Unit-2: Model-checker NuSMV

B. Srivathsan

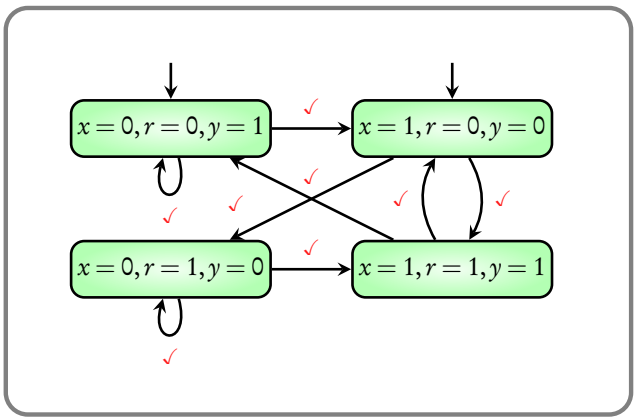
Chennai Mathematical Institute

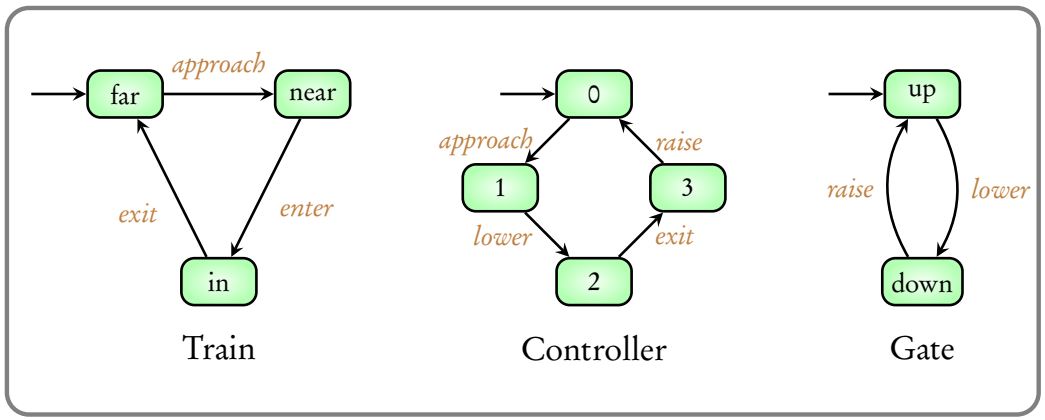
NPTEL-course

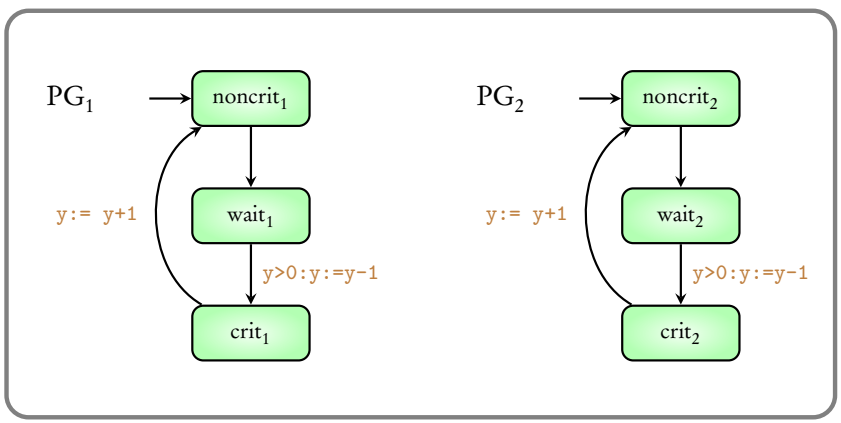
July - November 2015

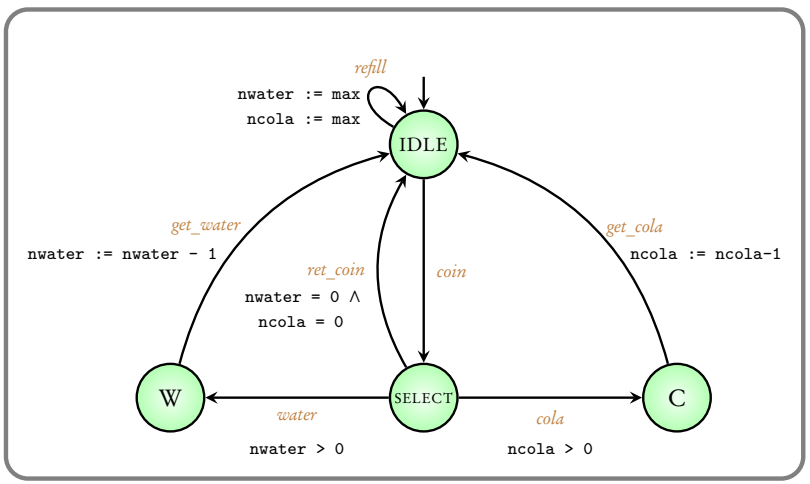
Module 1:
Model-checking tools

Models of code



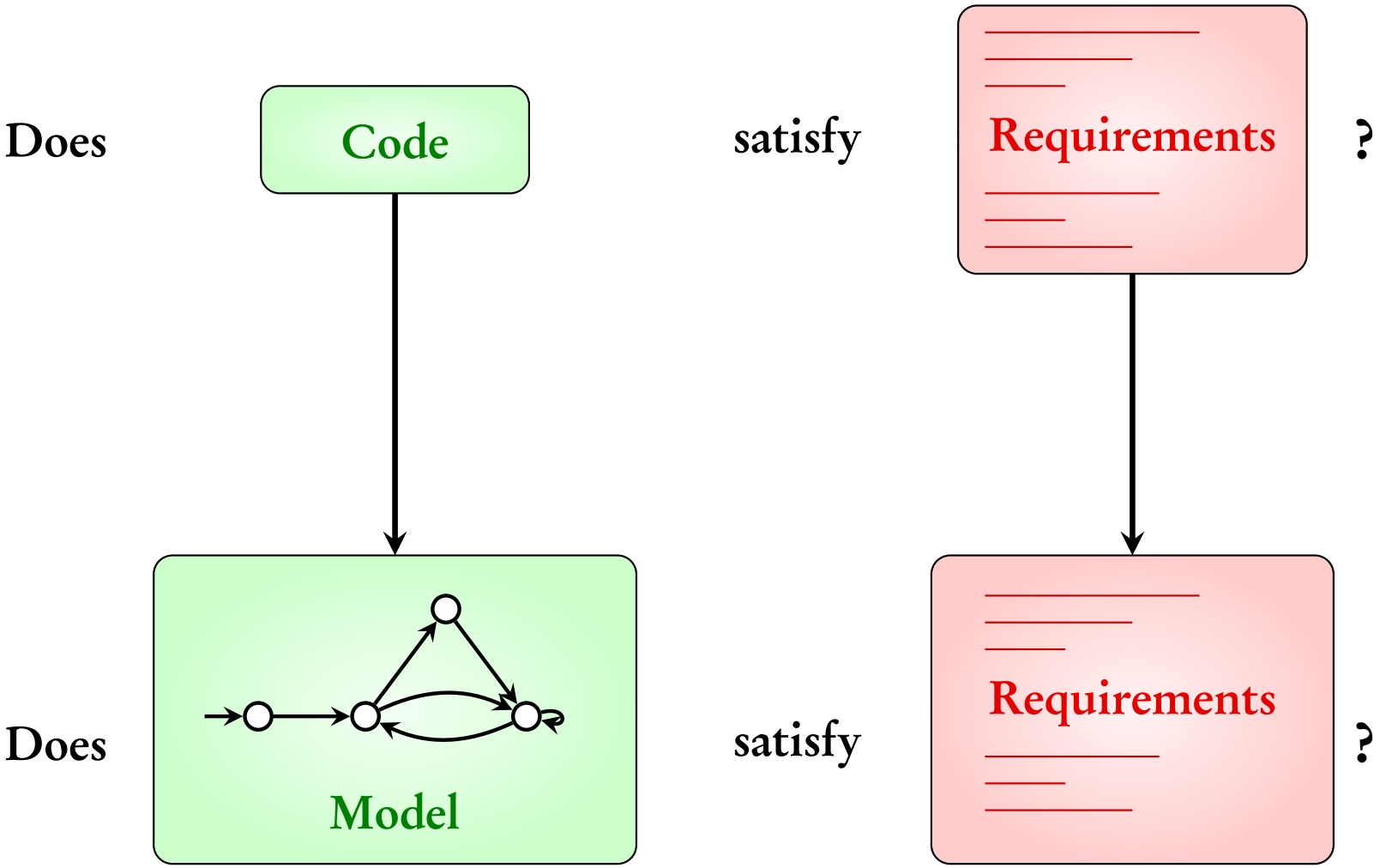


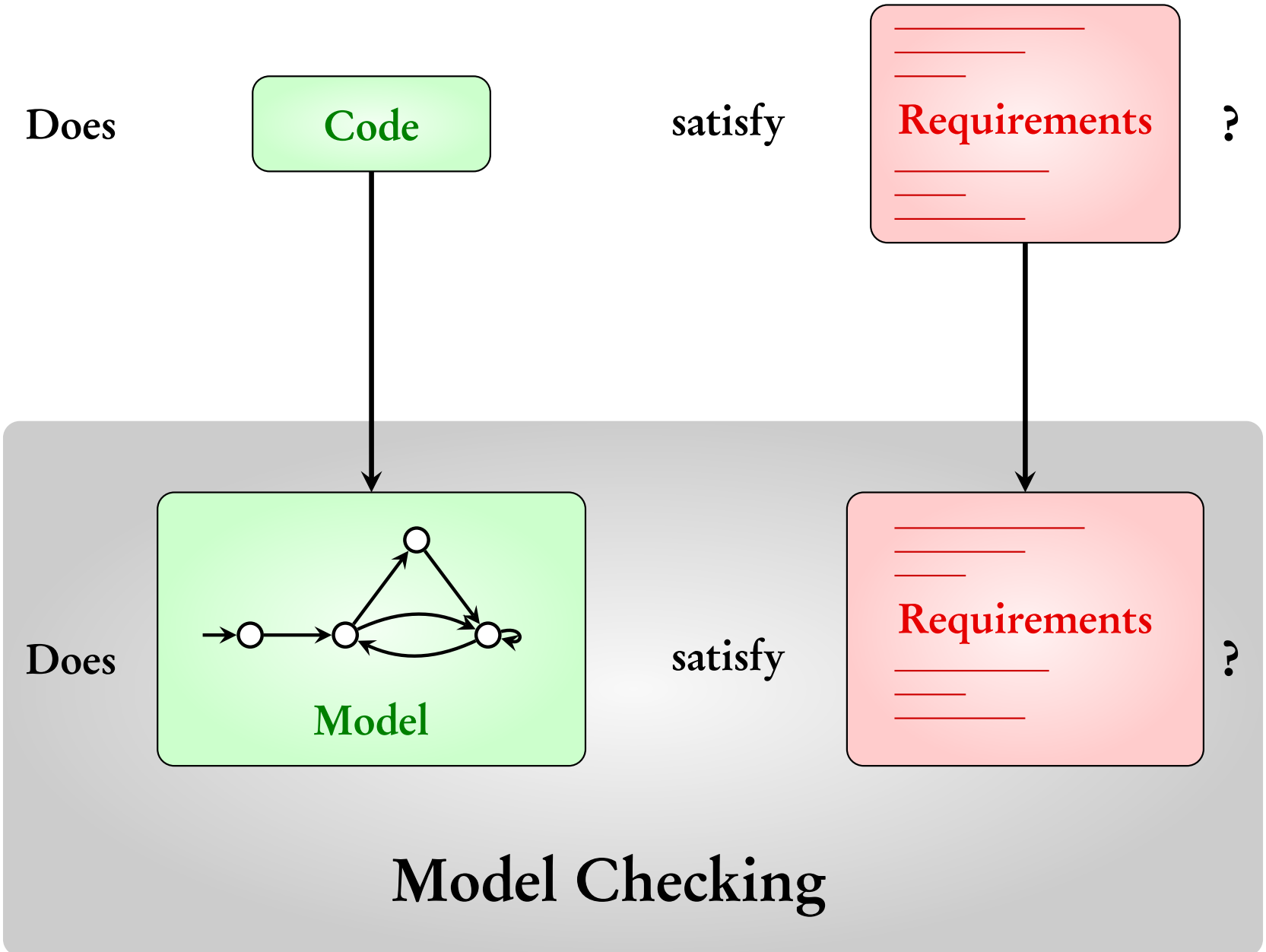




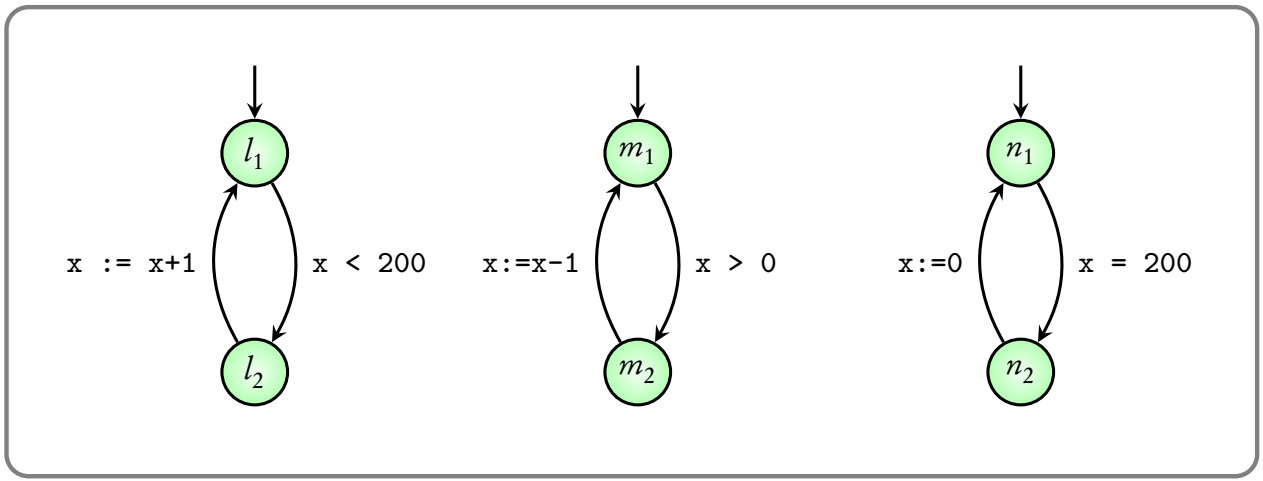
How **reliable** is the code?

Does Code satisfy Requirements ?

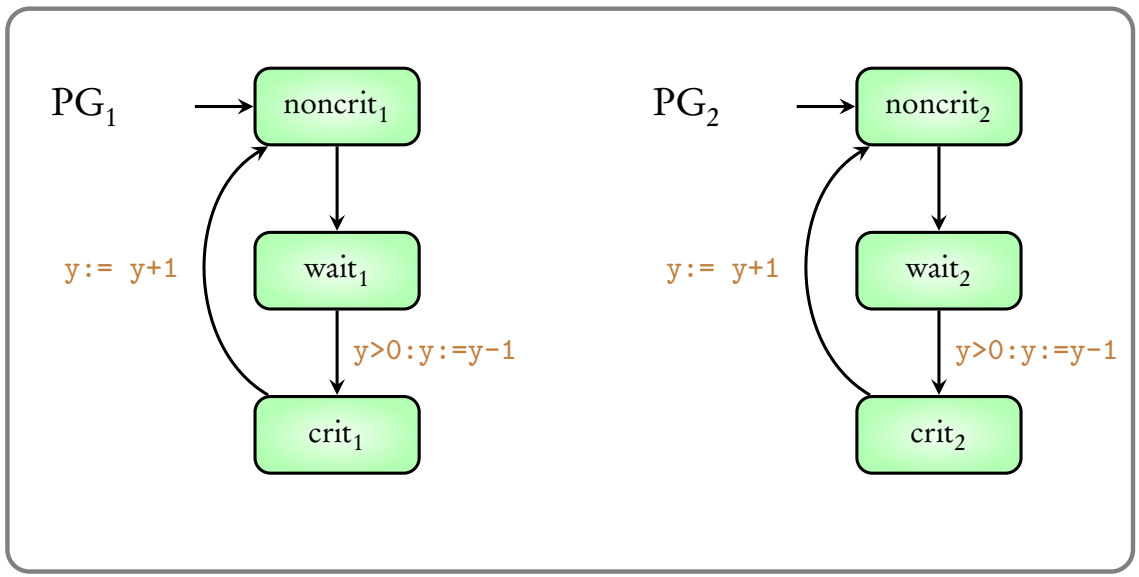

Does
Code
satisfy
Requirements
?
Does
Model
satisfy
Requirements
?


Does
Code
satisfy
Requirements
?
Does
Model
satisfy
Requirements
?
Model Checking

Examples of requirements

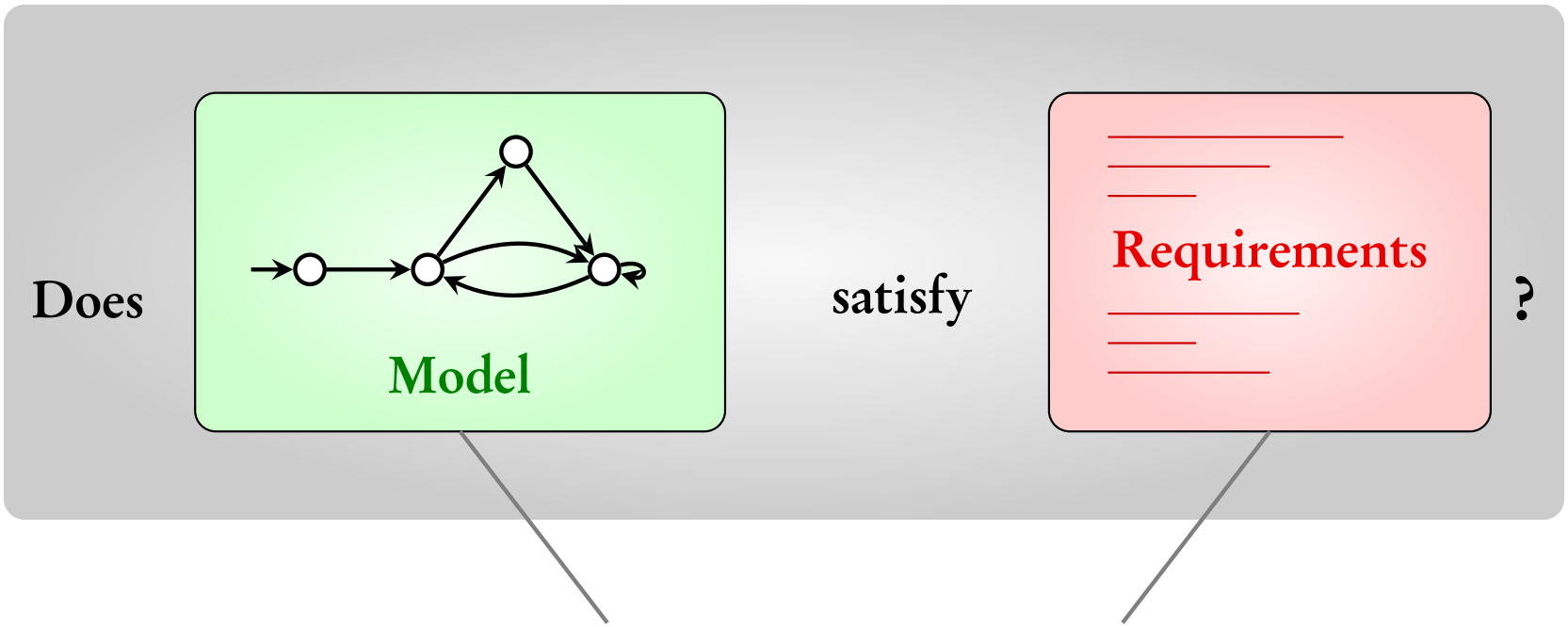


$x \geq 0$ **always**

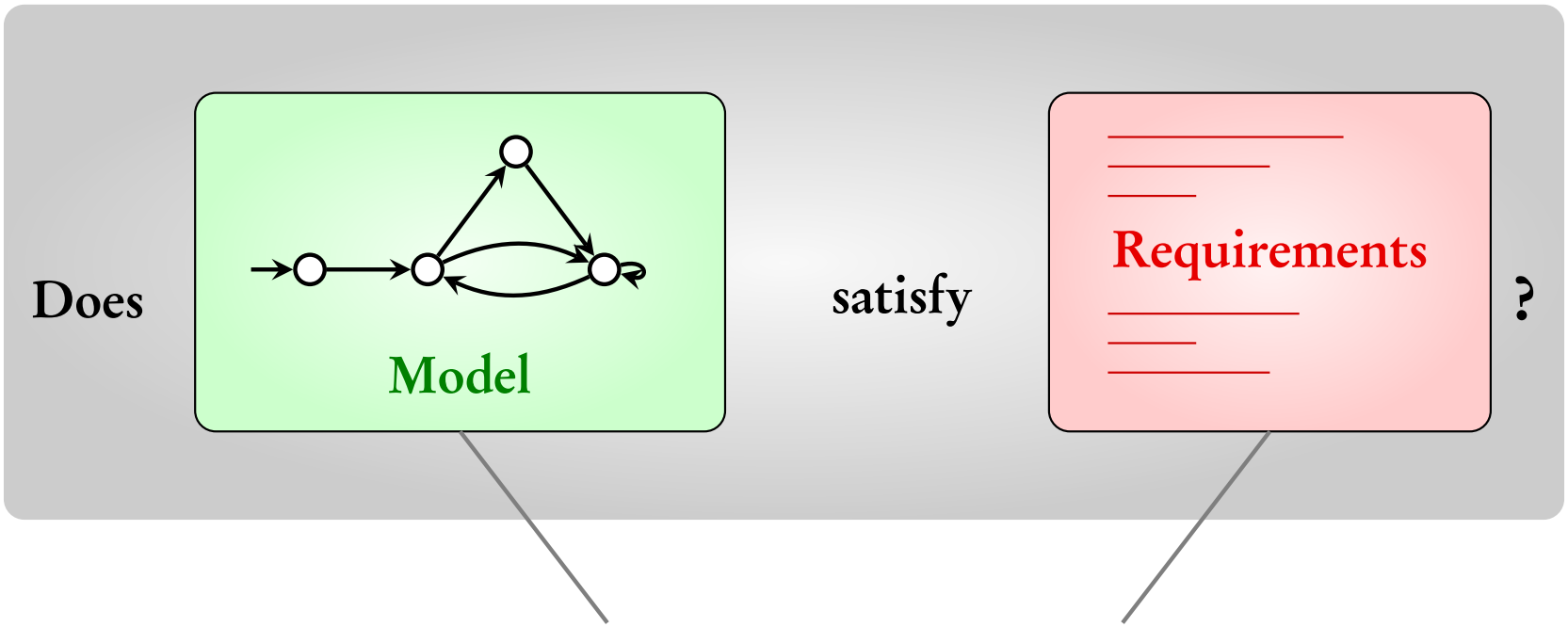


Mutual exclusion property

Model checkers



Model checkers



Model-checkers **automatically** solve the above question

Some open source model-checkers

- **SPIN**
  - Well-suited for **concurrent** systems
- **NuSMV**
  - Well-suited for **hardware** circuits
  - More requirements can be checked compared to SPIN

