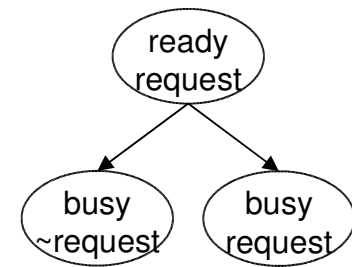
true or false ?



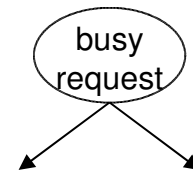
## Kripke structure



## Computation tree



property hold

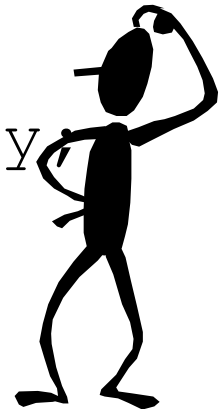


property already hold

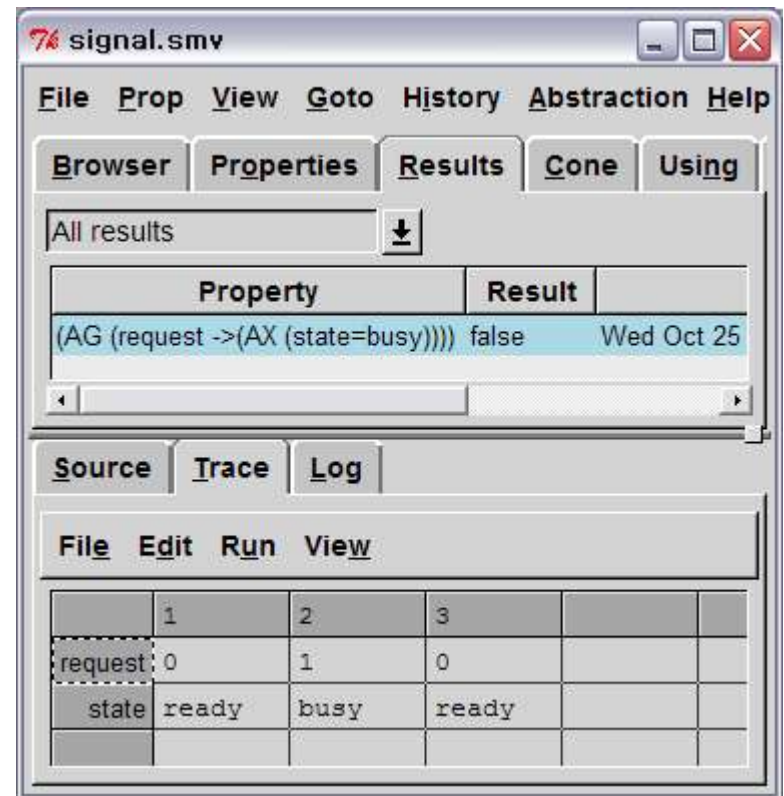
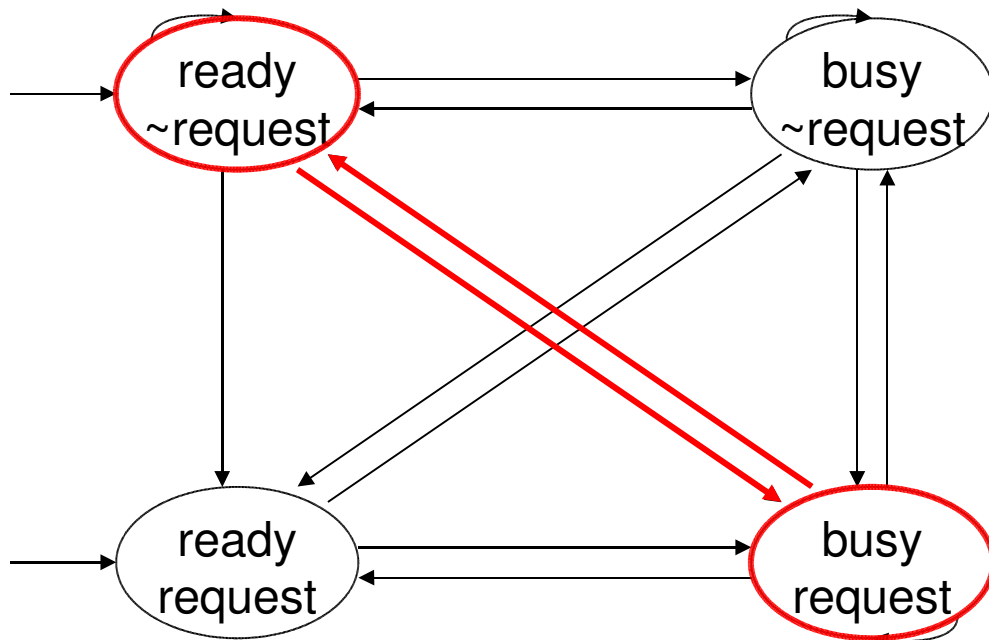
```
MODULE main
VAR
  request : boolean;
  state   : {ready,busy};
ASSIGN
  init(state) := ready;
  next(state) := case
    state = ready & request : busy;
    1 : {ready,busy};
  esac;

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

what if AF is  
changed to  
**AX** ?



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);
```