Some open source model-checkers

- **SPIN**
  - Well-suited for **concurrent** systems
- **NuSMV**
  - Well-suited for **hardware** circuits
  - More requirements can be checked compared to SPIN

In this course: NuSMV

Installing NuSMV

Summary

Model-checkers

Verifying requirements on models

NuSMV

Summary

Model-checkers

Verifying requirements on models

NuSMV

This week: Learn to **use NuSMV** through **examples**

Unit-2: Model-checker NuSMV

B. Srivathsan

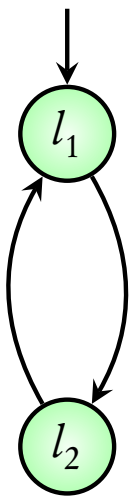
Chennai Mathematical Institute

NPTEL-course

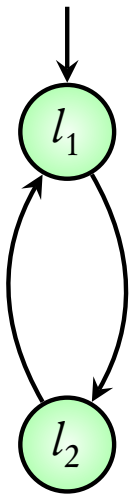
July - November 2015

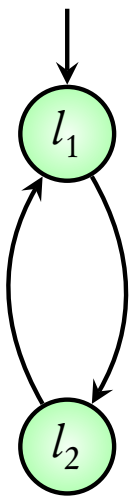
Module 2:

Simple models in NuSMV

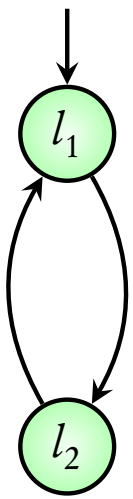

l_1
l_2

MODULE main

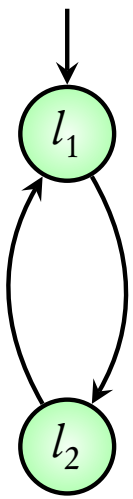




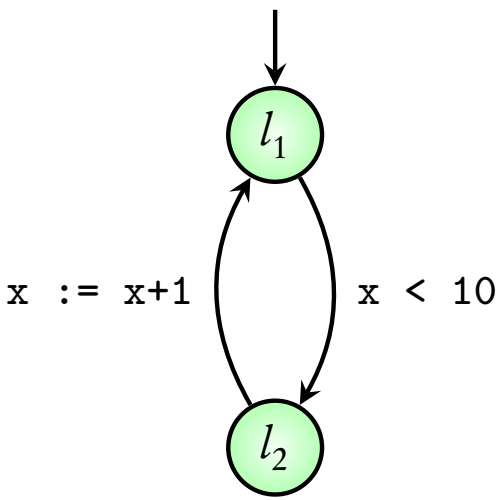

```
MODULE main

VAR

    location:  {l1,l2};
```

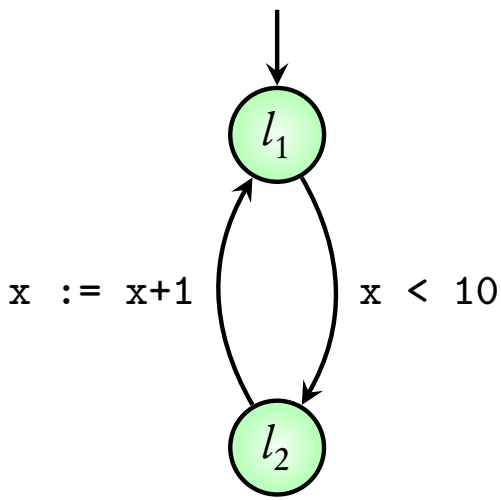



```
MODULE main
VAR
    location:  {l1,l2};
ASSIGN
    init(location) := l1;
```

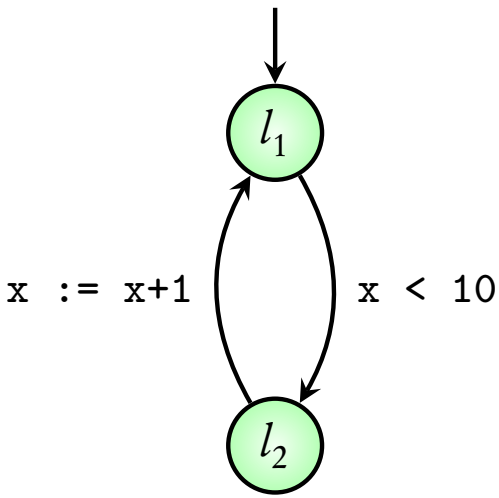



```
MODULE main
VAR
   location:  {l1,l2};
ASSIGN
   init(location) := l1;
   next(location) := case
                (location = l1) :  l2;
                (location = l2) :  l1;
                     esac;
```

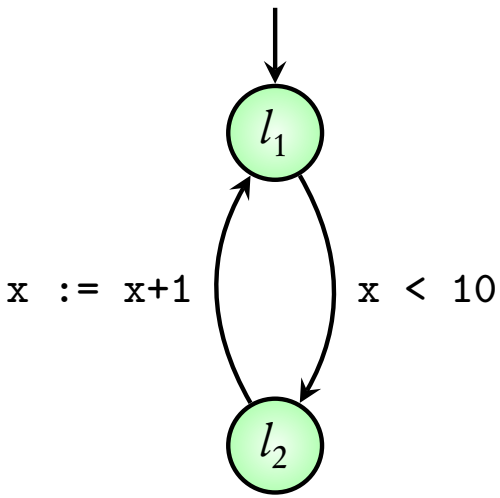

l_1
x := x+1
x < 10
l_2



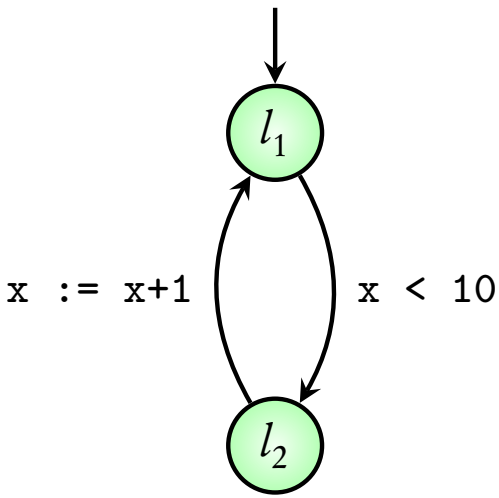

```
MODULE main
VAR
    location:  {l1,l2};

ASSIGN
    init(location) := l1;

    next(location) := case
                  (location = l1):  l2;
                  (location = l2) :  l1;

                       esac;
```



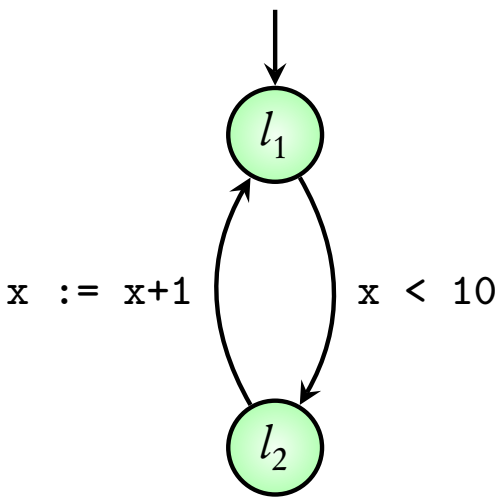

```
MODULE main
VAR
    location:  {l1,l2};
    x:  0 ..  100;
ASSIGN
    init(location) := l1;

    next(location) := case
                  (location = l1):  l2;
                  (location = l2) :  l1;

                       esac;
```



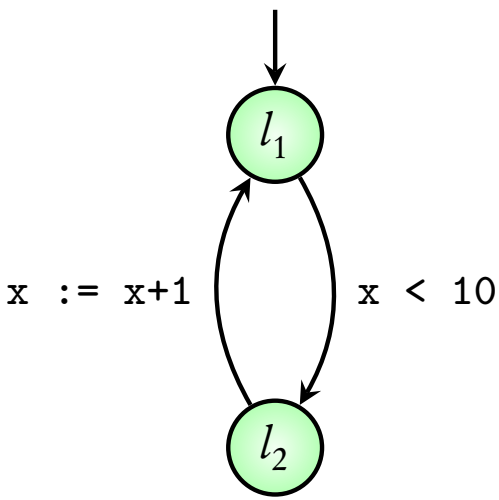

```
MODULE main
VAR
    location:  {l1,l2};
    x:  0 ..  100;
ASSIGN
    init(location) := l1;
    init(x) := 0;
    next(location) := case
                  (location = l1):  l2;
                  (location = l2) :  l1;

                      esac;
```




```
MODULE main
VAR
    location:  {l1,l2};
    x:  0 ..  100;
ASSIGN
    init(location) := l1;
    init(x) := 0;
    next(location) := case
                  (location = l1) & (x<10):  l2;
                  (location = l2) :  l1;

                       esac;
```

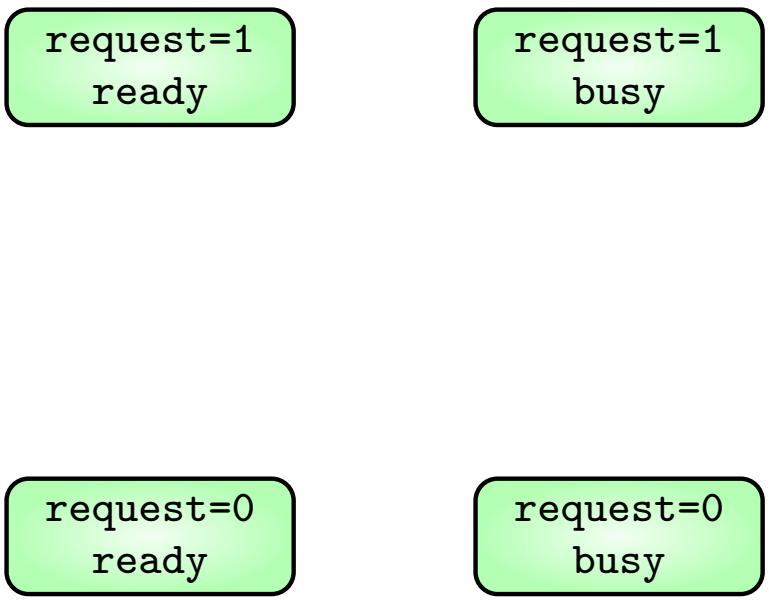



```
MODULE main
VAR
    location:  {l1,l2};
    x:  0 ..  100;
ASSIGN
    init(location) := l1;
    init(x) := 0;
    next(location) := case
                  (location = l1) & (x<10):  l2;
                  (location = l2) :  l1;
                  TRUE: location;
                       esac;
```

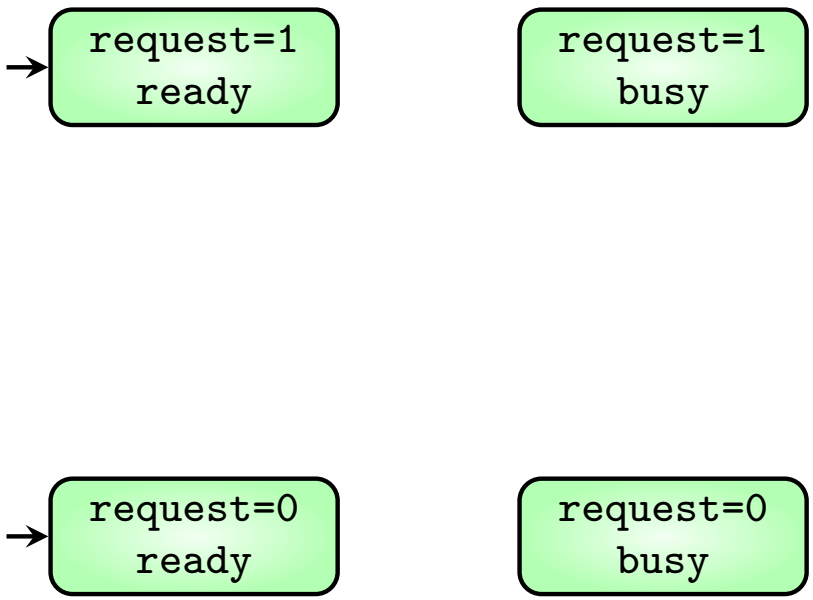



```
MODULE main
VAR
    location:  {l1,l2};
    x:  0 ..  100;
ASSIGN
    init(location) := l1;
    init(x) := 0;
    next(location) := case
                (location = l1) & (x<10):  l2;
                (location = l2) :  l1;
                TRUE: location;
                     esac;
    next(x) := case
                (location = l2) & x < 100:  x+1;
                TRUE: x;
              esac;
```

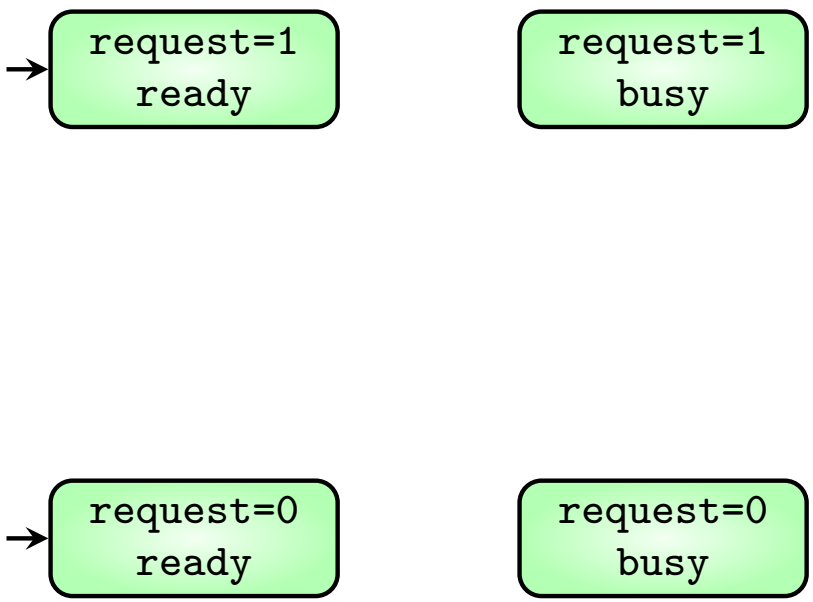
```
MODULE main
VAR
    request:  boolean;
    status:  {ready, busy}
```



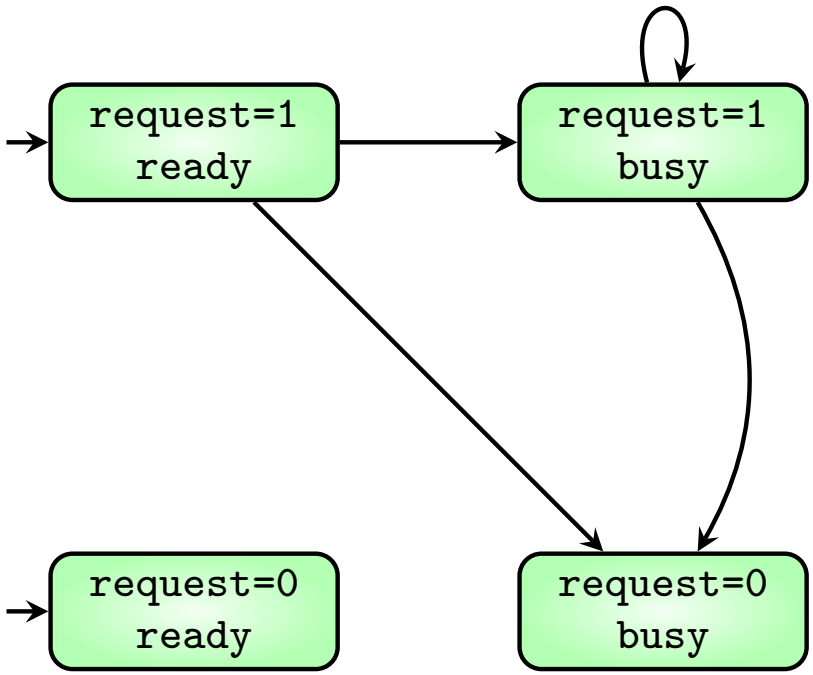

```
MODULE main
VAR
    request:  boolean;
    status:  {ready, busy}
```



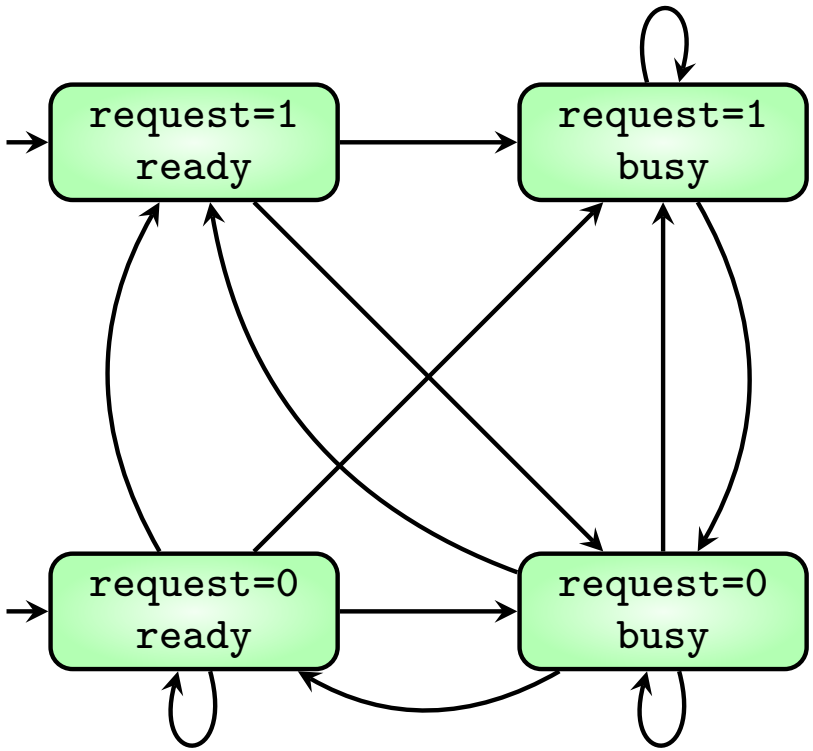

```
MODULE main
VAR
    request:  boolean;
    status:  {ready, busy}
ASSIGN
    init(status) := ready;
```




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

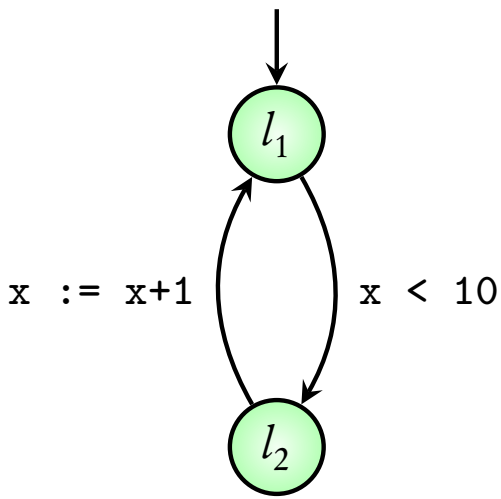



```
MODULE main
VAR
    request:  boolean;
    status:  {ready, busy}
ASSIGN
    init(status) := ready;
    next(status) := case
                    request :  busy;
                    TRUE : {ready,busy};
                          esac;
```

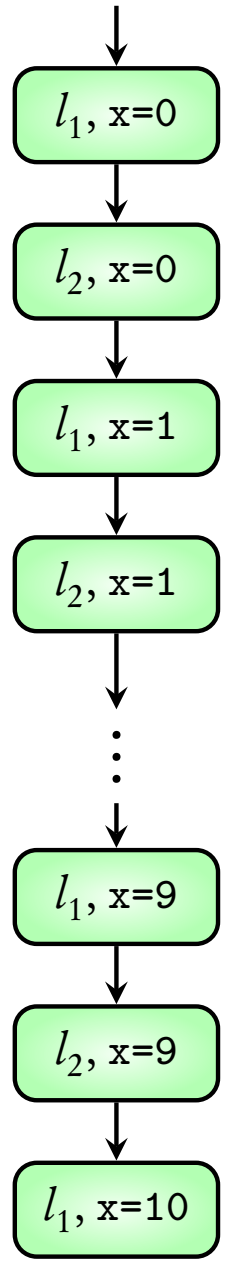



```
MODULE main
VAR
    request:  boolean;
    status:  {ready, busy}
ASSIGN
    init(status) := ready;
    next(status) := case
                    request :  busy;
                    TRUE : {ready,busy};
                          esac;
```

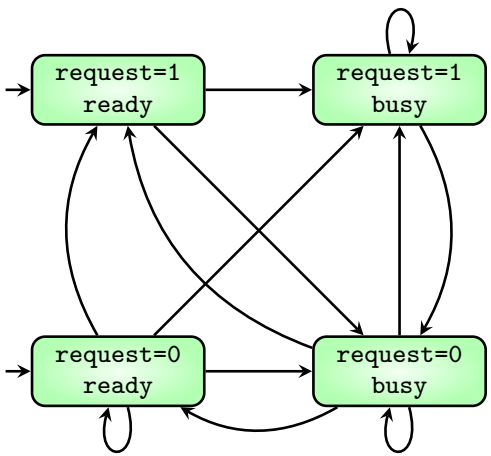
Coming next: checking **requirements** in NuSMV

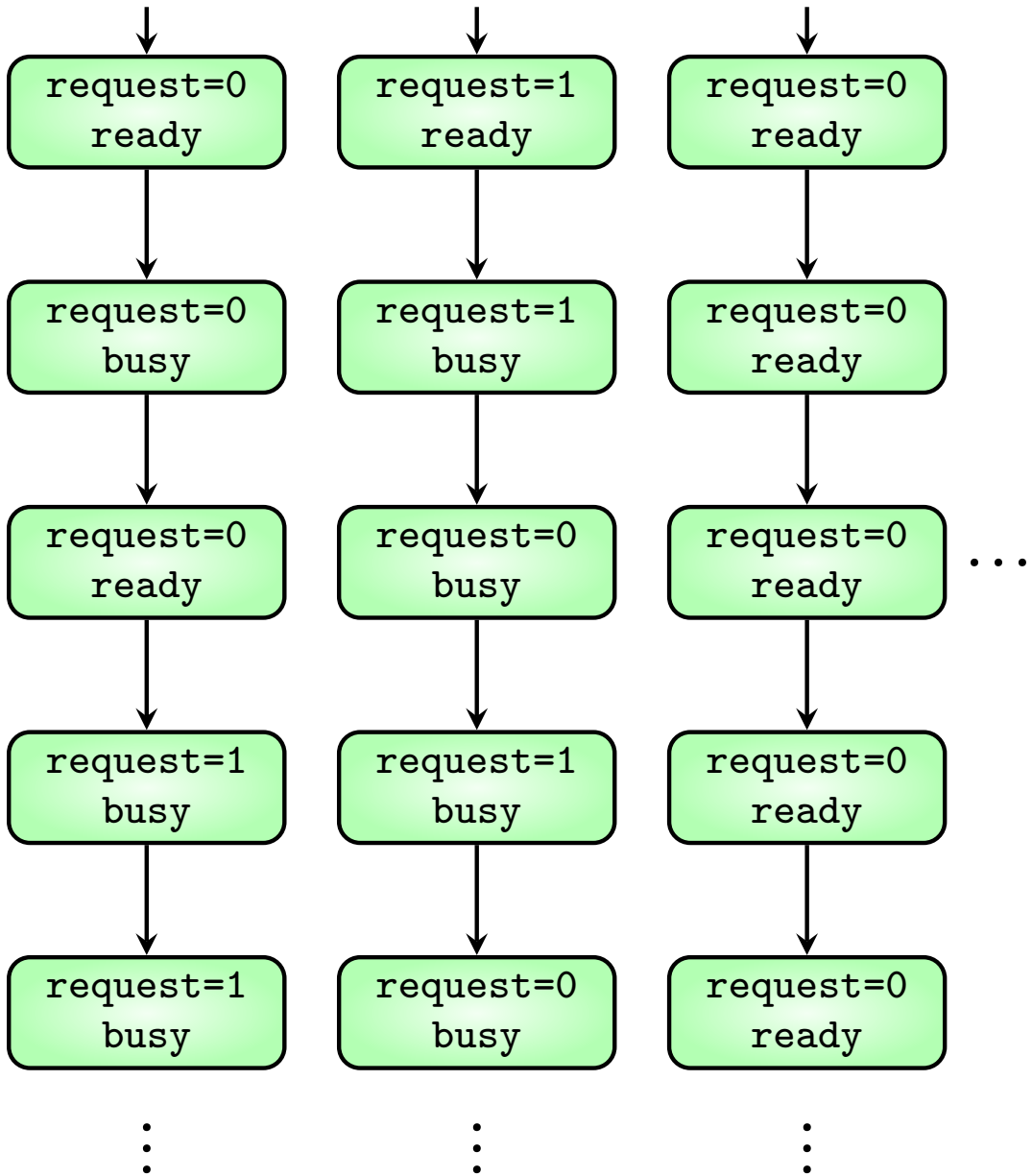


Executions



Executions


request=1
ready
request=1
busy
request=0
ready
request=0
busy


request=0
ready
request=0
busy
request=0
ready
request=1
busy
request=1
busy
request=0
busy
request=1
ready
request=1
busy
request=0
busy
request=0
ready
request=0
ready
request=0
ready
request=0
ready
request=0
ready

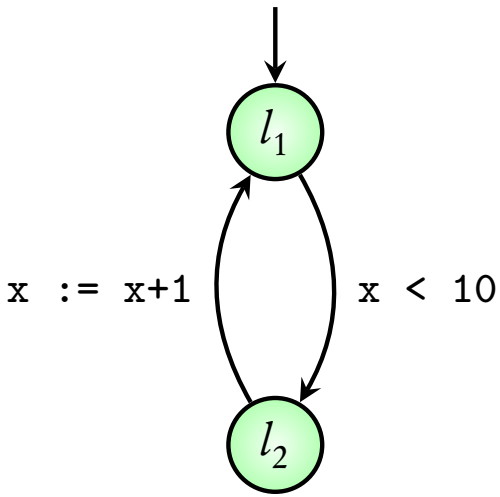
Transition system **satisfies a requirement**

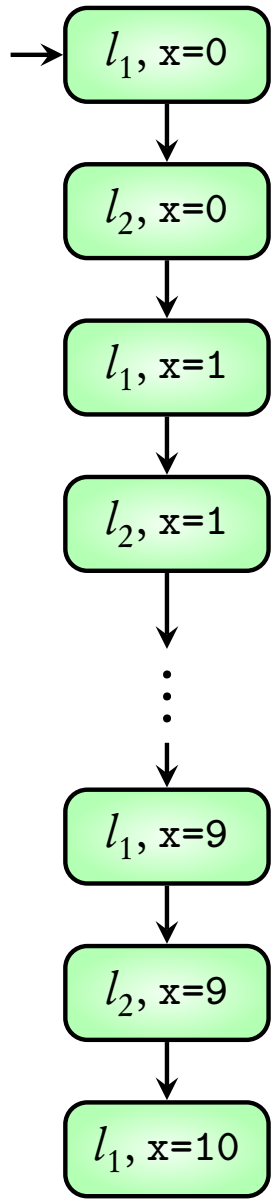
means

all its executions satisfy the requirement

Requirement type 1: G

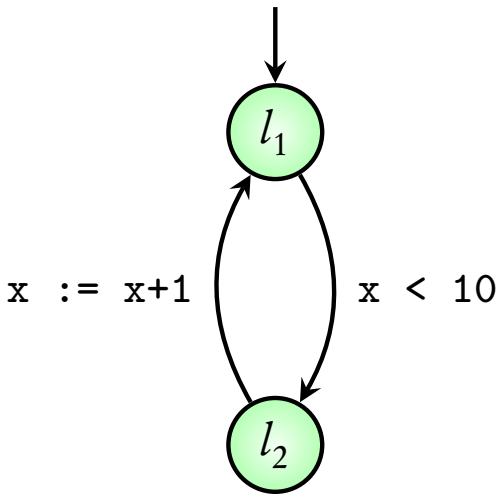
Requirement type 1: G





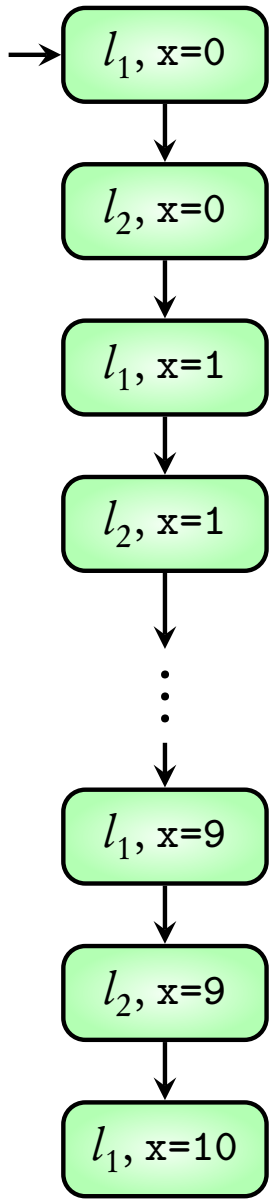
G (x >= 0)

Requirement type 1: G



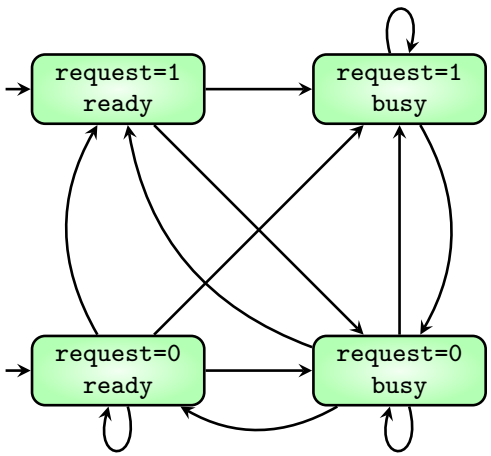
TS of above PG with initial value x=0

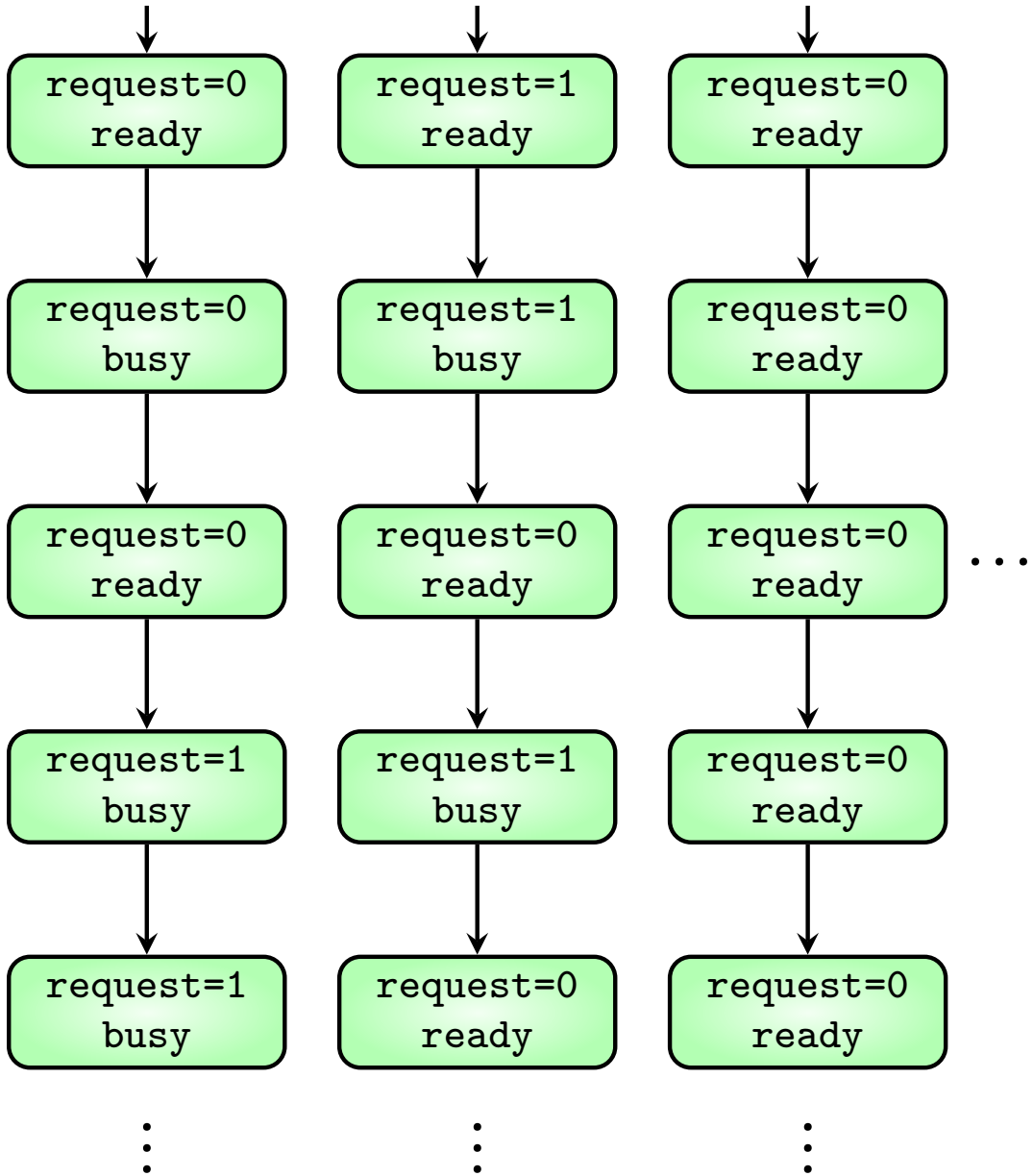
satisfies G (x >= 0)



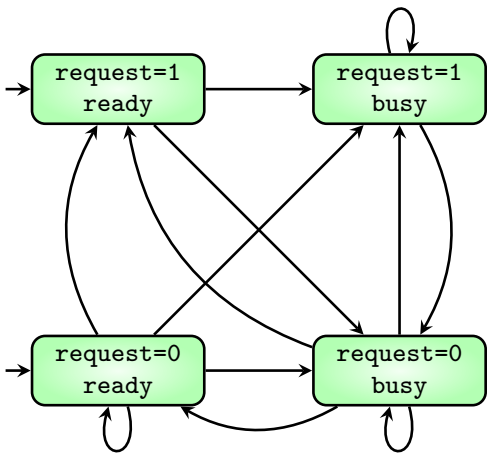
G (x >= 0)

G (request=0)

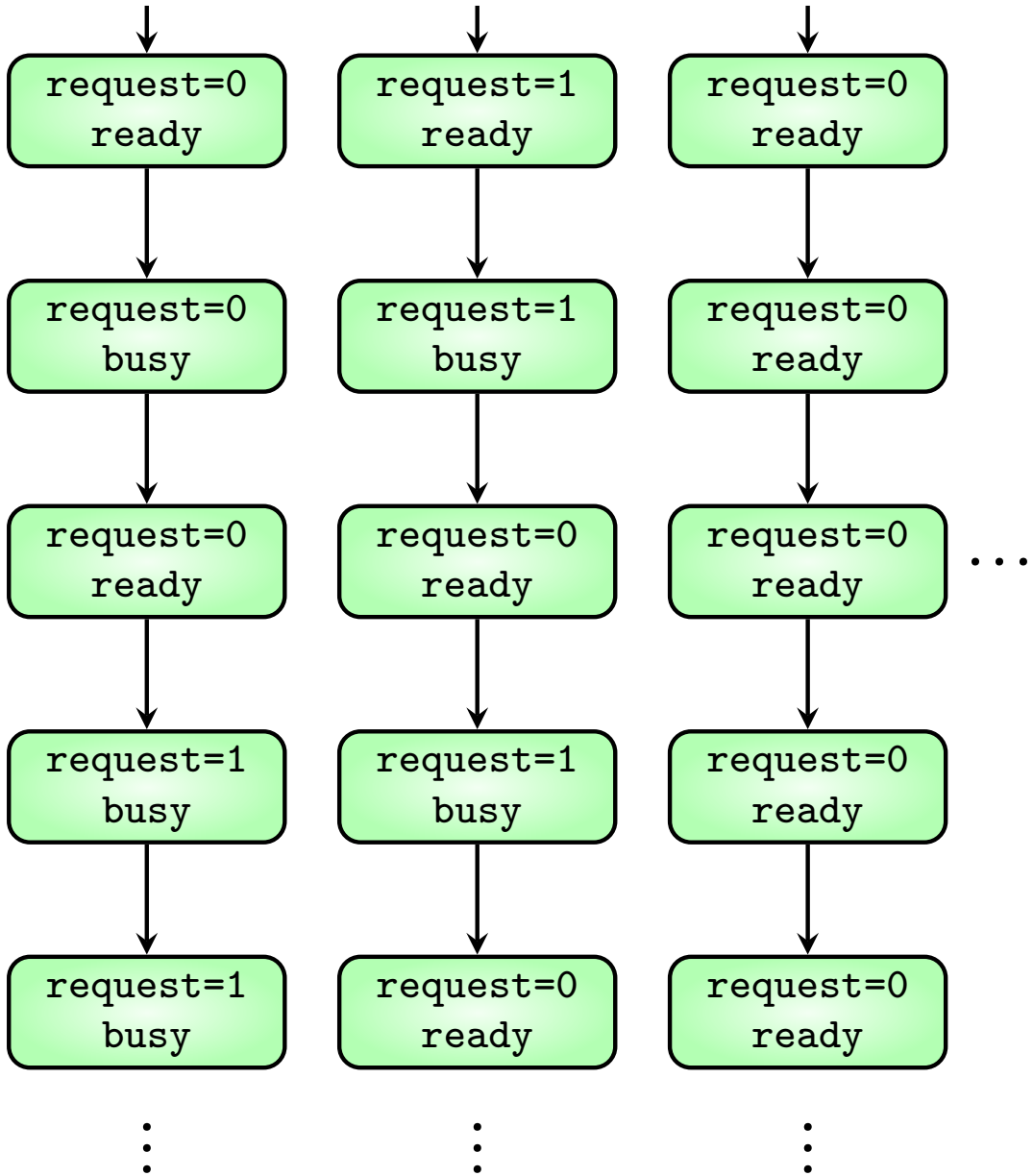

request=1
ready
request=1
busy
request=0
ready
request=0
busy


request=0
ready
request=0
busy
request=0
ready
request=1
busy
request=1
busy
request=1
busy
request=0
ready
request=0
ready
request=0
ready
request=0
ready
request=0
ready
request=0
ready
...

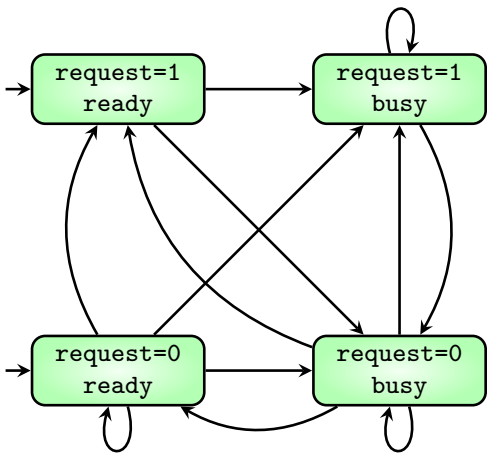
G (request=0)

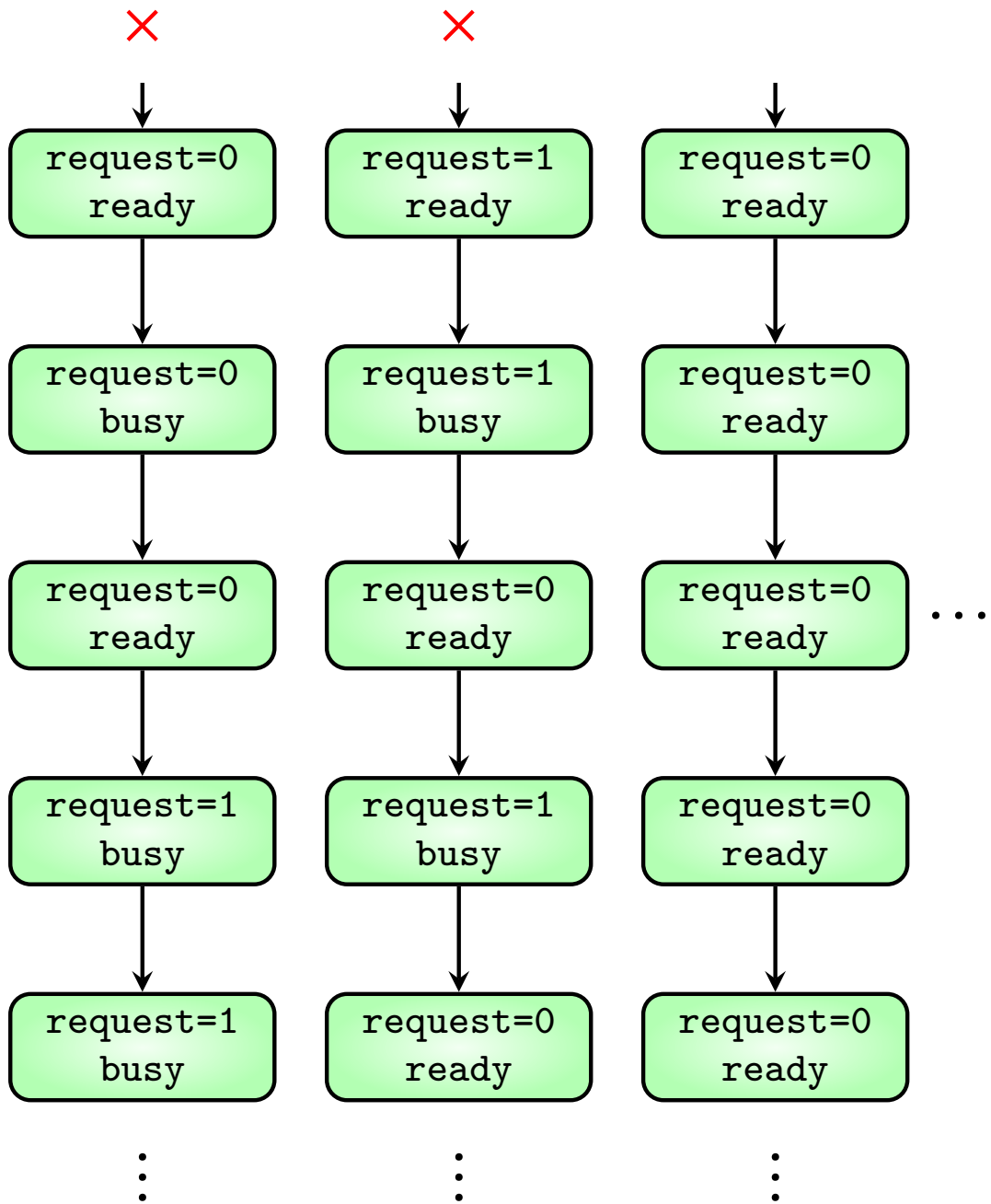

request=1
ready
request=1
busy
request=0
ready
request=0
busy

×

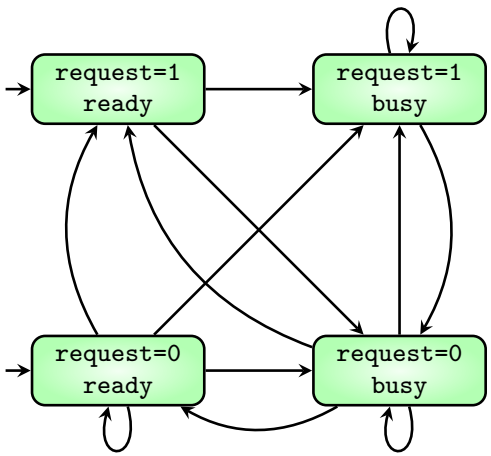

request=0
ready
request=0
busy
request=0
ready
request=1
busy
request=1
busy
request=1
ready
request=1
busy
request=0
ready
request=0
ready
request=0
ready
request=0
ready
request=0
ready
request=0
ready