true or false ?

```
SPEC  AG AF bit2.carry_out
```

```
MODULE counter_cell(carry_in)
```

```
VAR
```

```
    value : boolean;
```

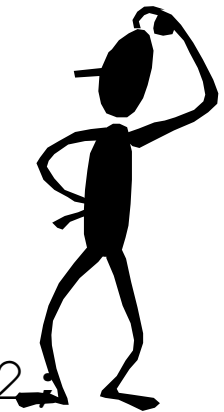
```
ASSIGN
```

```
    init(value) := 0;
```

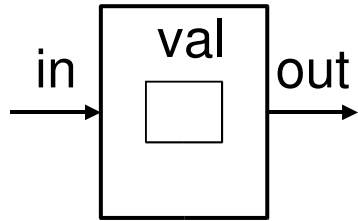
```
    next(value) := value + carry_in mod 2;
```

```
DEFINE
```

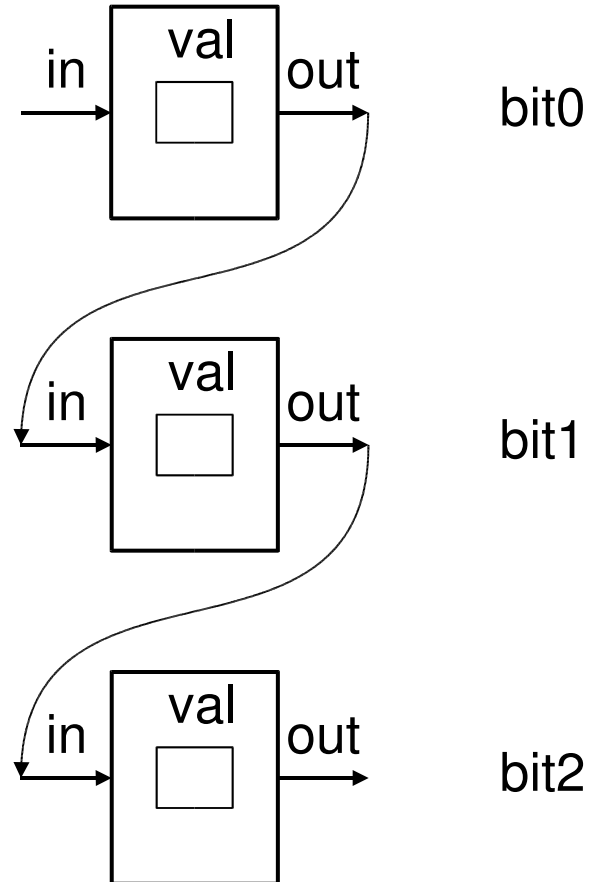
```
    carry_out := value & carry_in;
```