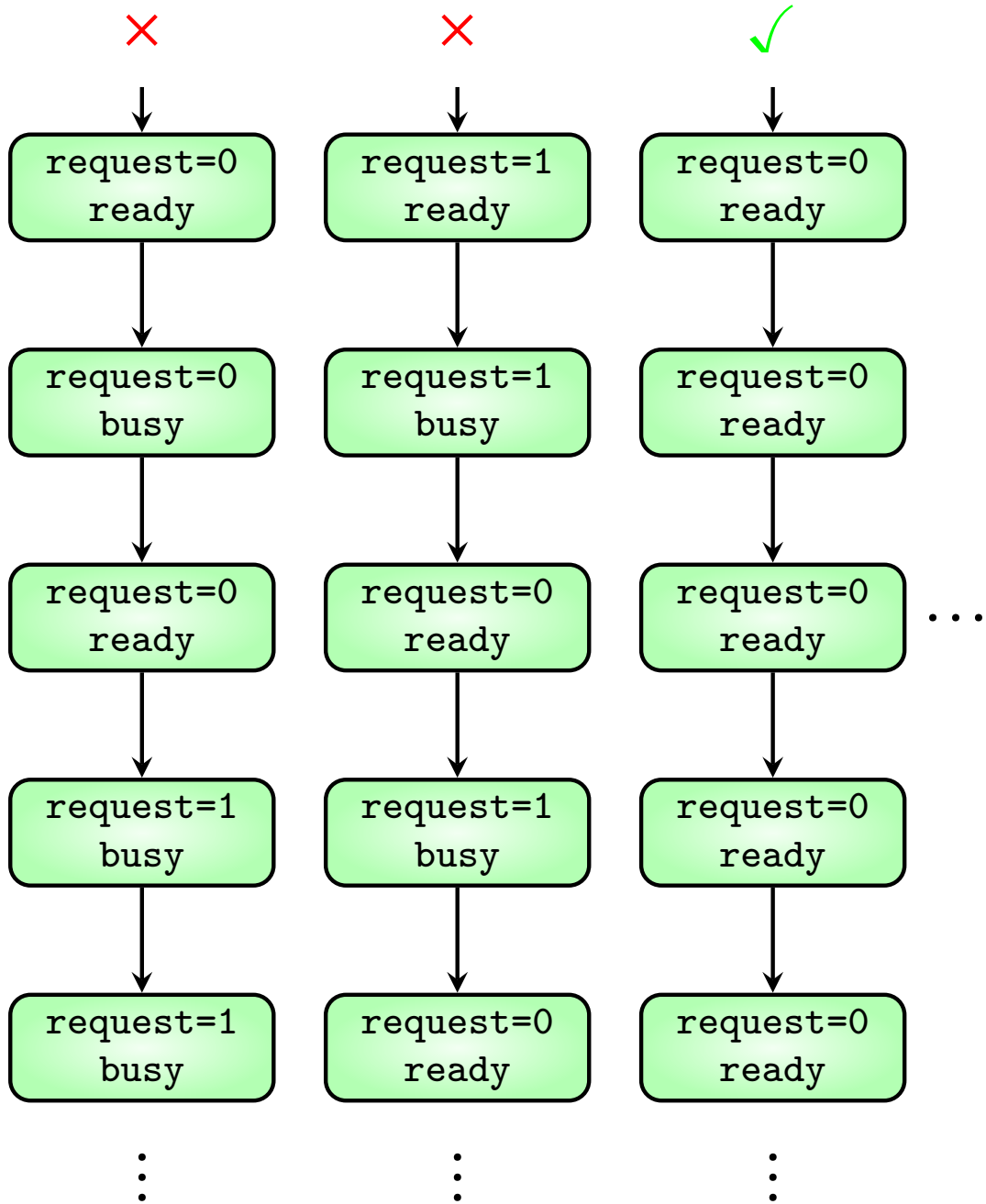
G (request=0)


request=1
ready
request=1
busy
request=0
ready
request=0
busy

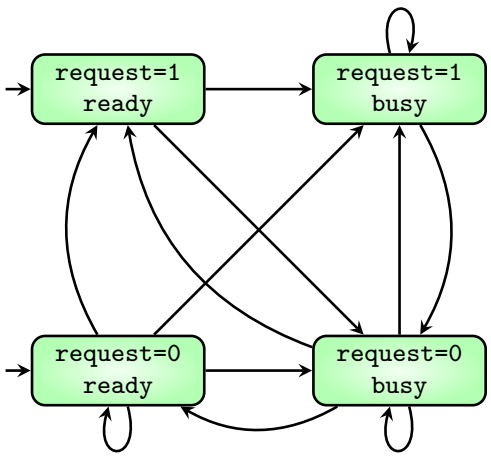

request=0
ready
request=0
busy
request=0
ready
request=1
busy
request=1
busy
request=1
ready
request=1
busy
request=0
ready
request=0
ready
request=0
ready
request=0
ready
request=0
ready
request=0
ready

G (request=0)


request=1
ready
request=1
busy
request=0
ready
request=0
busy

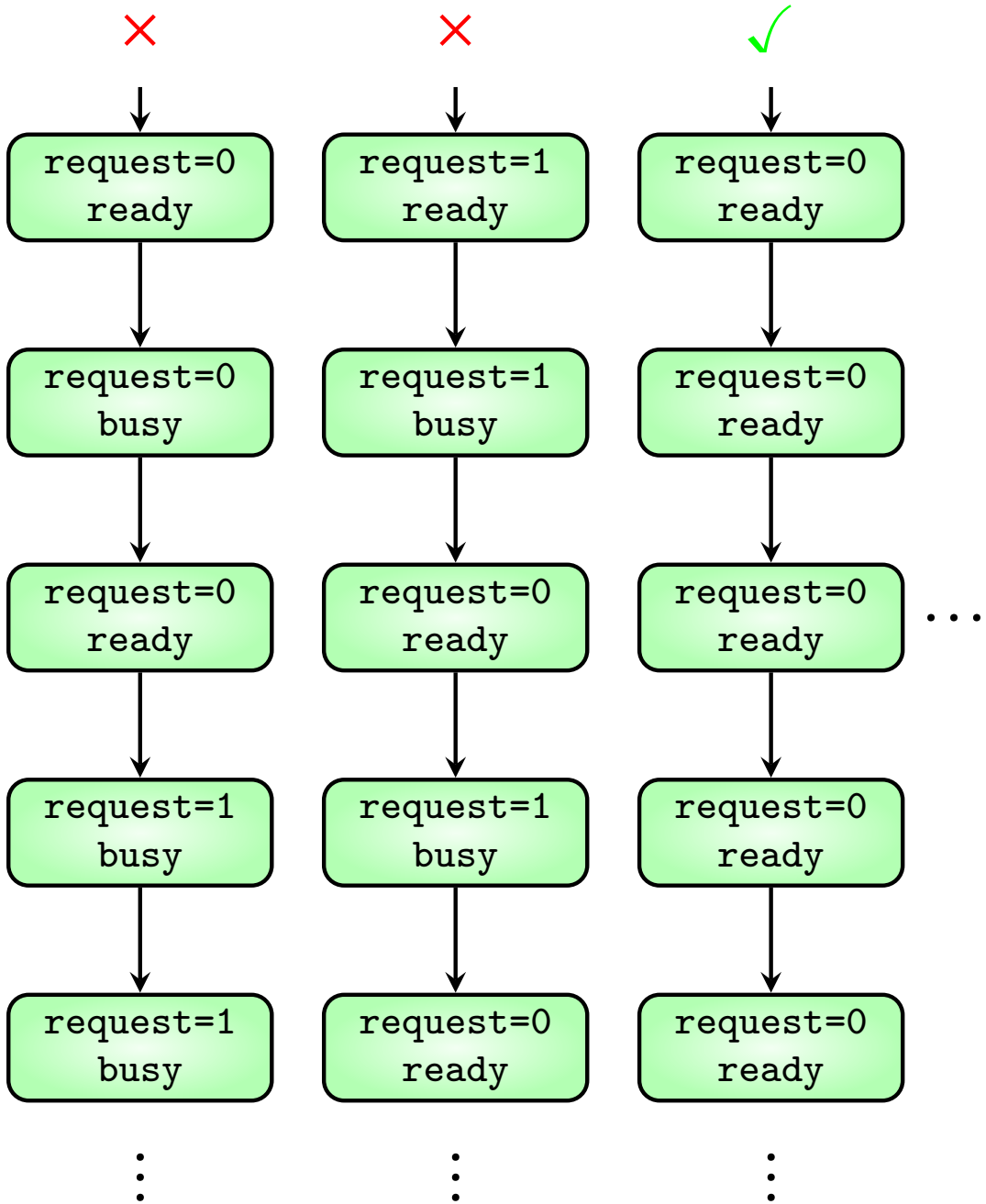

request=0
ready
request=0
busy
request=0
ready
request=1
busy
request=1
busy
request=1
busy
request=1
busy
request=0
ready
request=0
ready
request=0
ready
request=0
ready
request=0
ready
request=0
ready

$G\,(\texttt{request=0})$



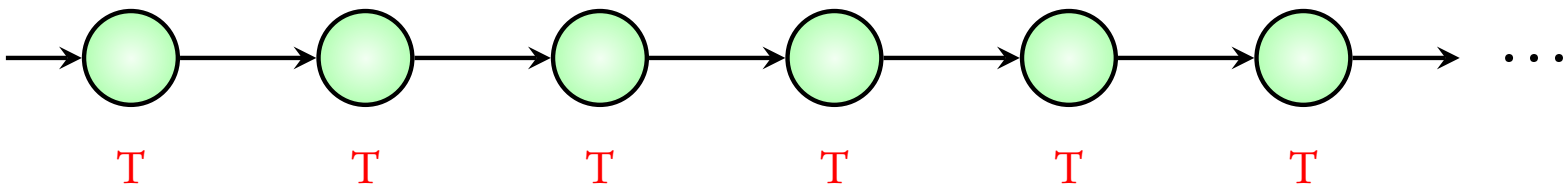
TS does not satisfy

$G\,(\texttt{request=0})$



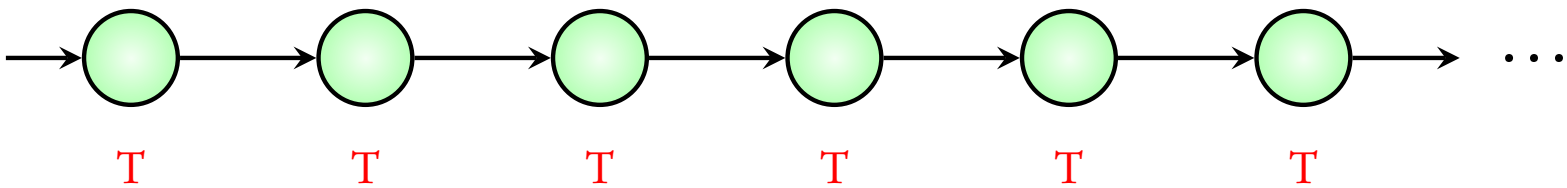
Execution **satisfies** G `(expr)` if

`expr` evaluates to T in **all its states**



Execution **satisfies** G `(expr)` if

`expr` evaluates to T in **all its states**



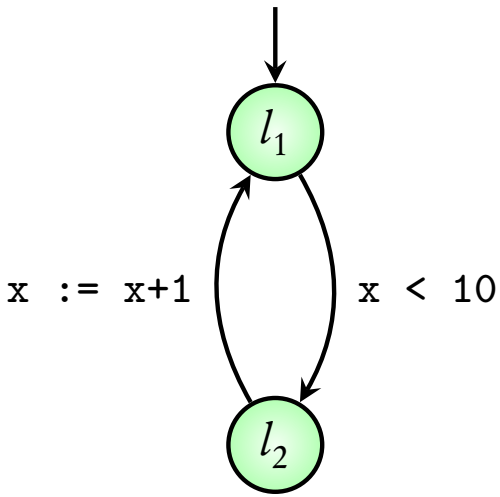
Transition system **satisfies** G `(expr)` if

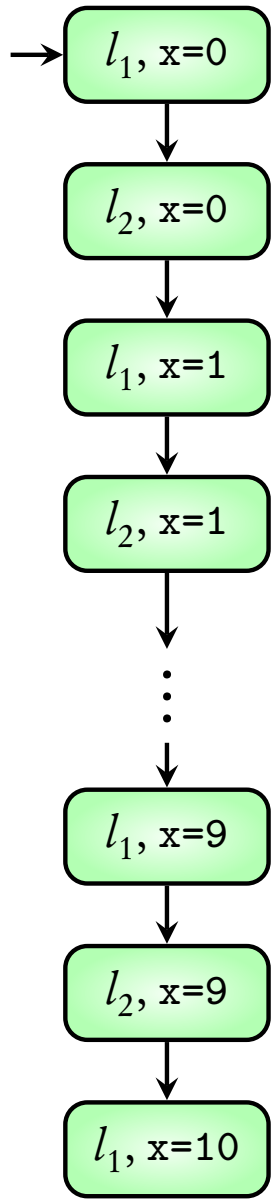
all its executions satisfy G `(expr)`

Checking the G requirement: NuSMV demo

Requirement type 2: F

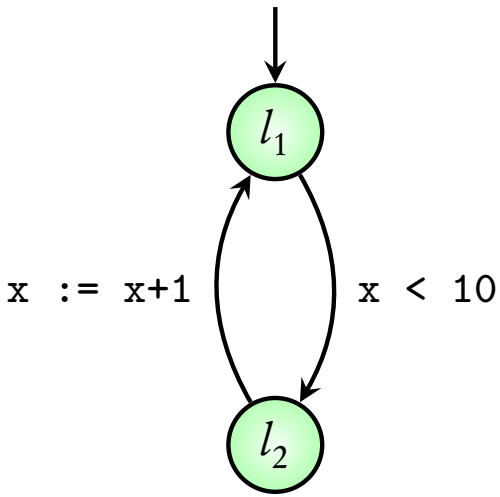
Requirement type 2: F





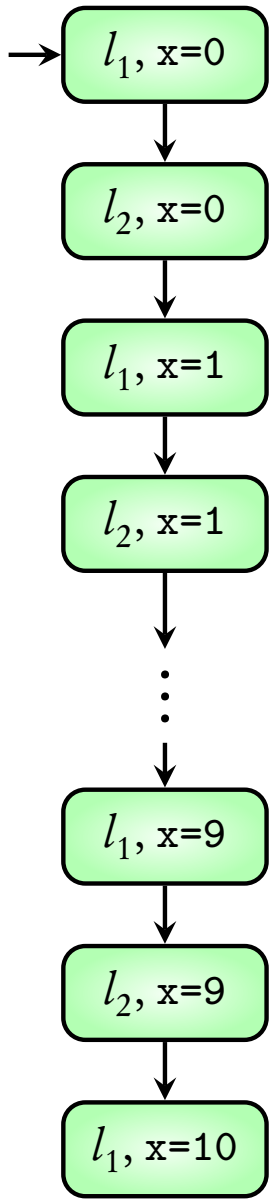
F (x >= 5)

Requirement type 2: F



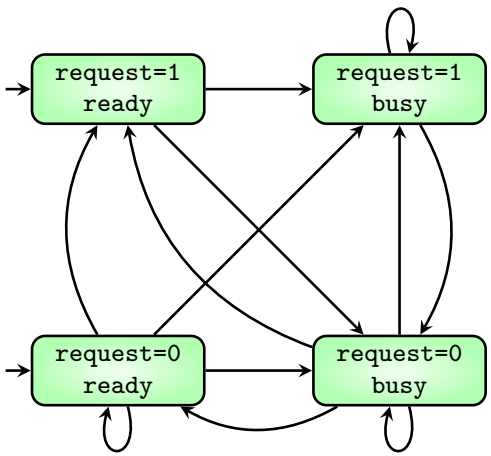
TS of above PG with initial value x=0

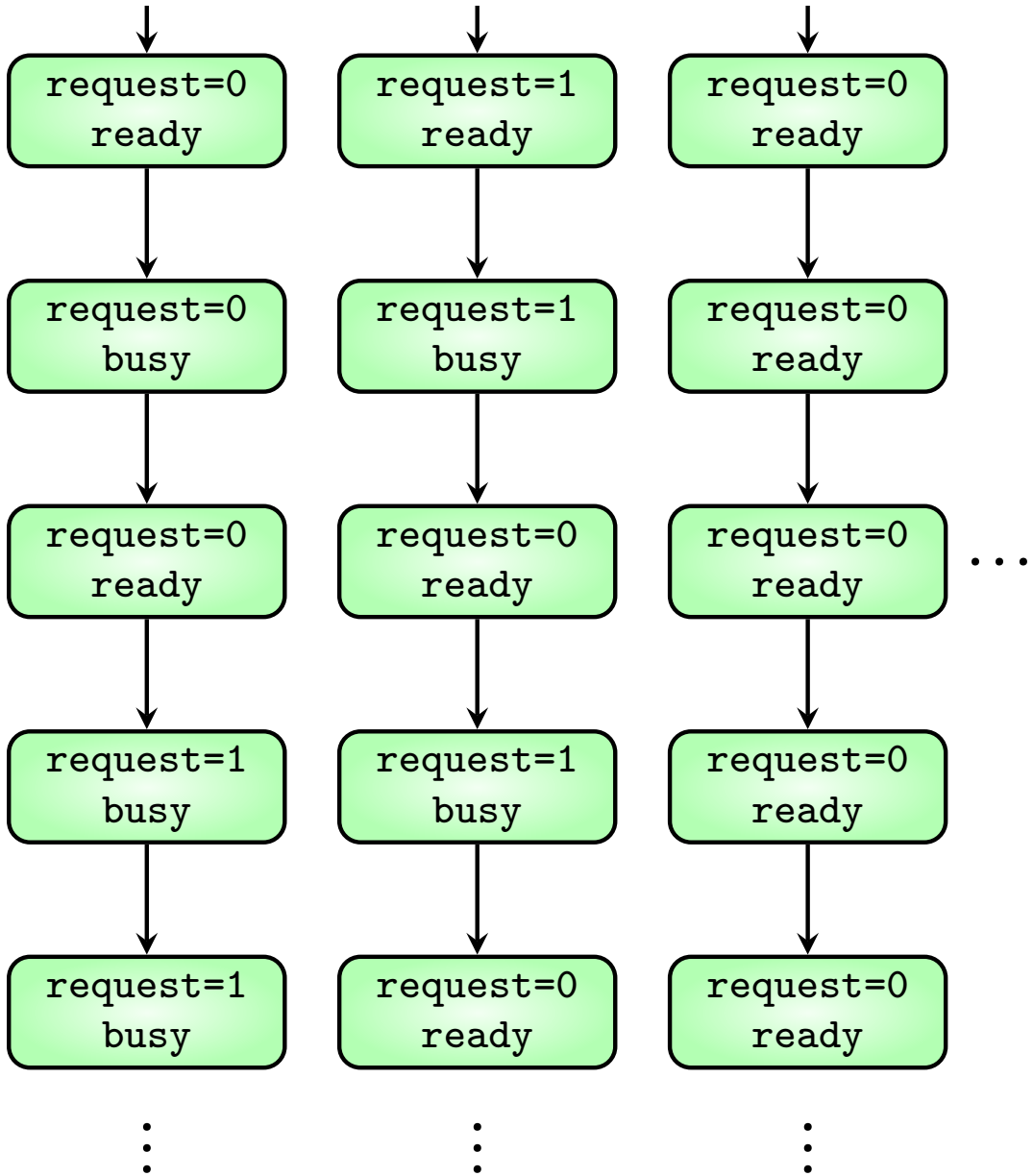
satisfies F (x >= 5)



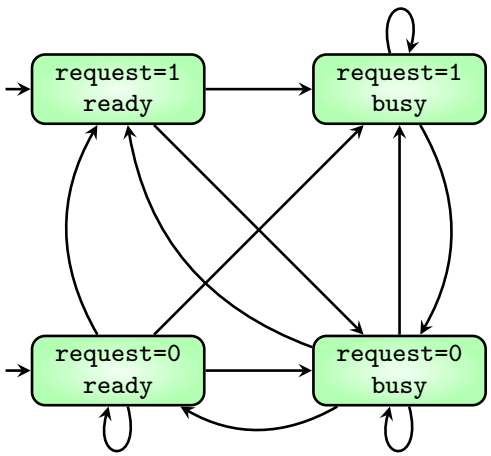
F (x >= 5)

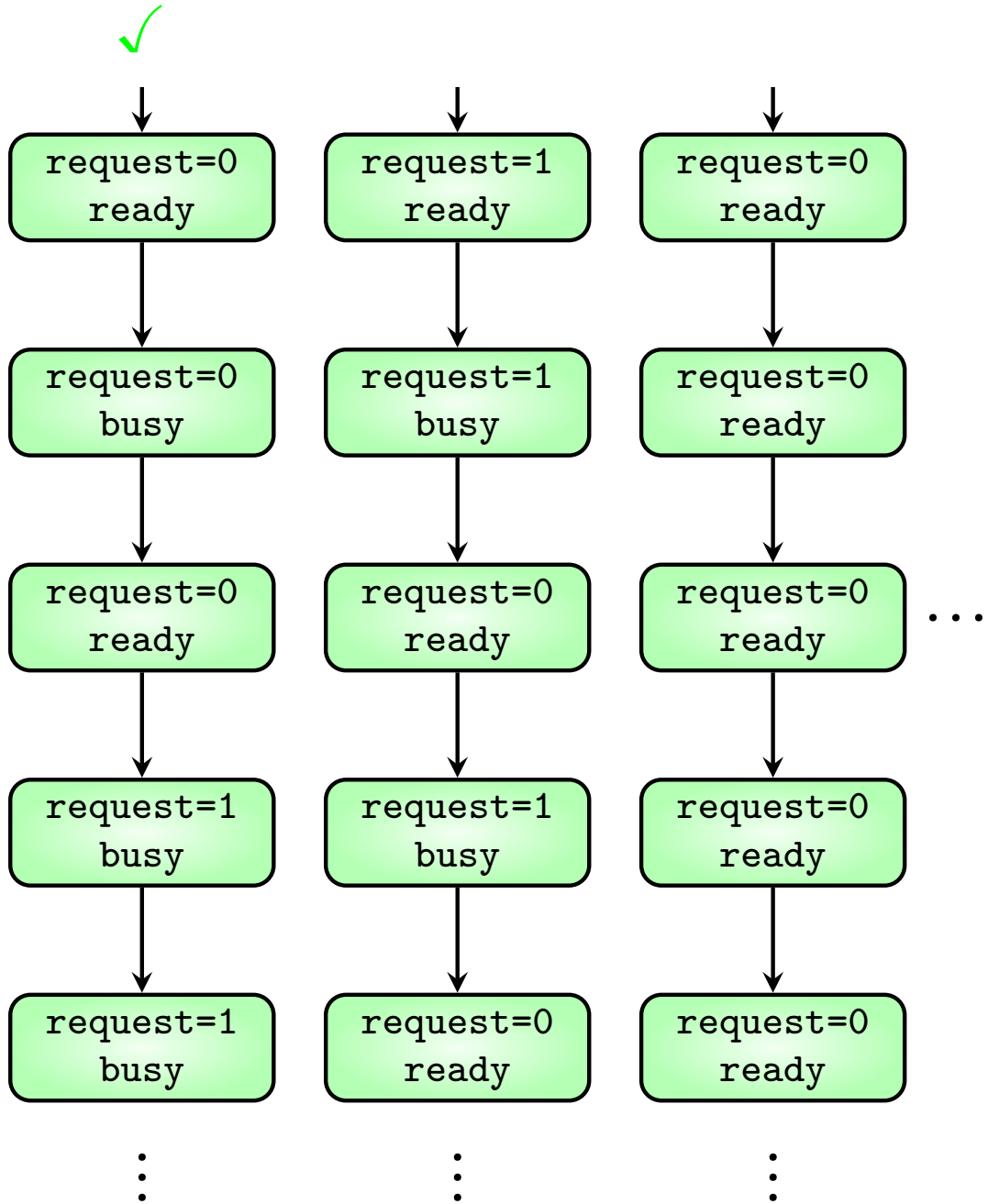
F (request=1)


request=1
ready
request=1
busy
request=0
ready
request=0
busy

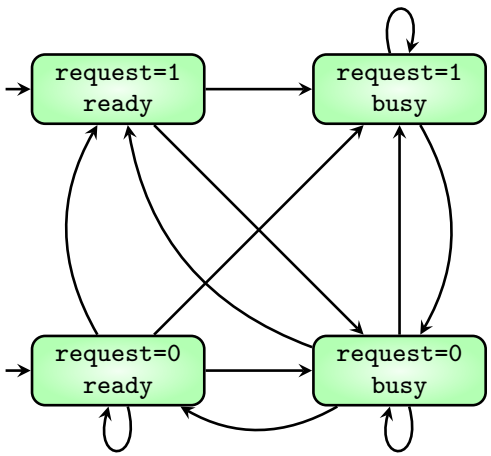

request=0
ready
request=0
busy
request=0
ready
request=1
busy
request=1
busy
request=1
busy
request=1
busy
request=0
ready
request=0
ready
request=0
ready
request=0
ready
request=0
ready
request=0
ready
...