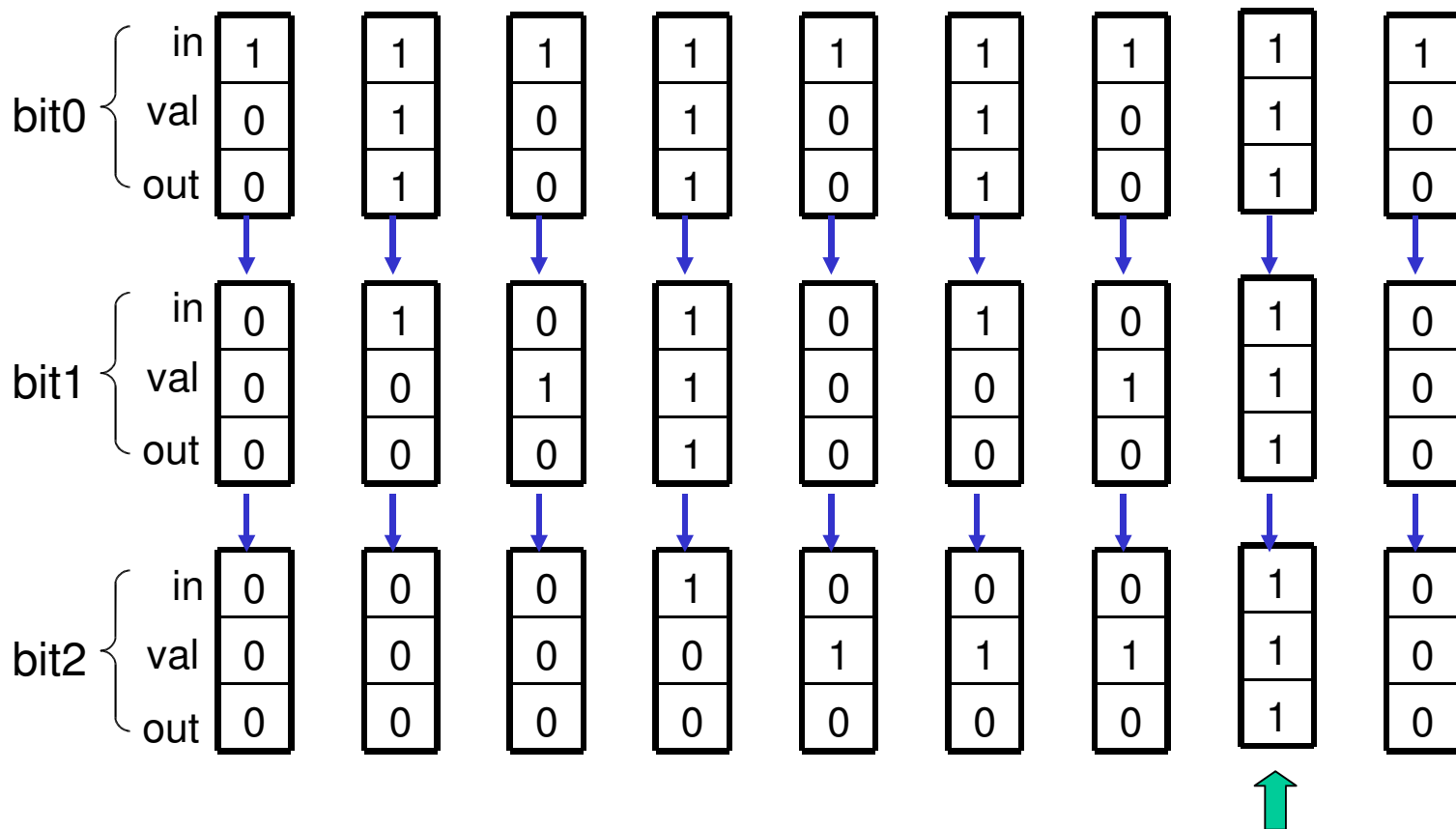
## module declaration



## module instantiations



AG AF bit2.carry\_out  
is true



bit2.carry\_out is true

```
MODULE main
```

```
VAR
```

```
    bit0 : counter_cell(1);
```

```
    bit1 : counter_cell(bit0.carry_out);
```

```
    bit2 : counter_cell(bit1.carry_out);
```

true or false ?

```
SPEC  AG (!bit2.carry_out)
```

```
MODULE counter_cell(carry_in)
```

```
VAR
```

```
    value : boolean;
```

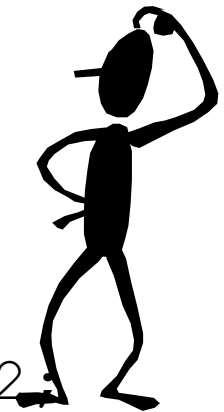
```
ASSIGN
```

```
    init(value) := 0;
```

```
    next(value) := value + carry_in mod 2;
```