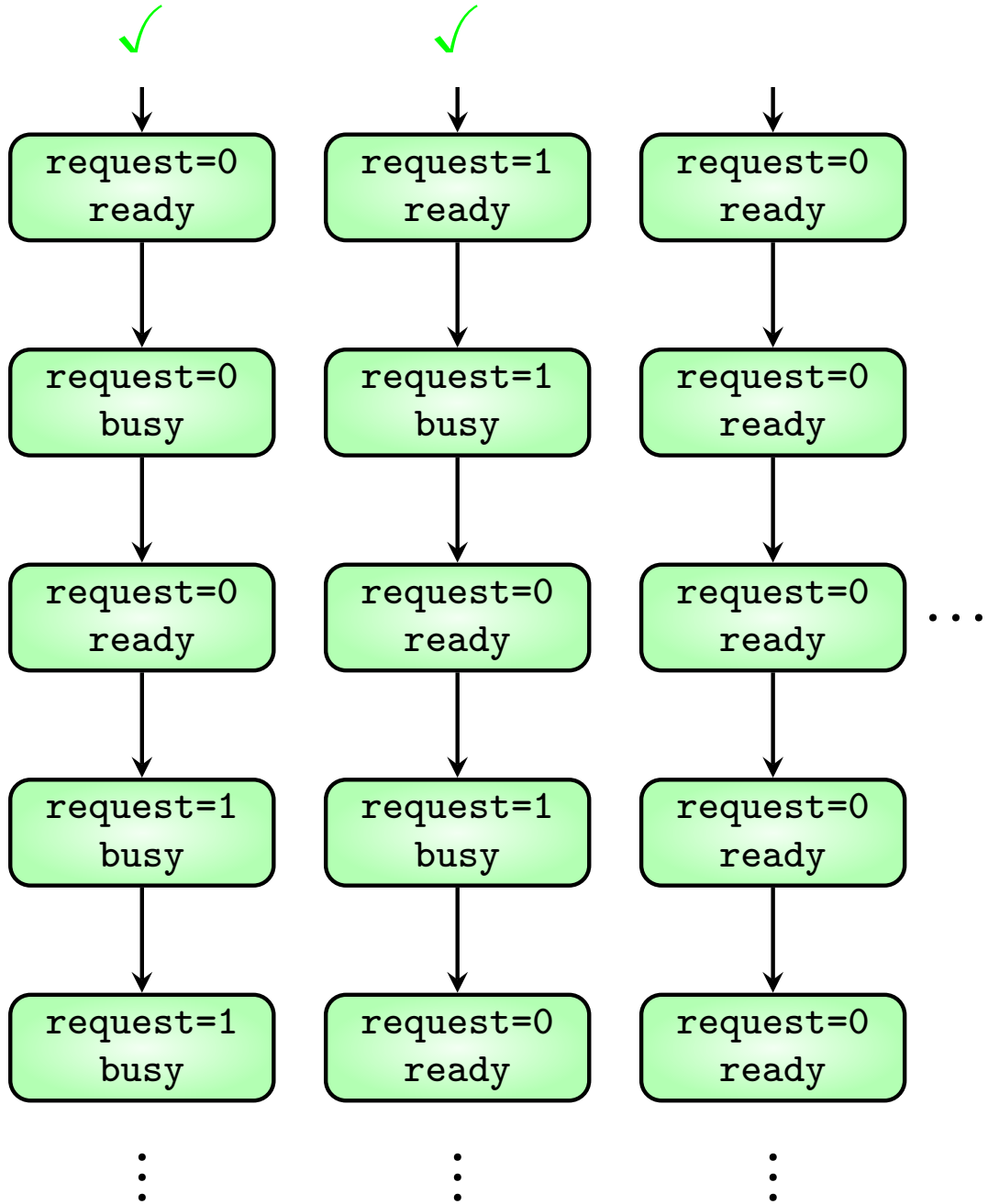
F (request=1)


request=1
ready
request=1
busy
request=0
ready
request=0
busy

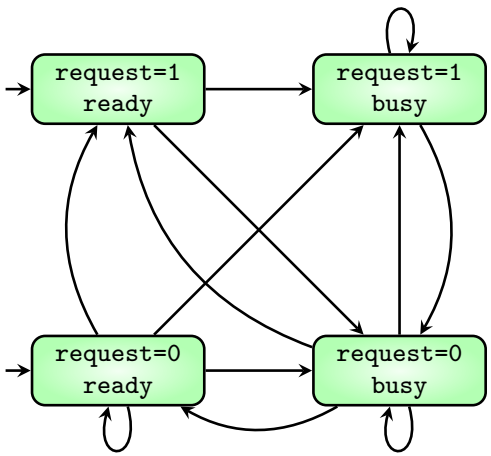

request=0
ready
request=0
busy
request=0
ready
request=1
busy
request=1
busy
request=1
busy
request=1
ready
request=1
busy
request=0
ready
request=0
ready
request=0
ready
request=0
ready
request=0
ready
request=0
ready

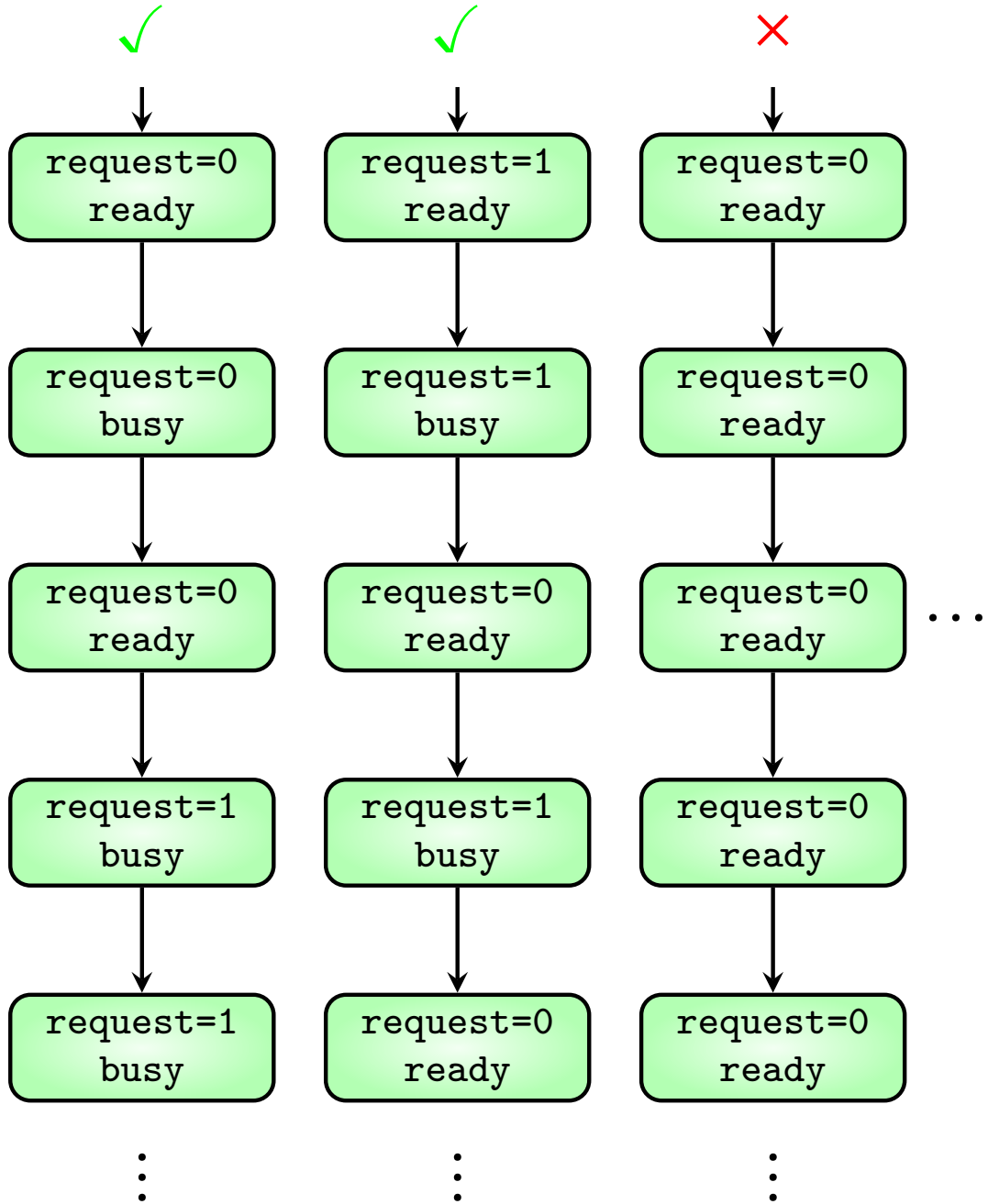
F (request=1)


request=1
ready
request=1
busy
request=0
ready
request=0
busy

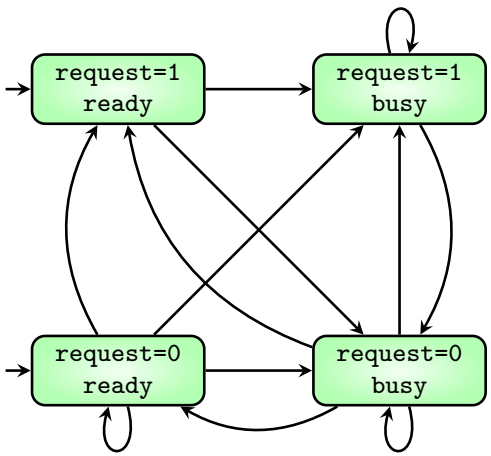

request=0
ready
request=0
busy
request=0
ready
request=1
busy
request=1
busy
request=1
busy
request=1
ready
request=1
busy
request=0
ready
request=0
ready
request=0
ready
request=0
ready
request=0
ready
request=0
ready

F (request=1)


request=1
ready
request=1
busy
request=0
ready
request=0
busy

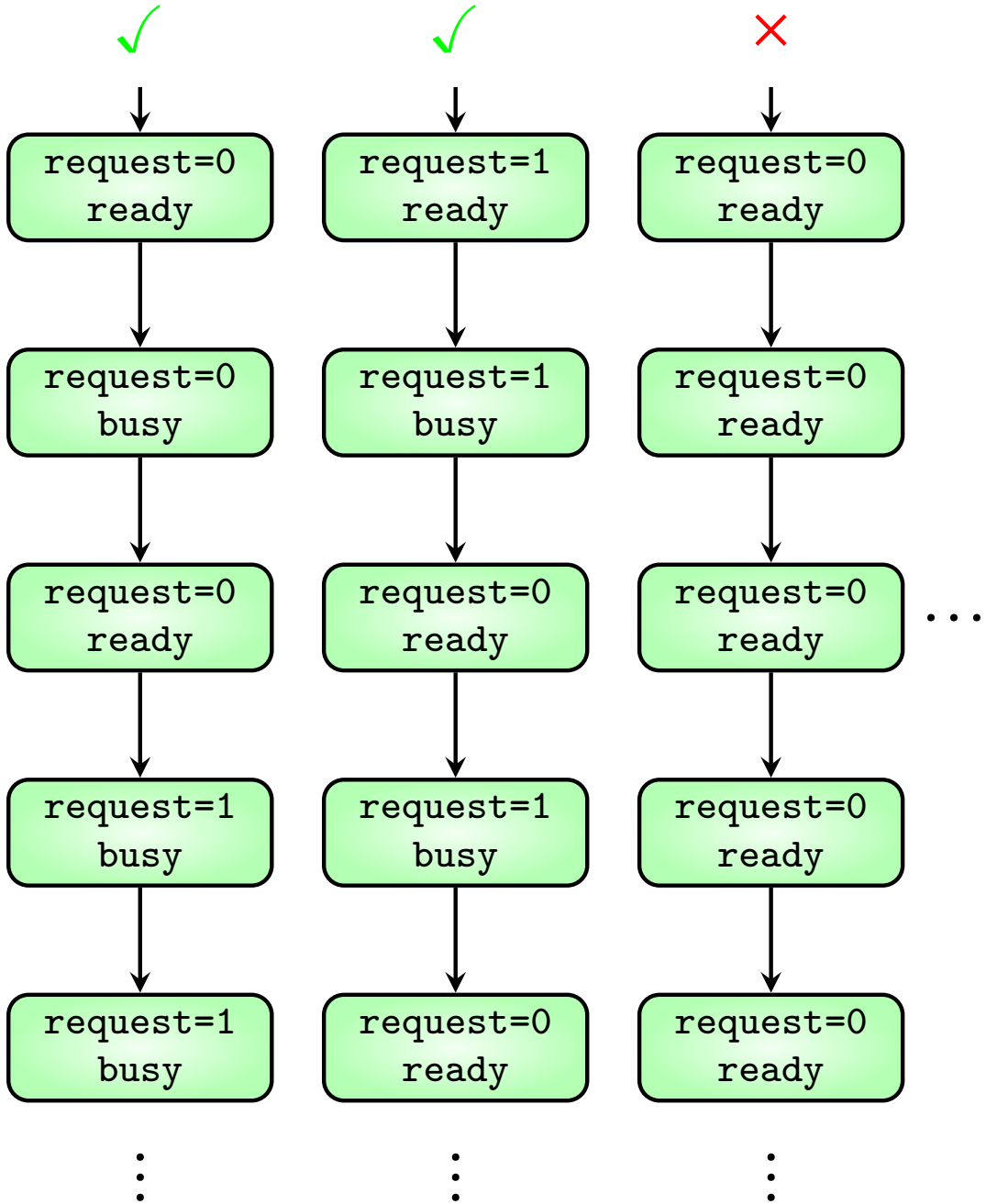

request=0
ready
request=0
busy
request=0
ready
request=1
busy
request=1
busy
request=1
busy
request=1
ready
request=1
busy
request=0
ready
request=0
ready
request=0
ready
request=0
ready
request=0
ready
request=0
ready

F (request=1)



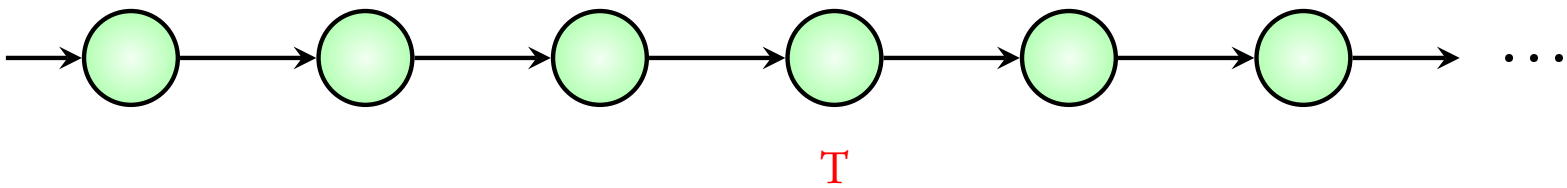
TS does not satisfy

F (request=1)



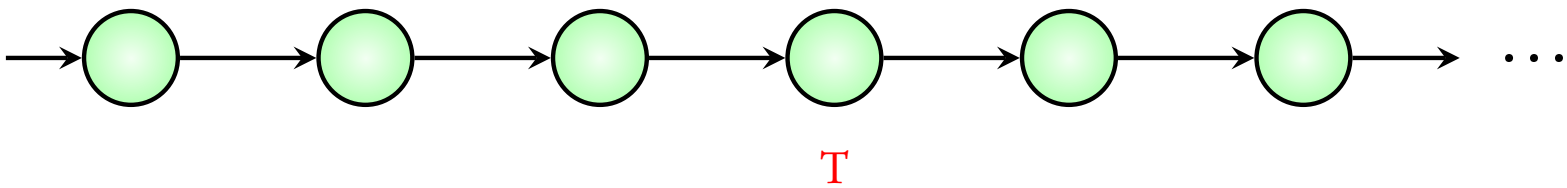
Execution **satisfies** F `(expr)` if

`expr` evaluates to T in **one of its states**



Execution **satisfies** F `(expr)` if

`expr` evaluates to T in **one of its states**



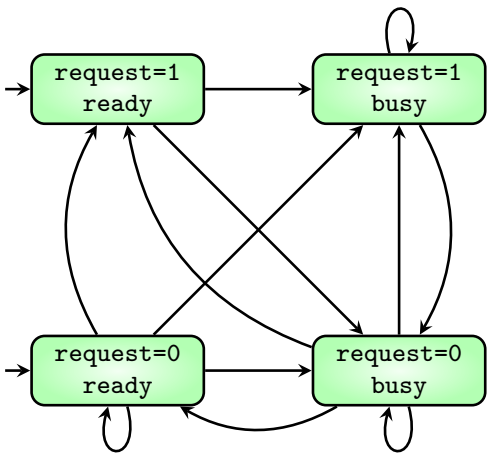
Transition system **satisfies** F `(expr)` if

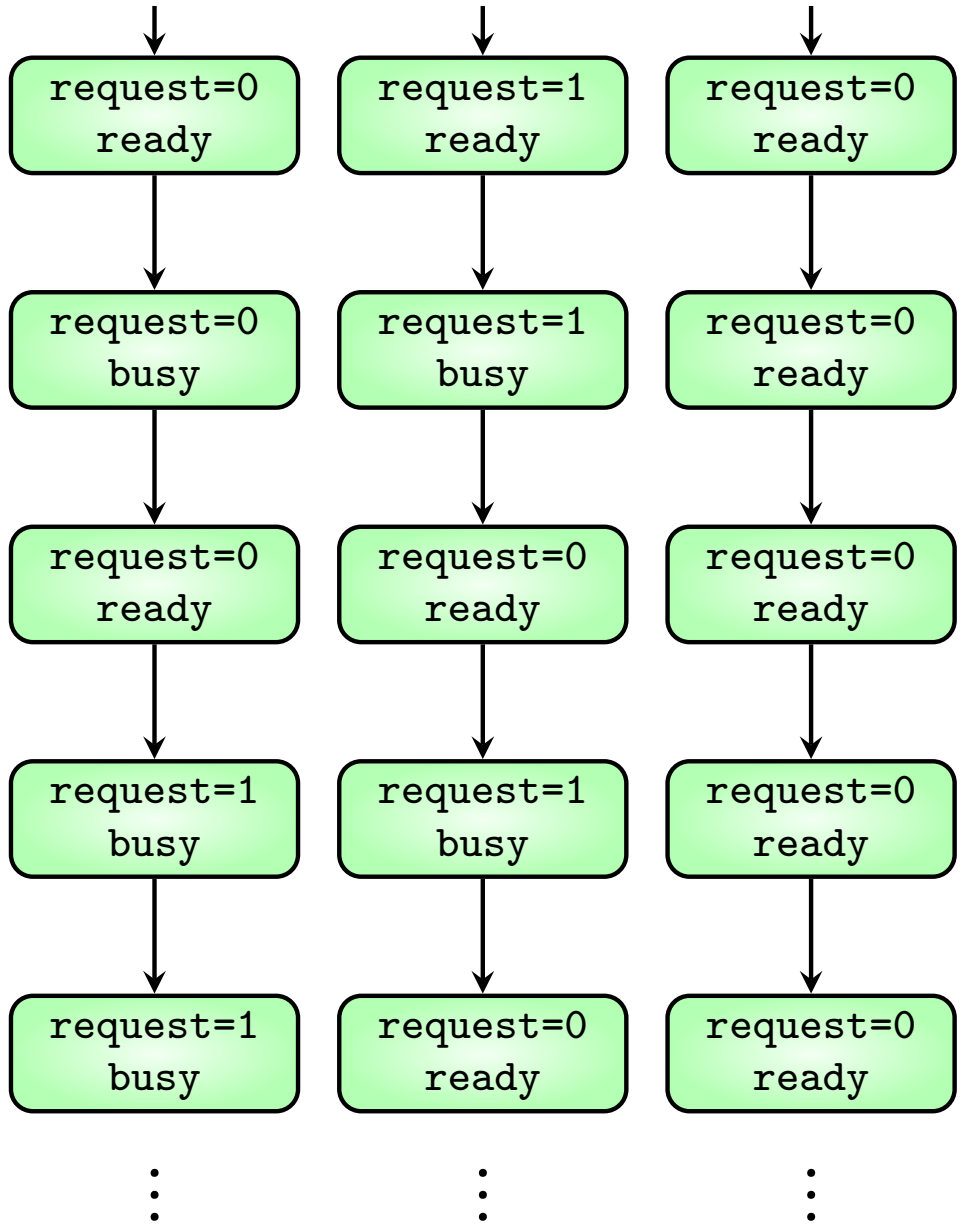
all its executions satisfy F `(expr)`

Checking the **F** requirement: **NuSMV demo**

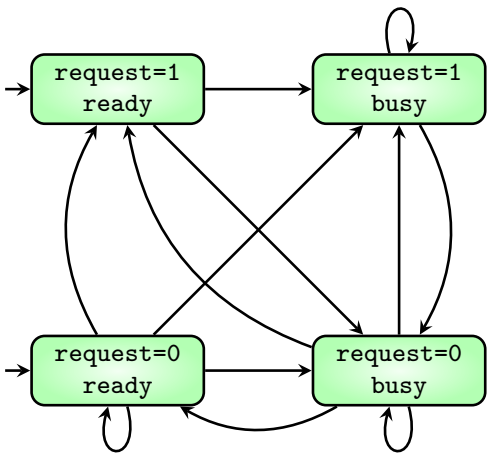
Coming next: Combining G and F

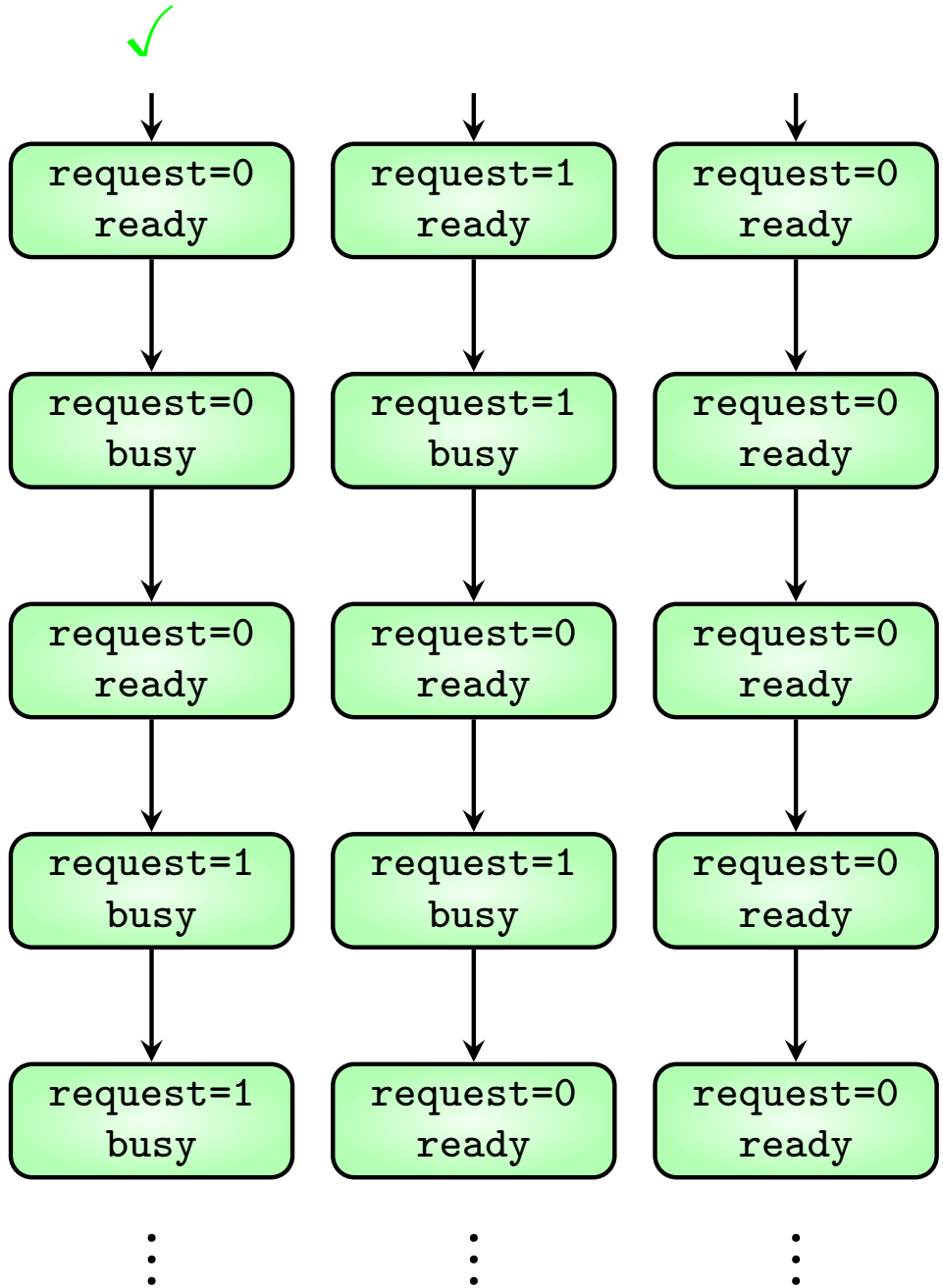
G (request=1 => F status=busy)