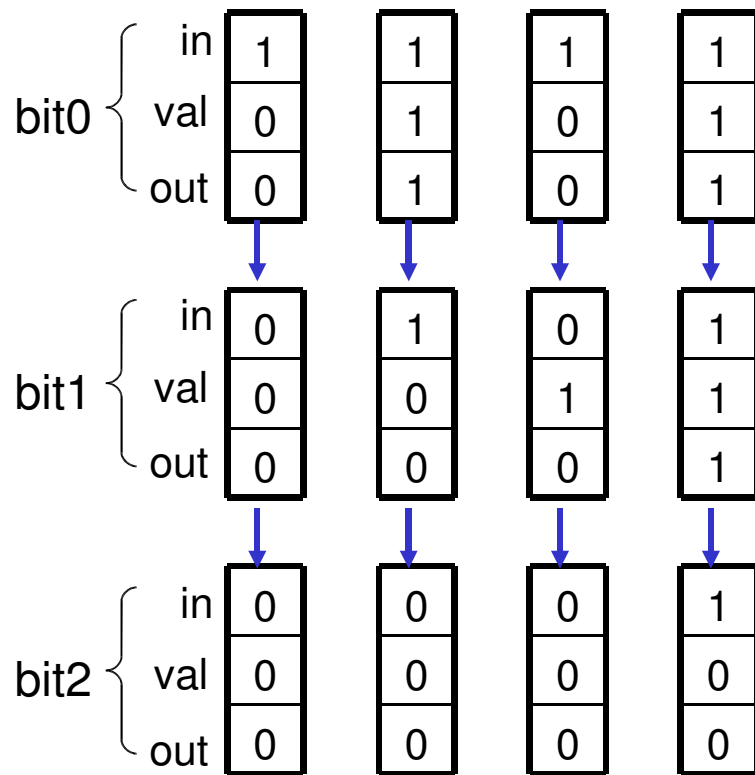
```
DEFINE
```

```
    carry_out := value & carry_in;
```





AG (!bit2.carry\_out  
is false



signal.smv

File Prop View Goto History Abstraction Help

Browser Properties Results Cone Using Groups

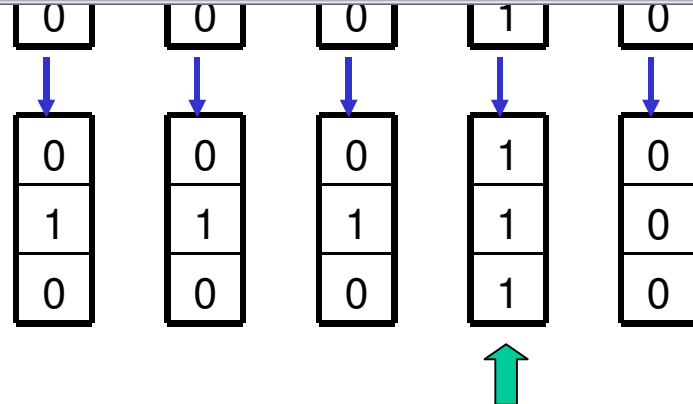
All results

Property	Result	Time
(AG (~bit2.carry_out))	false	Wed Oct 25 03:02:55 2006

Source Trace Log

File Edit Run View

	1	2	3	4	5	6	7	8
bit0.carry_out	0	1	0	1	0	1	0	1
bit0.value	0	1	0	1	0	1	0	1
bit1.carry_out	0	0	0	1	0	0	0	1
bit1.value	0	0	1	1	0	0	1	1
bit2.carry_out	0	0	0	0	0	0	0	1
bit2.value	0	0	0	0	1	1	1	1