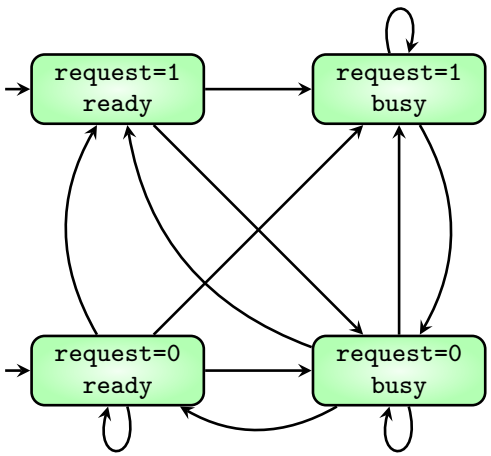


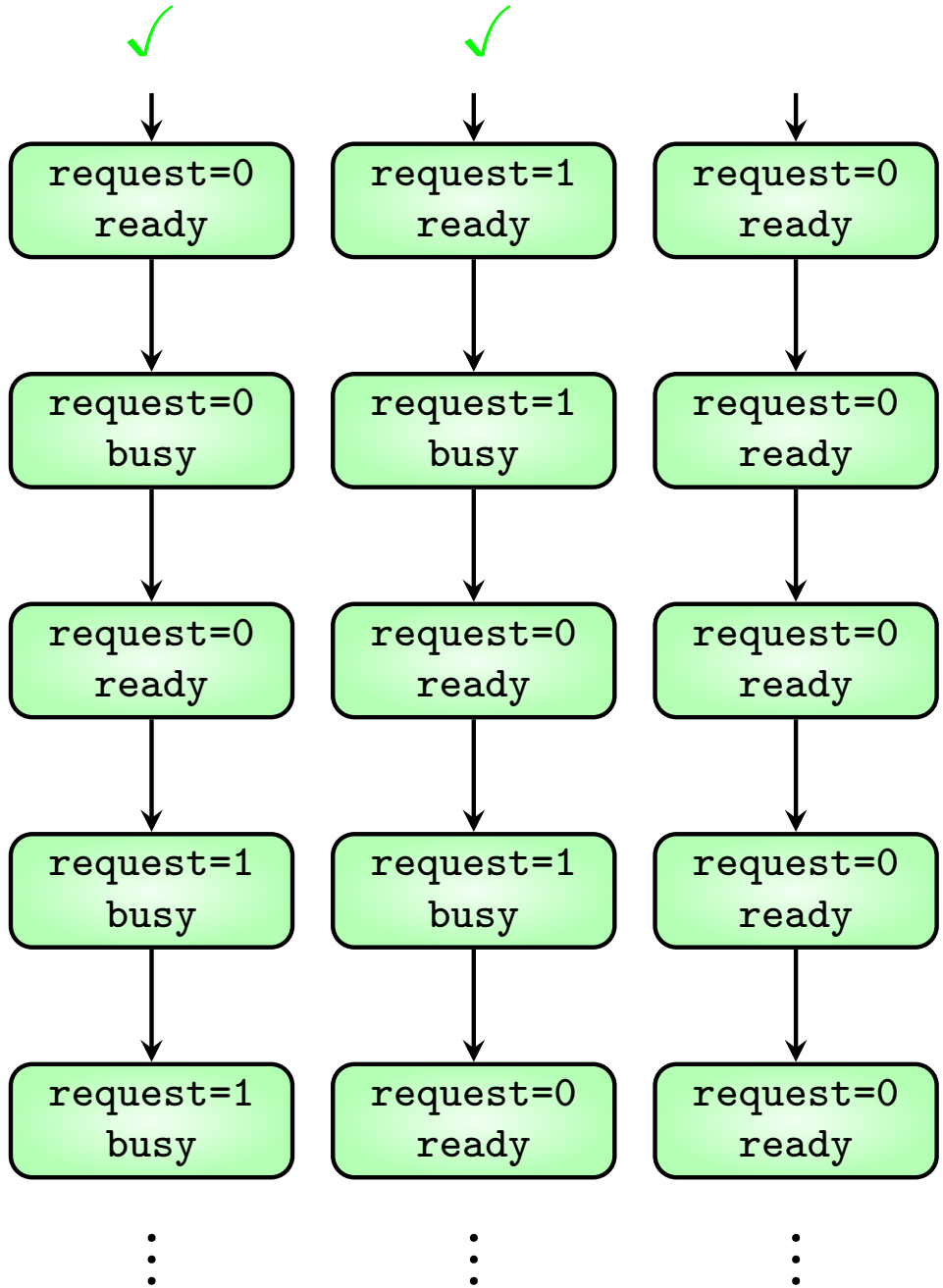
G (request=1 => F status=busy)



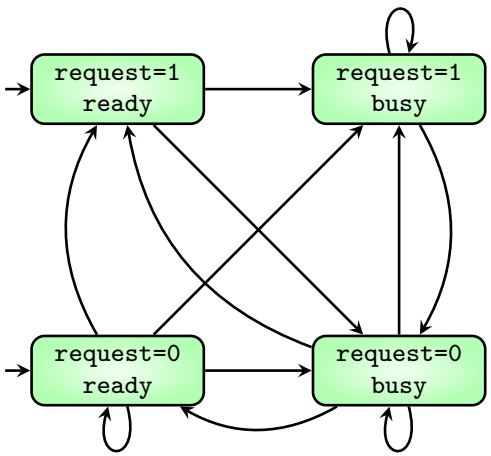


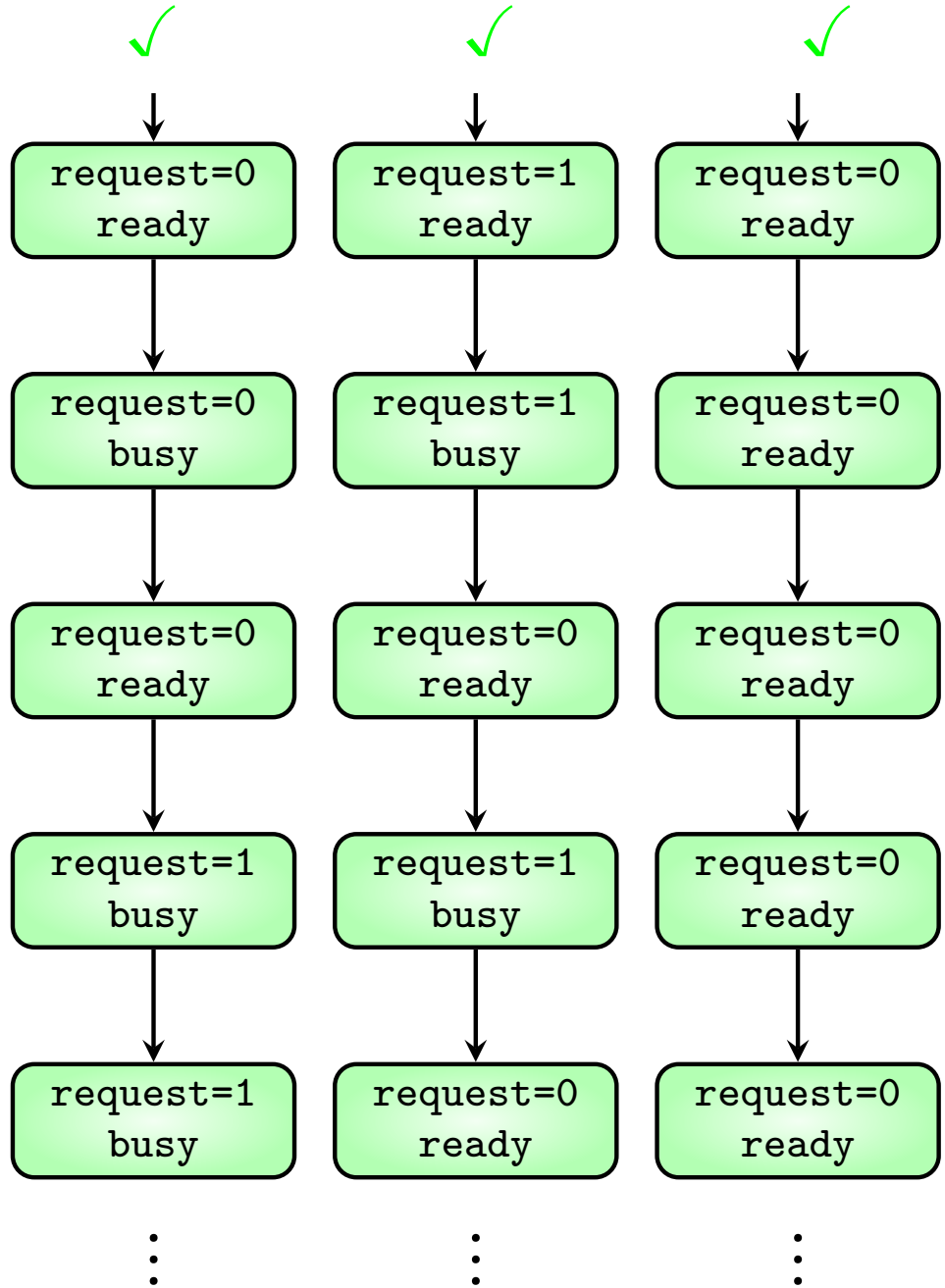
G (request=1 => F status=busy)



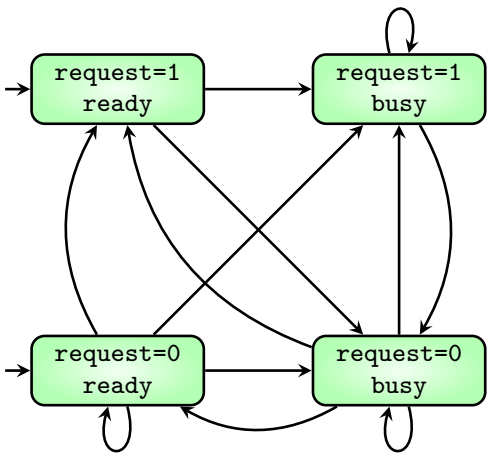


G (request=1 => F status=busy)



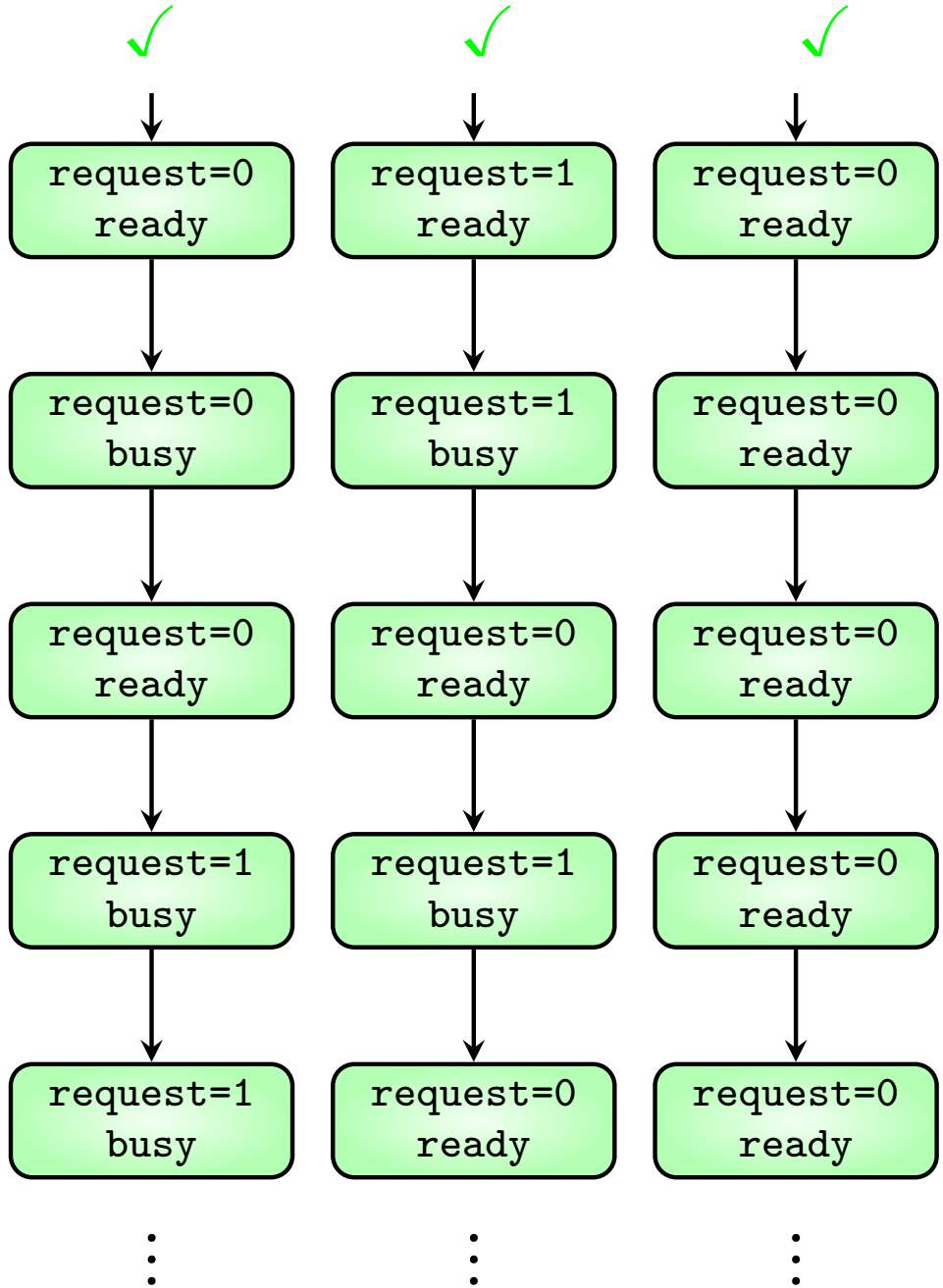


G (request=1 => F status=busy)



TS satisfies

G (request => F (status=busy))



Summary

Using NuSMV

Format for writing models

G and F requirements

Unit-2: Model-checker NuSMV

B. Srivathsan

Chennai Mathematical Institute

NPTEL-course

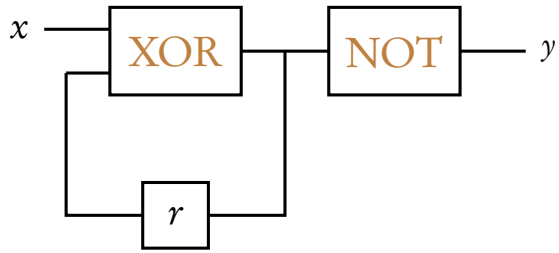
July - November 2015

Module 3:

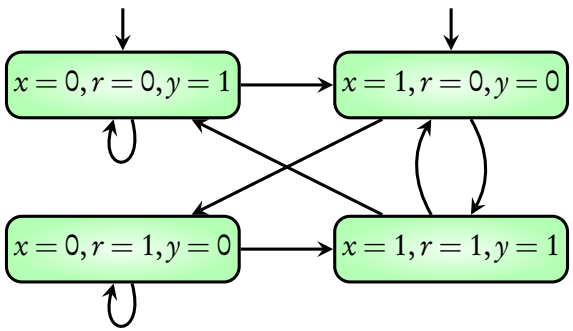
Hardware verification using NuSMV



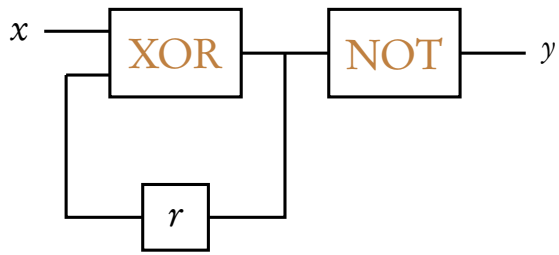
$$y = \text{NOT}\ (\ \text{XOR}\ (x, r)\ )$$
$$r_{next} = \text{XOR}\ (x, r)$$



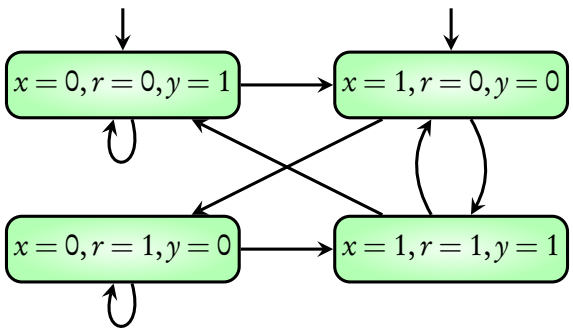
register



$$y = \text{NOT}\ (\ \text{XOR}\ (x, r)\ )$$
$$r_{next} = \text{XOR}\ (x, r)$$

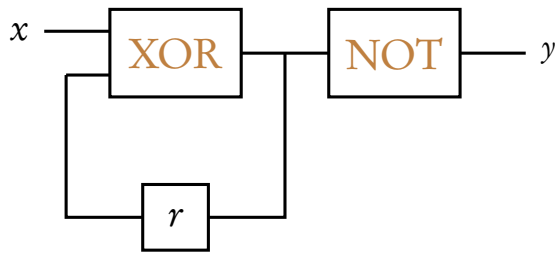


register

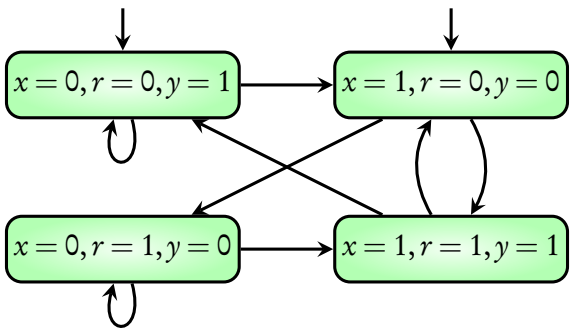



```
MODULE main
VAR
   x:  boolean;
   r:  boolean;
```

$$y = \text{NOT}\ (\ \text{XOR}\ (x, r)\ )$$
$$r_{next} = \text{XOR}\ (x, r)$$



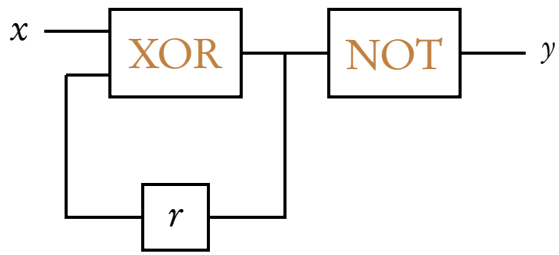
register



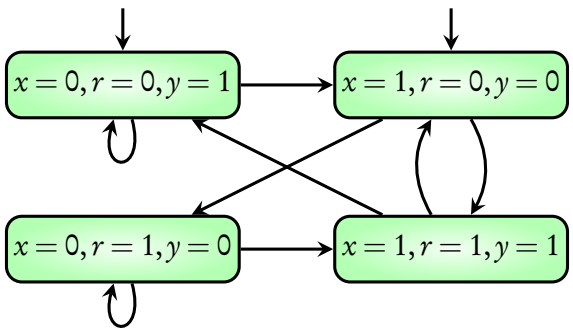

```
MODULE main
VAR
   x:  boolean;
   r:  boolean;
```

```
ASSIGN
   init(r) := FALSE;
```

$$y = \text{NOT}\ (\ \text{XOR}\ (x, r)\ )$$
$$r_{next} = \text{XOR}\ (x, r)$$

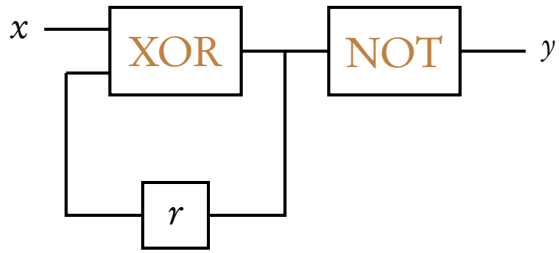


register

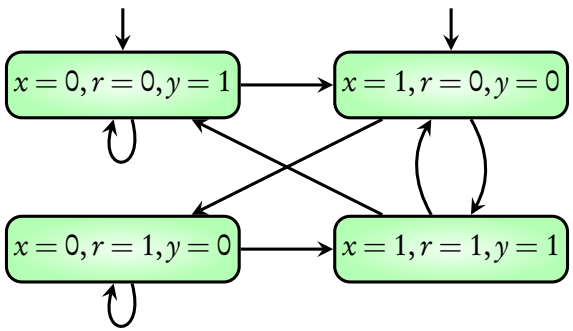



```
MODULE main
VAR
   x:  boolean;
   r:  boolean;

ASSIGN
   init(r) := FALSE;
   next(r) := x xor r;
```

$$y = \text{NOT}\ (\ \text{XOR}\ (x, r)\ )$$
$$r_{next} = \text{XOR}\ (x, r)$$



register




```
MODULE main
VAR
   x:  boolean;
   r:  boolean;
DEFINE
   y := !(x xor r);
ASSIGN
   init(r) := FALSE;
   next(r) := x xor r;
```