bit2.carry\_out 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
```

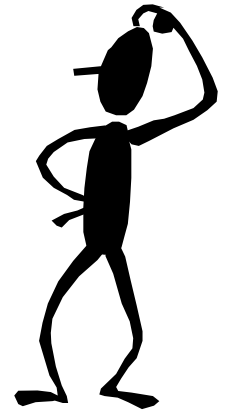
```
  init(output) := 0;
```

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

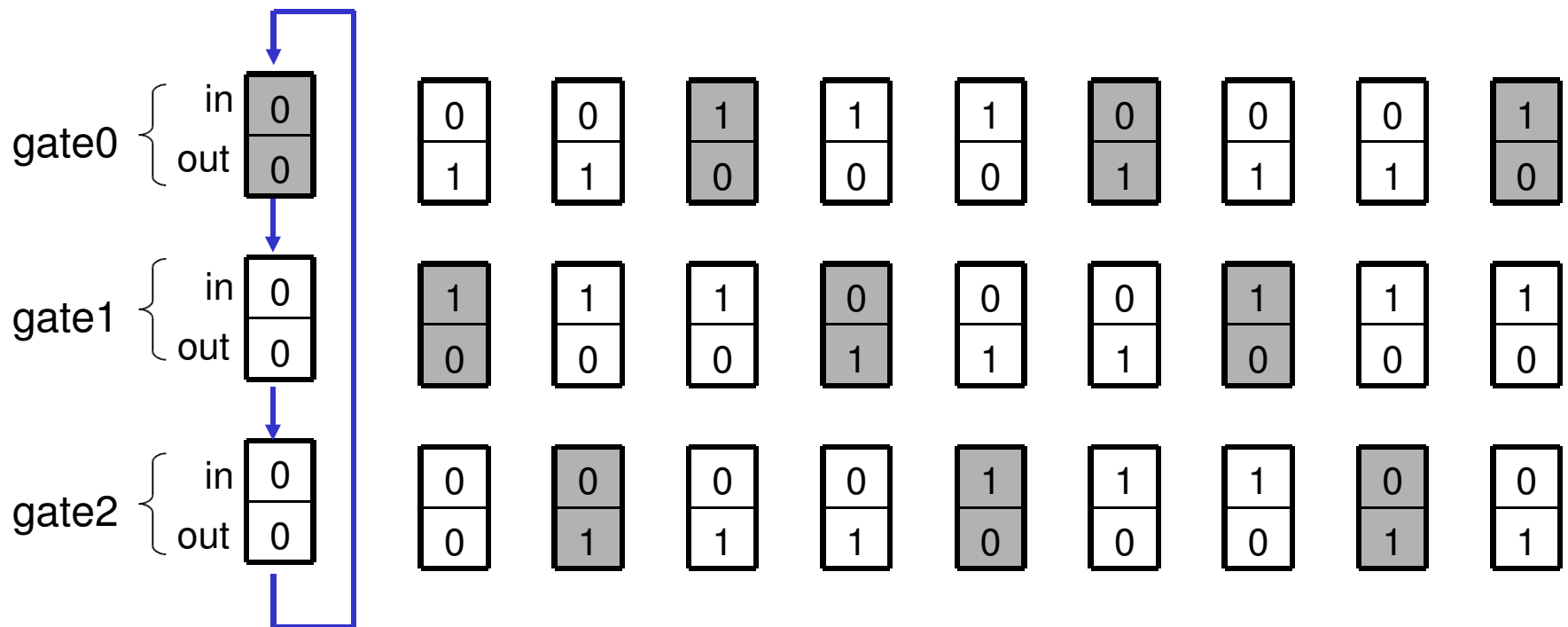
```
FAIRNESS
```

```
  running
```

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.



$(\text{AG AF gate1.output}) \ \& \ (\text{AG AF ! gate1.output})$  is true

# Inverter Ring

```
MODULE main
```

```
VAR
```

```
  gate1 : process inverter(gate3.output);
```

```
  gate2 : process inverter(gate1.output);
```

```
  gate3 : process inverter(gate2.output);
```

what if  
FAIRNESS is  
deleted?

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

```
MODULE inverter(input)
```

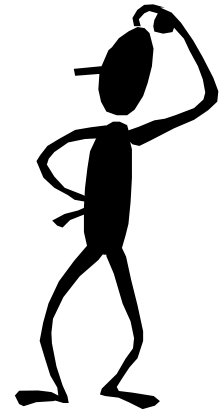
```
VAR
```