Simple circuit

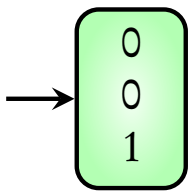
Use of `DEFINE`

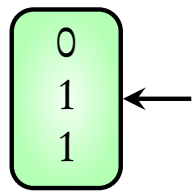

in_1
in_2
NAND
out

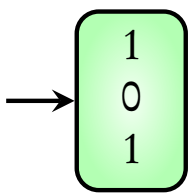

```
MODULE main
VAR
   in1:  boolean;
   in2:  boolean;
DEFINE
   -- ZERO DELAY
   out := !(in1 & in2);
```

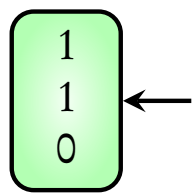
0 0 1		0 1 1
1 0 1		1 1 0

```
MODULE main
VAR
   in1:  boolean;
   in2:  boolean;
DEFINE
   -- ZERO DELAY
   out := !(in1 & in2);
```

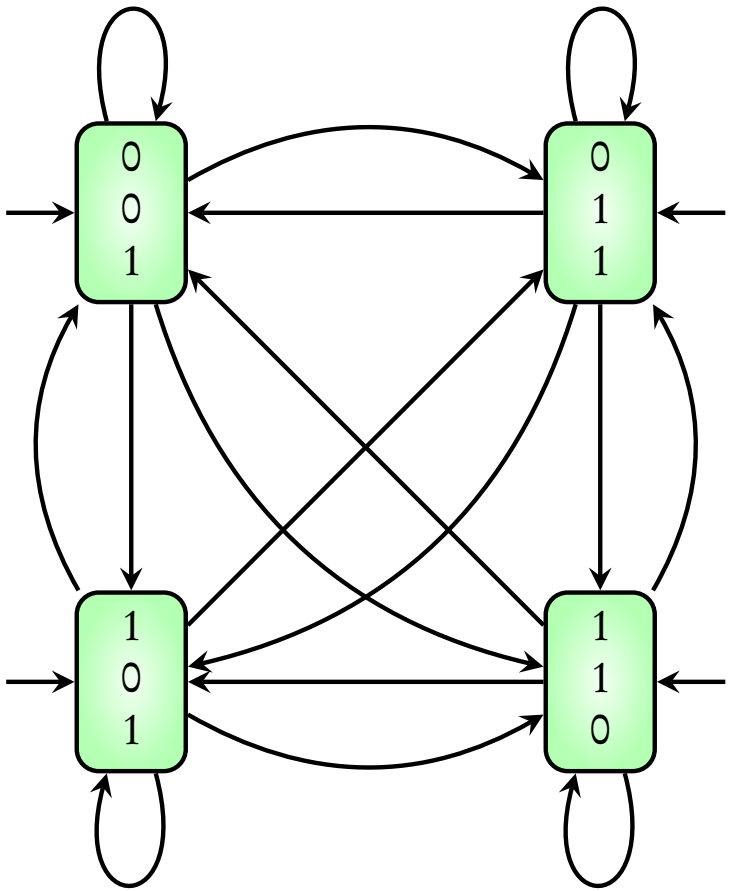






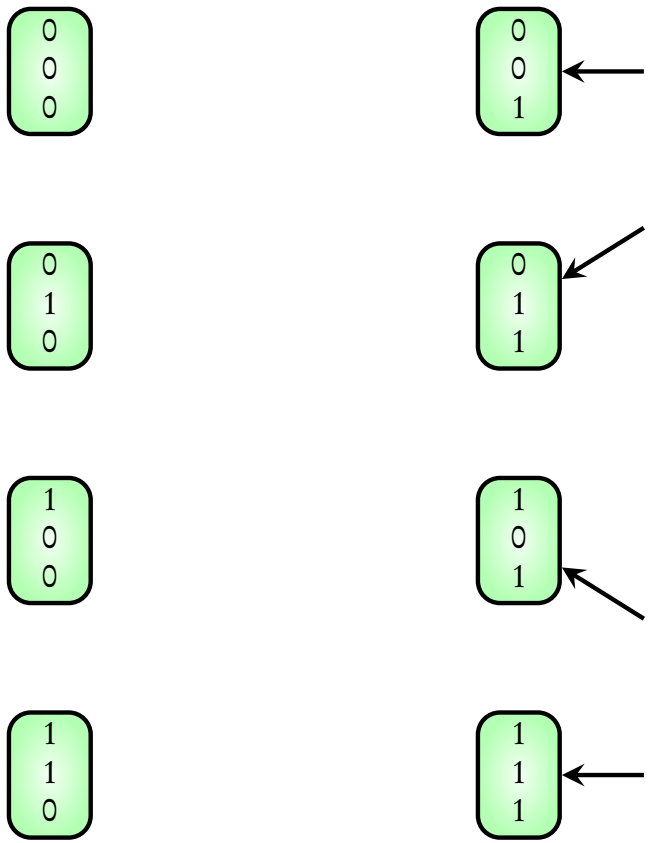



```
MODULE main
VAR
   in1:  boolean;
   in2:  boolean;
DEFINE
   -- ZERO DELAY
   out := !(in1 & in2);
```

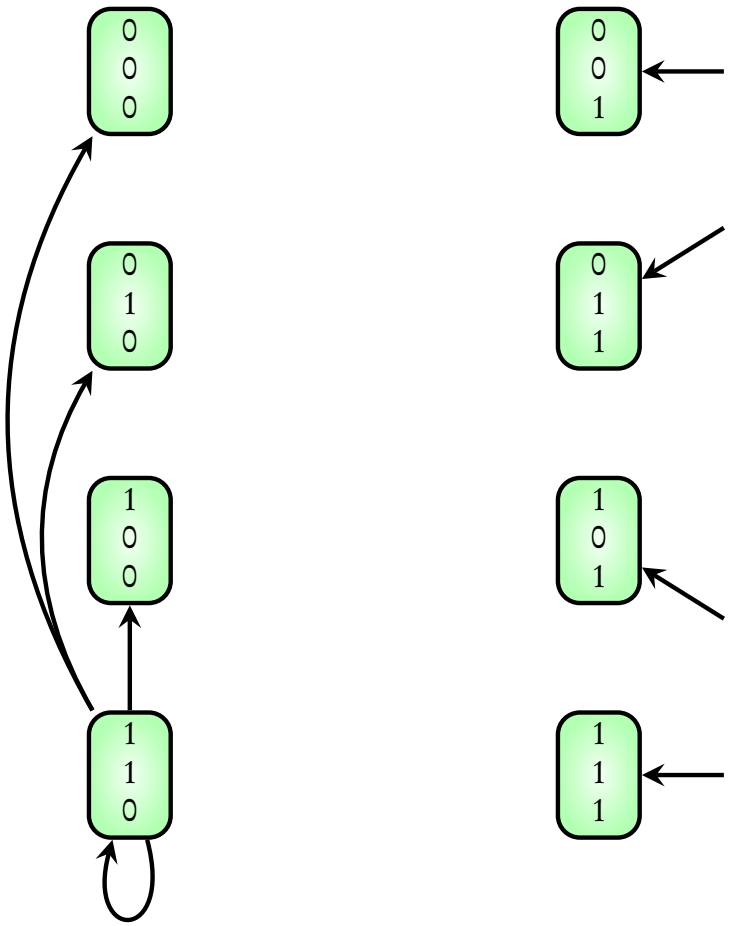



```
MODULE main
VAR
   in1:  boolean;
   in2:  boolean;
DEFINE
   -- ZERO DELAY
   out := !(in1 & in2);
```

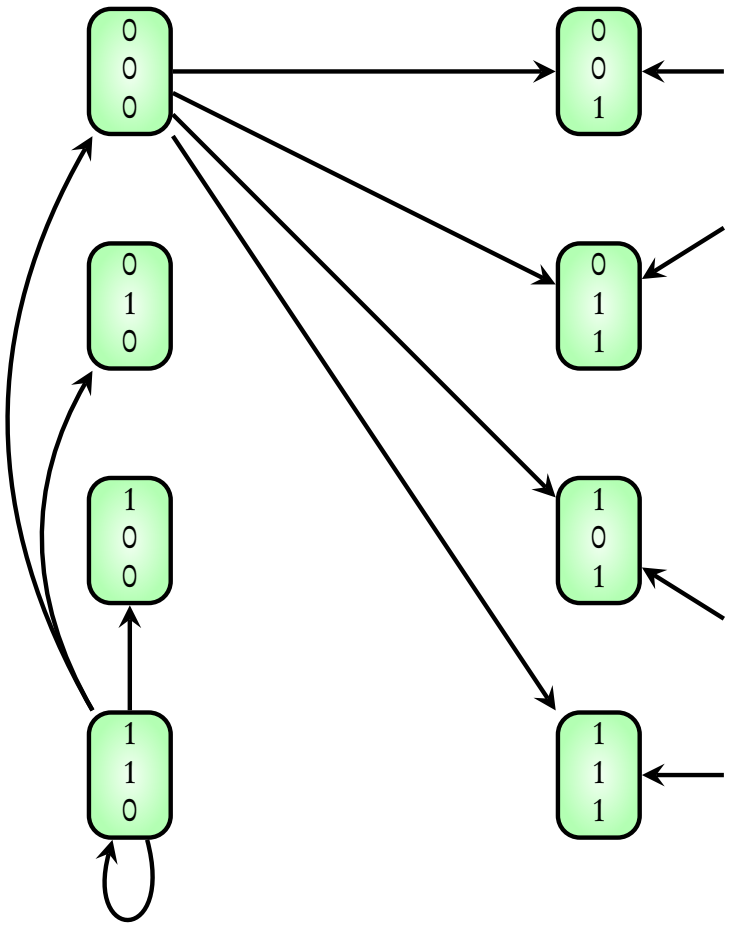

```
MODULE main
VAR
   in1:  boolean;
   in2:  boolean;
   out:  boolean;
ASSIGN
   -- UNIT DELAY
   init(out) := TRUE;
   next(out) := !(in1 & in2);
```



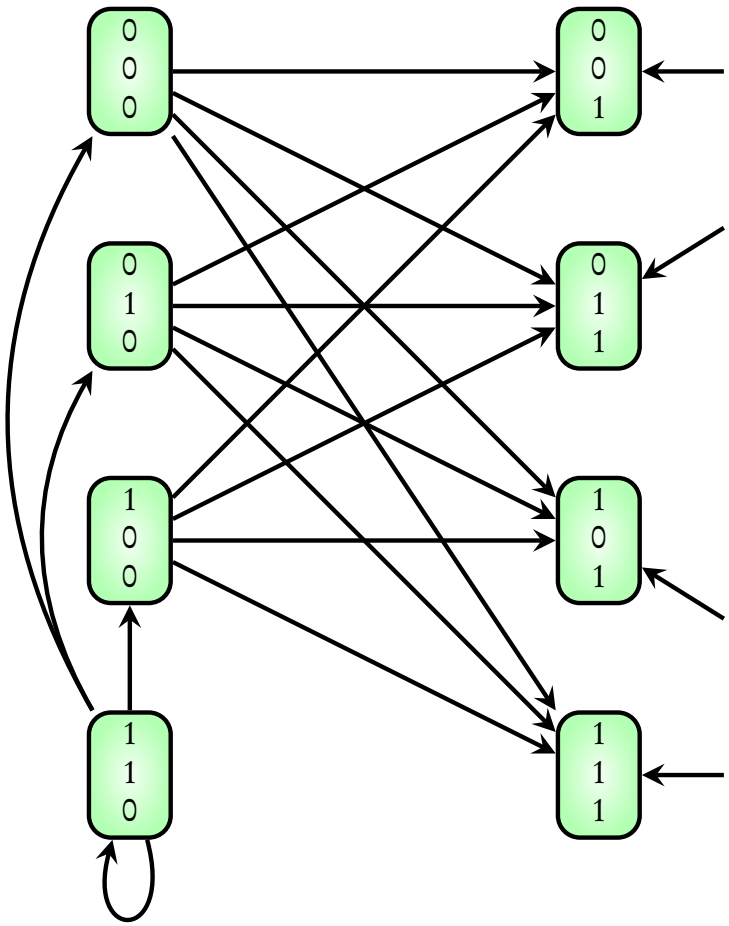

```
MODULE main
VAR
   in1:  boolean;
   in2:  boolean;
   out:  boolean;
ASSIGN
   -- UNIT DELAY
   init(out) := TRUE;
   next(out) := !(in1 & in2);
```



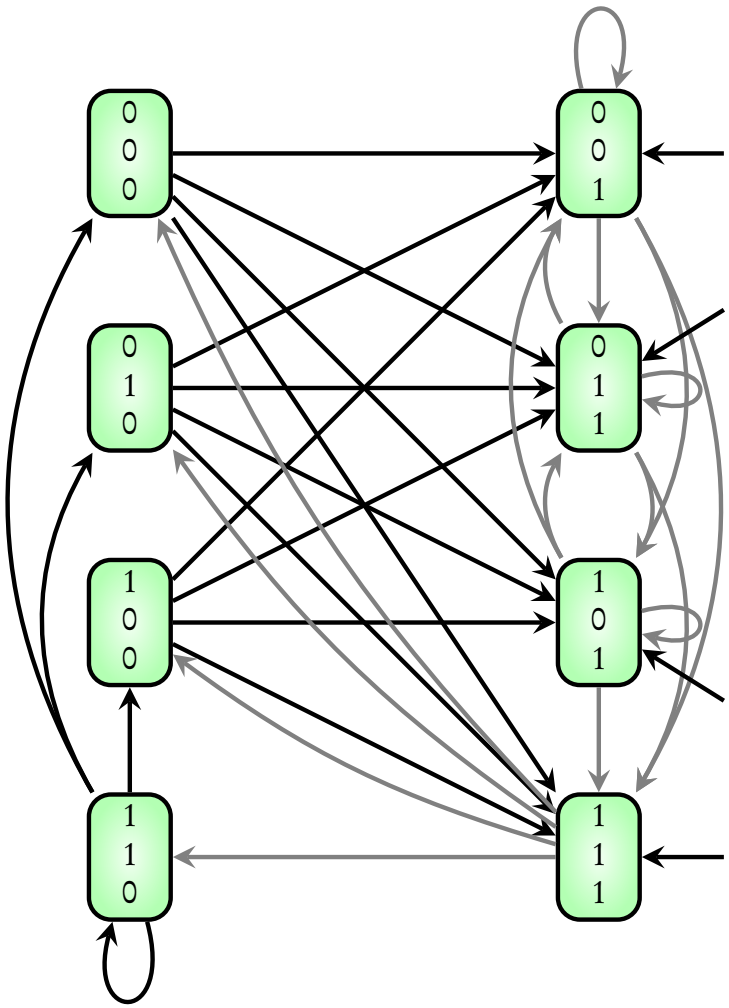

```
MODULE main
VAR
   in1:  boolean;
   in2:  boolean;
   out:  boolean;
ASSIGN
   -- UNIT DELAY
   init(out) := TRUE;
   next(out) := !(in1 & in2);
```



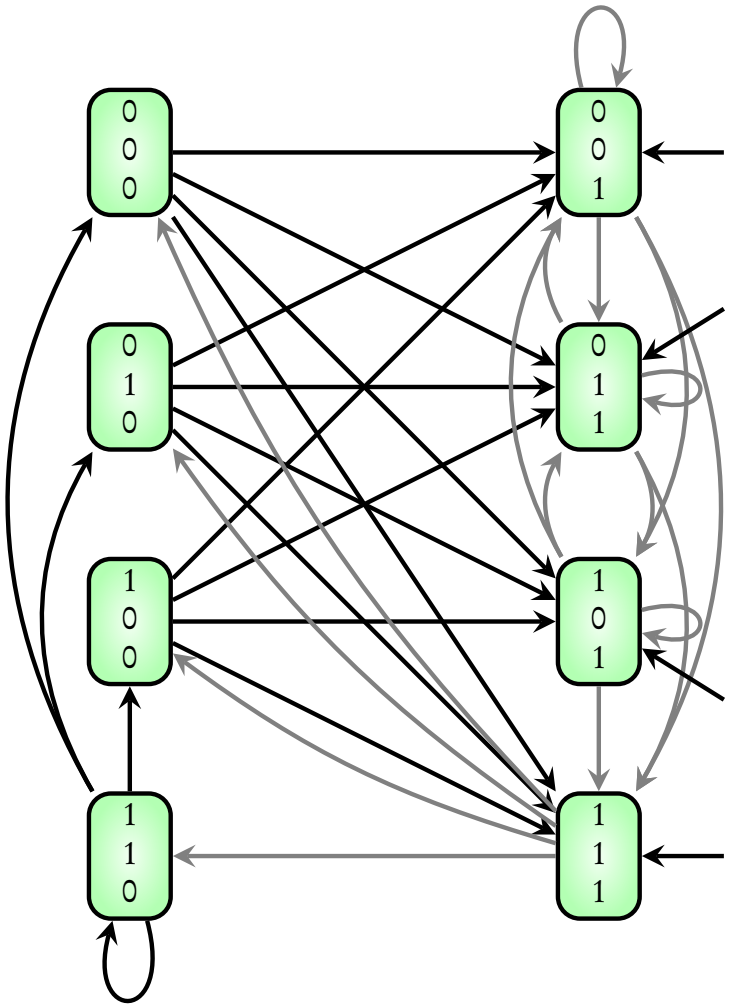

```
MODULE main
VAR
   in1:  boolean;
   in2:  boolean;
   out:  boolean;
ASSIGN
   -- UNIT DELAY
   init(out) := TRUE;
   next(out) := !(in1 & in2);
```



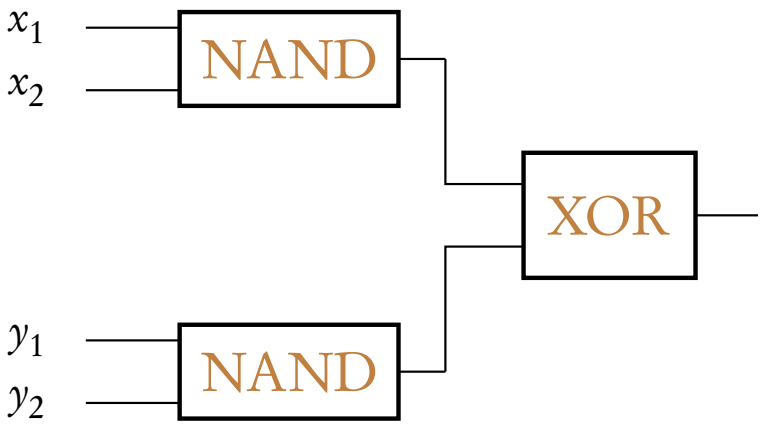

```
MODULE main
VAR
   in1:  boolean;
   in2:  boolean;
   out:  boolean;
ASSIGN
   -- UNIT DELAY
   init(out) := TRUE;
   next(out) := !(in1 & in2);
```



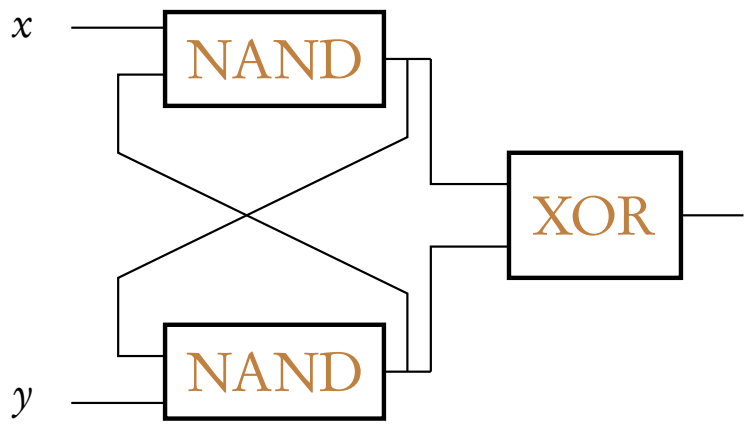

```
MODULE main
VAR
   in1:  boolean;
   in2:  boolean;
   out:  boolean;
ASSIGN
   -- UNIT DELAY
   init(out) := TRUE;
   next(out) := !(in1 & in2);
```




```
MODULE main
VAR
   input1:  boolean;
   input2:  boolean;
   q:  nand2(input1, input2);

MODULE nand2(in1, in2)
VAR
   out:  boolean;
ASSIGN
   -- UNIT DELAY
   init(out) := TRUE;
   next(out) := !(in1 & in2);
```




```
MODULE main
VAR
   x1:  boolean; x2:boolean;
   y1:  boolean; y2:boolean;
   q1:  nand2(x1, x2);
   q2:  nand2(y1, y2);
DEFINE
   -- ZERO DELAY
   fout := q1.out xor q2.out;


MODULE nand2(in1, in2)
VAR
   out:  boolean;
ASSIGN
   -- UNIT DELAY
   init(out) := TRUE;
   next(out) := !(in1 & in2);
```




```
MODULE main
VAR
   x:  boolean;
   y:  boolean;
   q1:  nand2(x, q2.out);
   q2:  nand2(q1.out, y);
DEFINE
   -- ZERO DELAY
   fout := q1.out xor q2.out;

MODULE nand2(in1, in2)
VAR
   out:  boolean
ASSIGN
   -- UNIT DELAY
   init(out) := TRUE;
   next(out) := !(in1 & in2);
```