```
  output : boolean;
```

```
ASSIGN
```

```
  init(output) := 0;
```

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

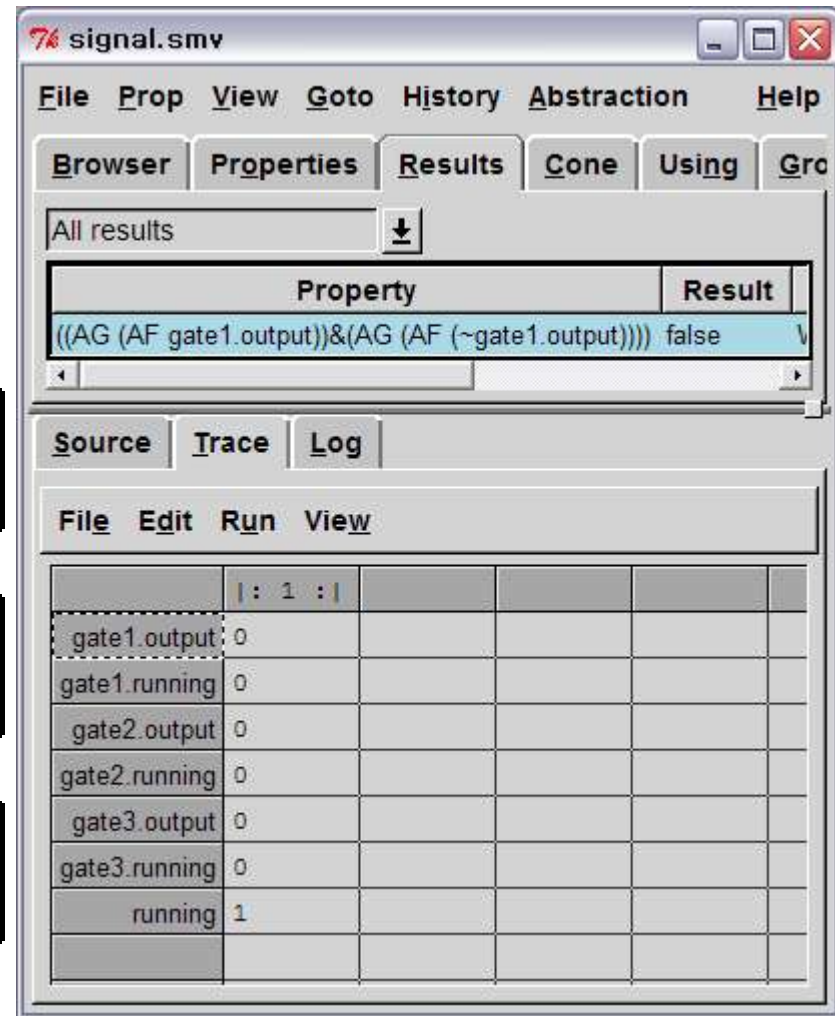
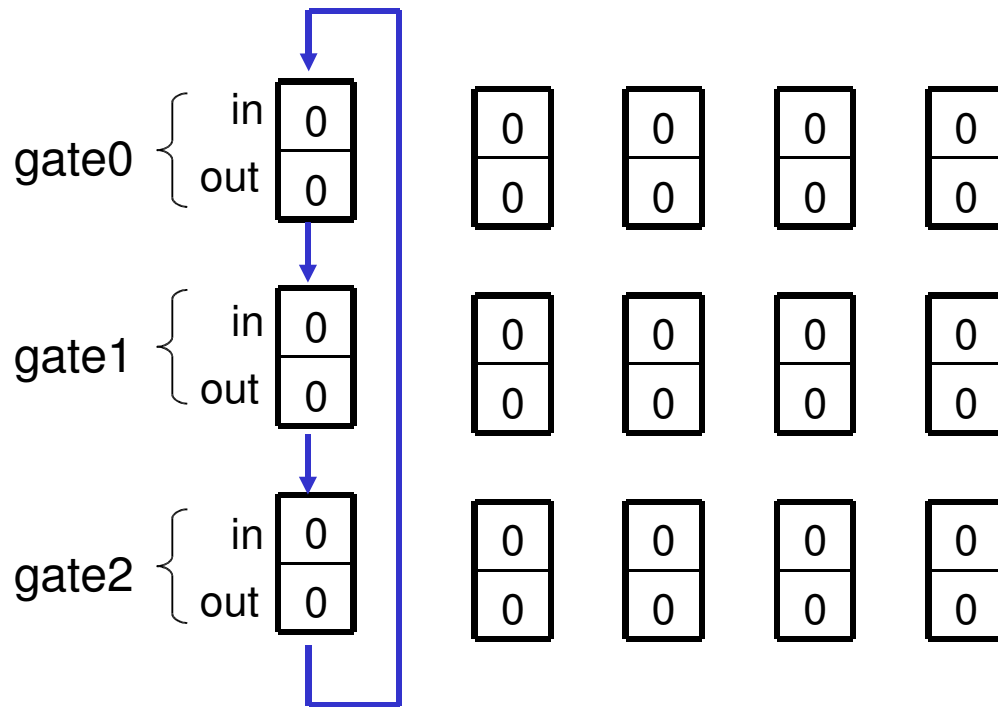


---

FAIRNESS

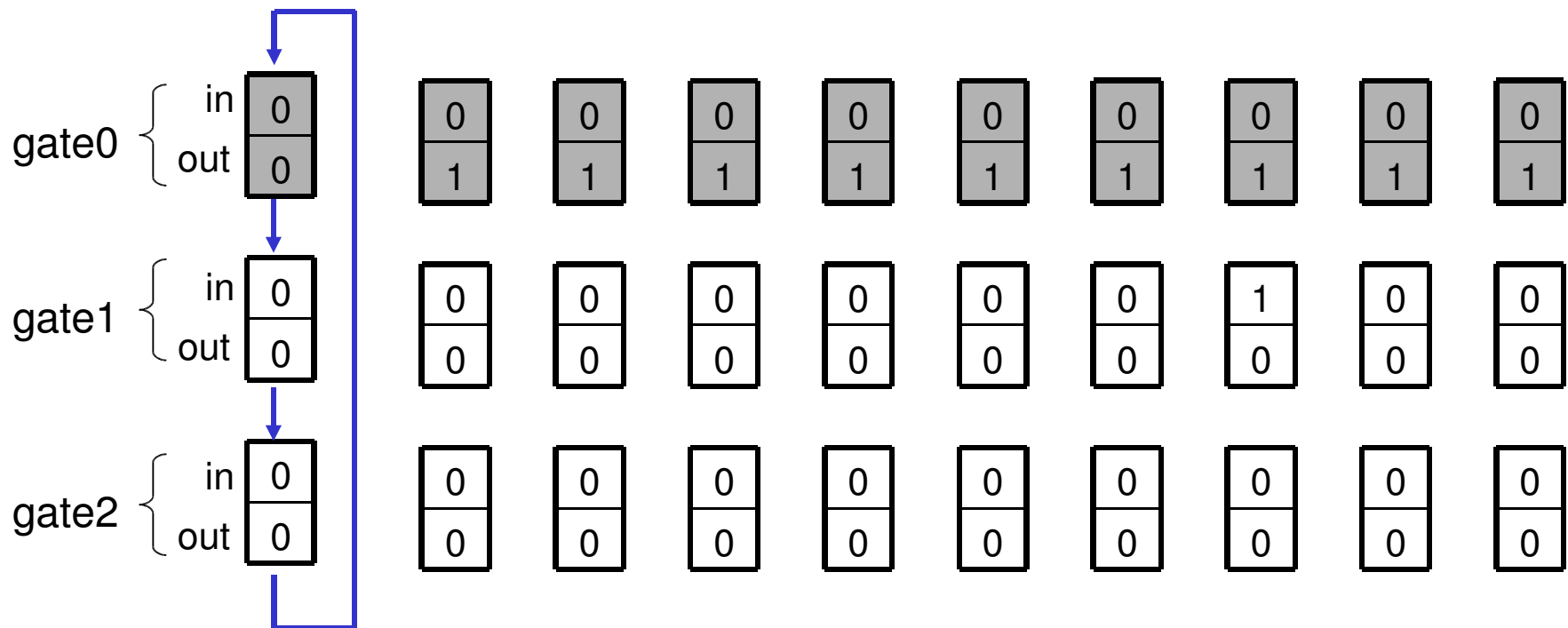
running

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



$(AG \ AF \ \text{gate1.output}) \ \& \ (AG \ AF \ ! \ \text{gate1.output})$  is false

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



$(\text{AG AF gate1.output}) \ \& \ (\text{AG AF ! gate1.output})$  is false