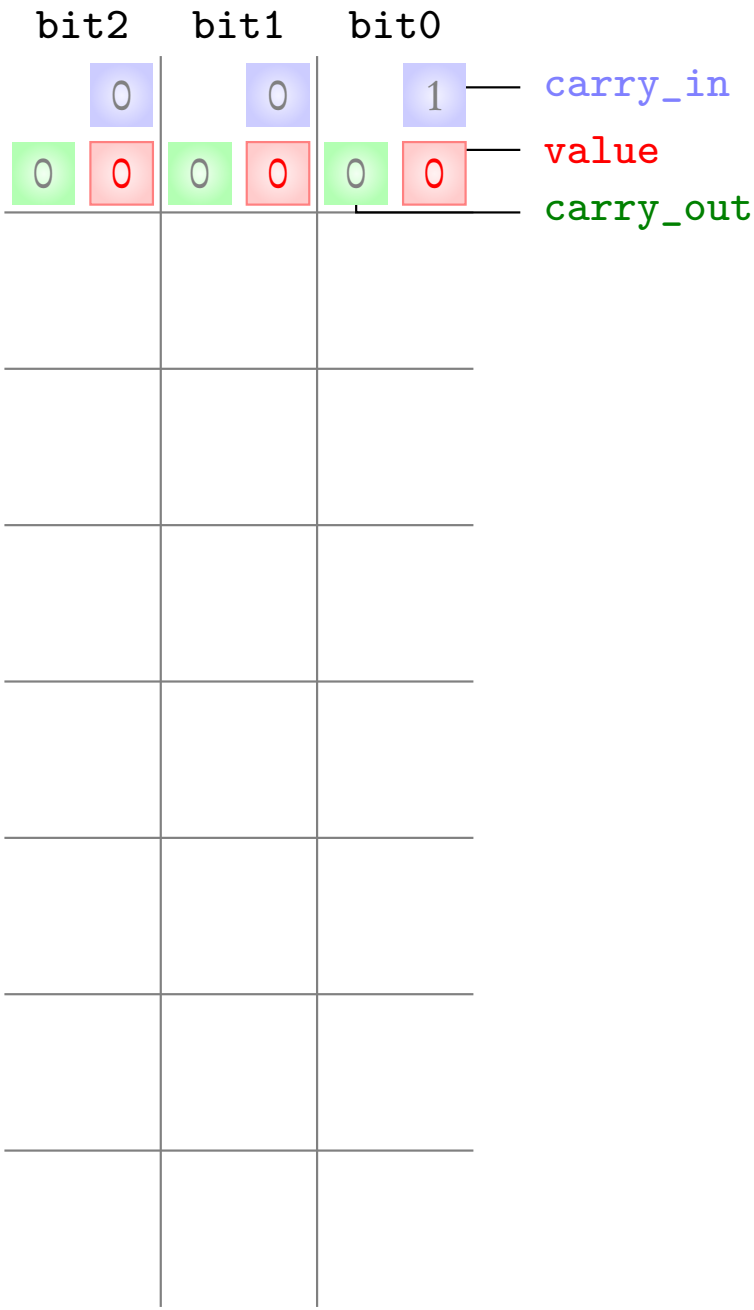
Simple circuit

Use of `DEFINE`

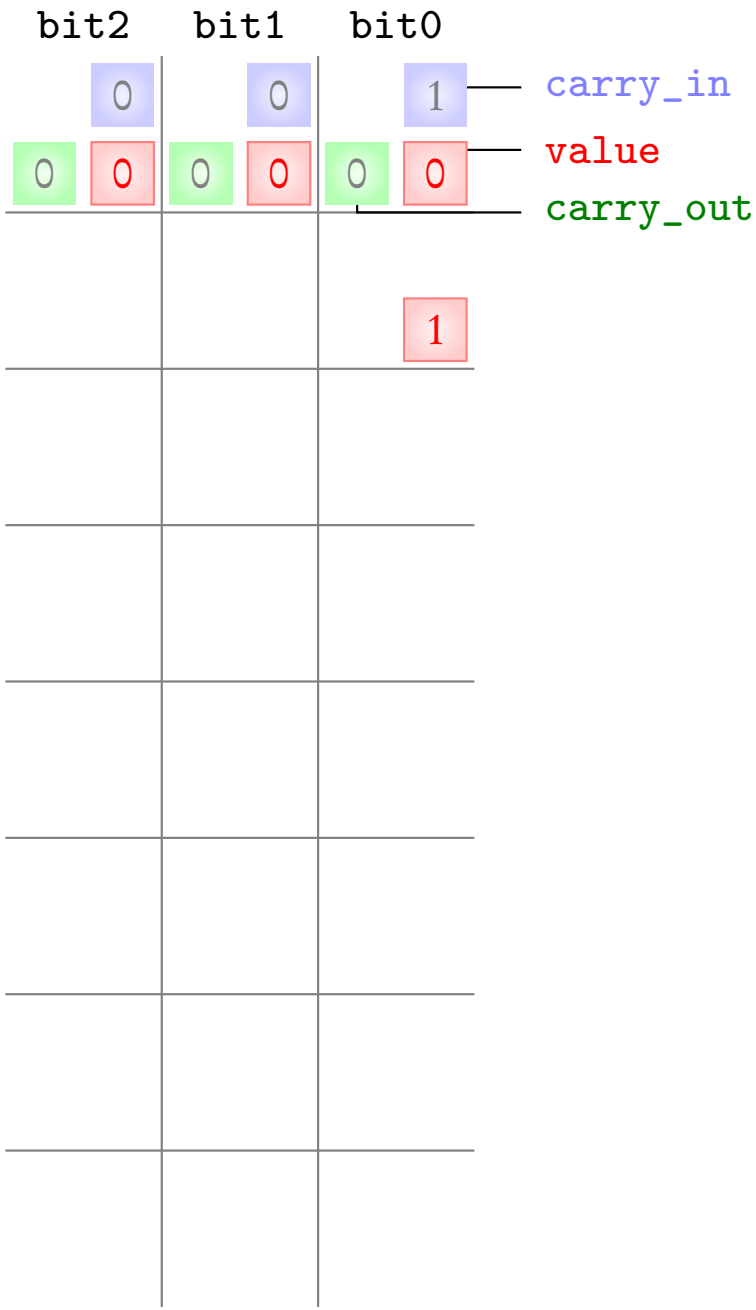
Hierarchical designs

Use of `MODULE`

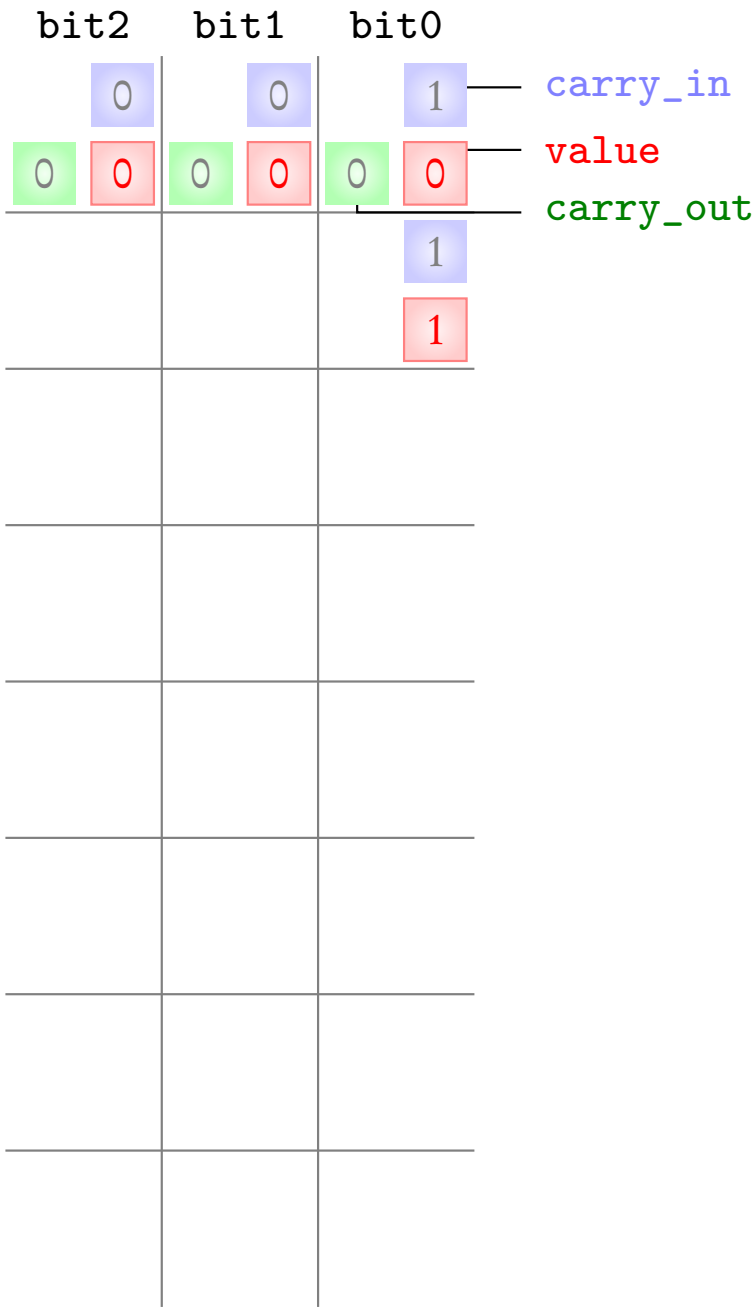
```
MODULE counter_cell(carry_in)
VAR
   value:boolean;
ASSIGN
   init(value):=FALSE;
   next(value):= value xor carry_in;
DEFINE
   carry_out := carry_in & value;

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



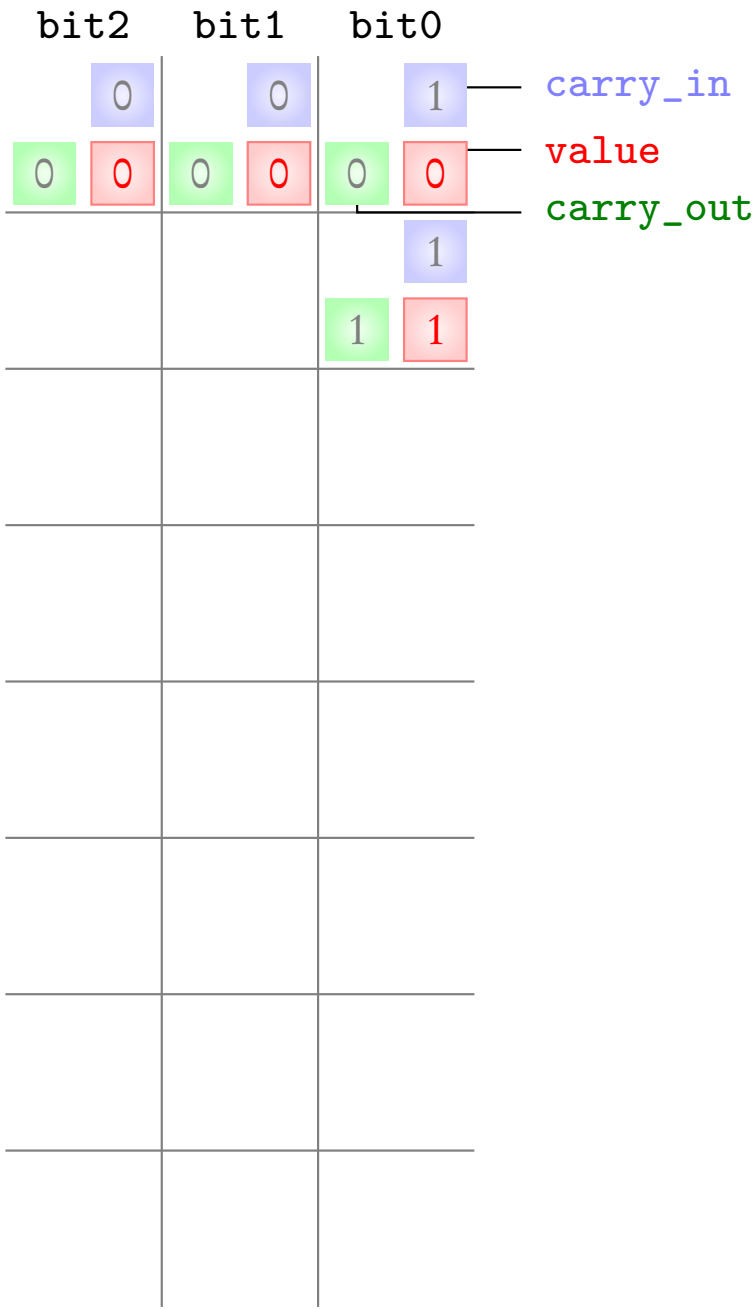

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


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



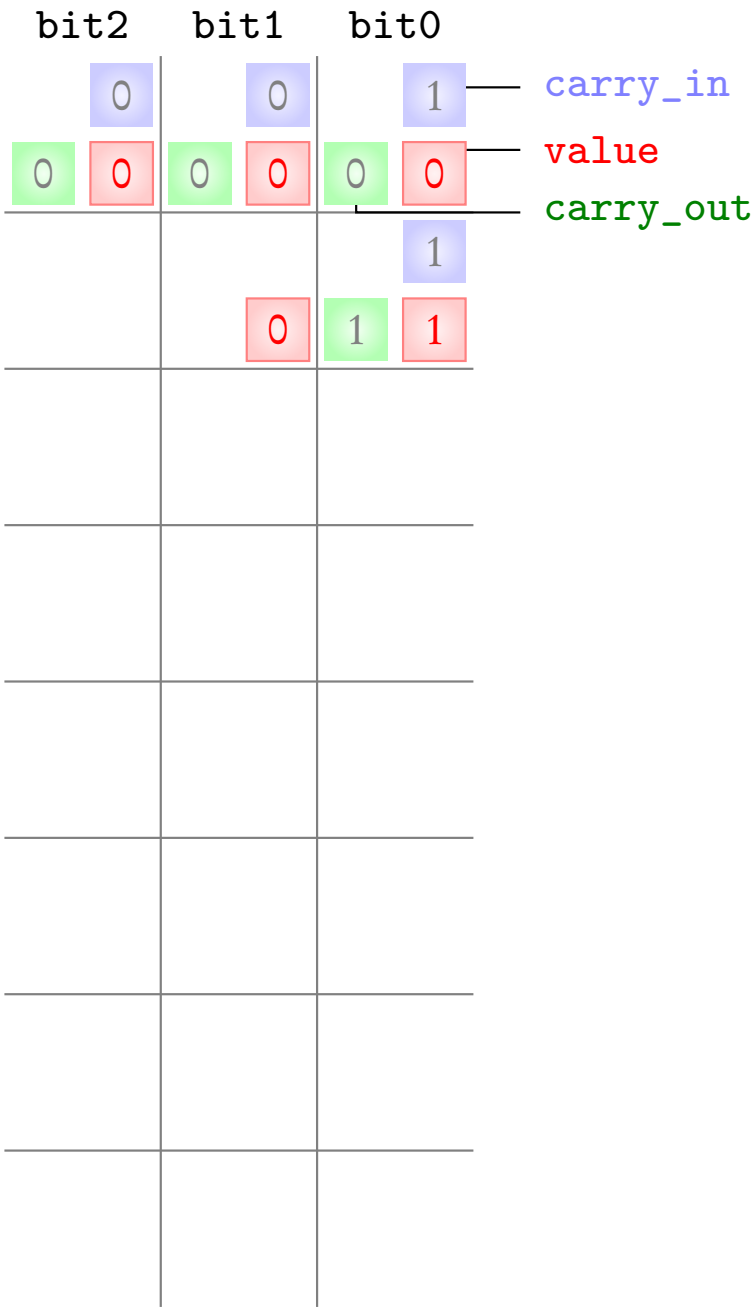

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


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



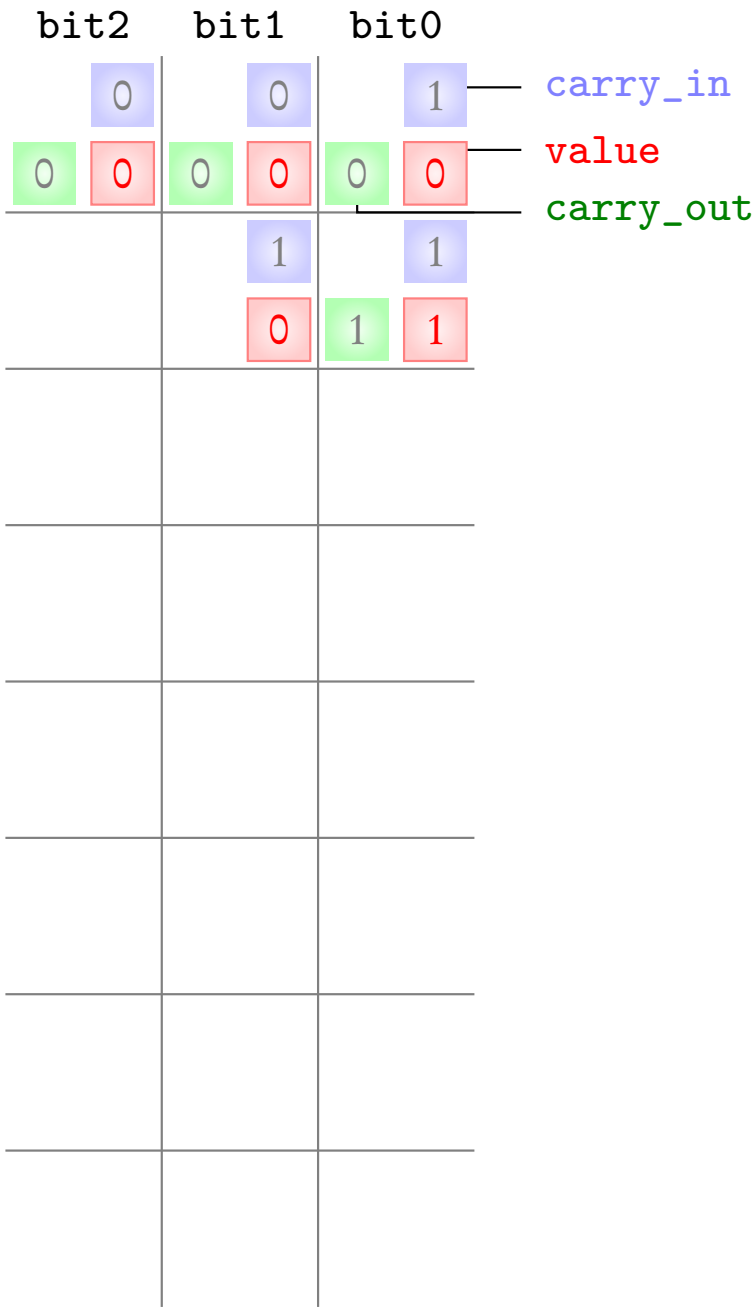

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


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



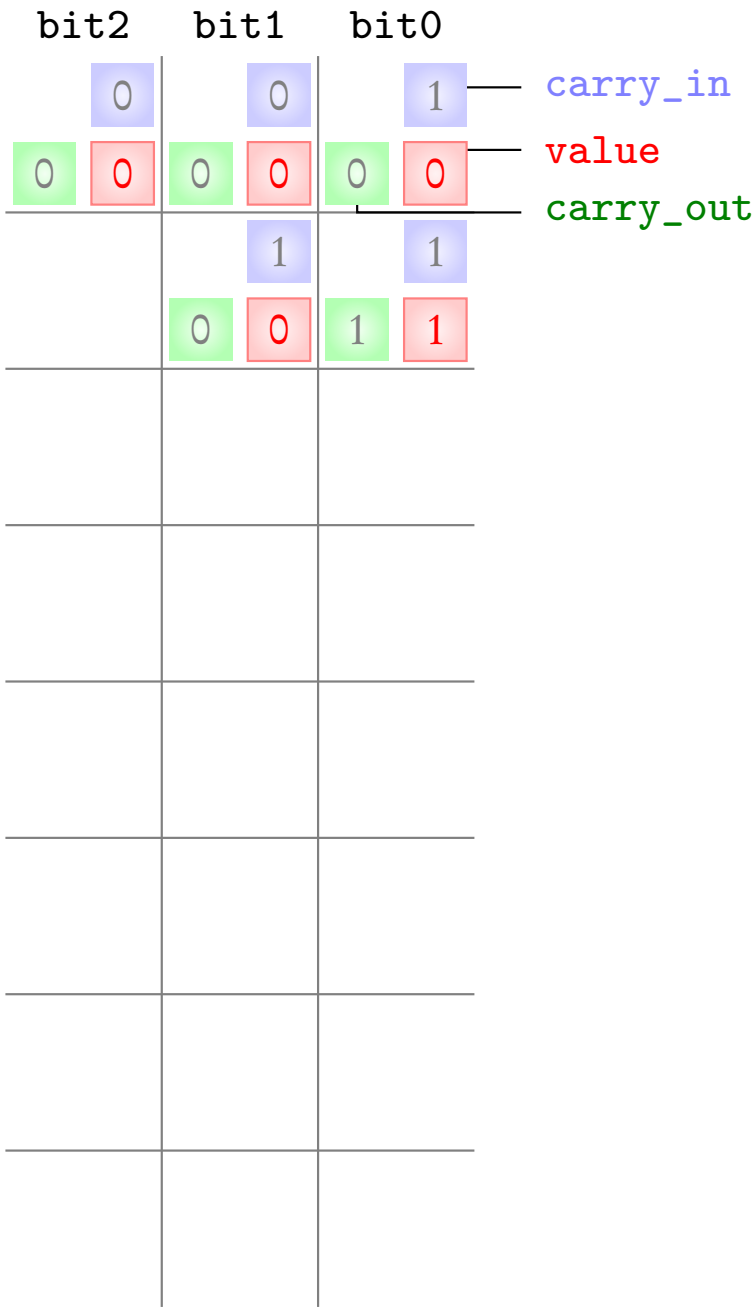

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


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



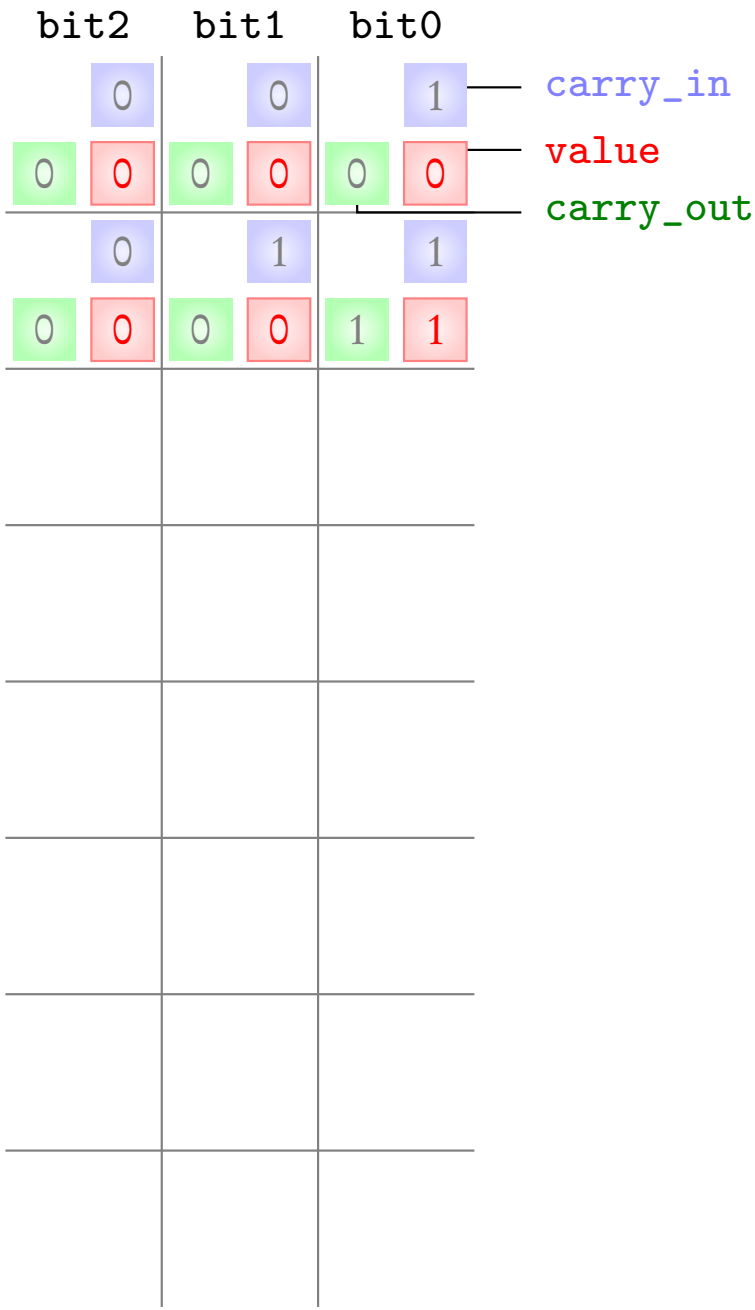

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


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



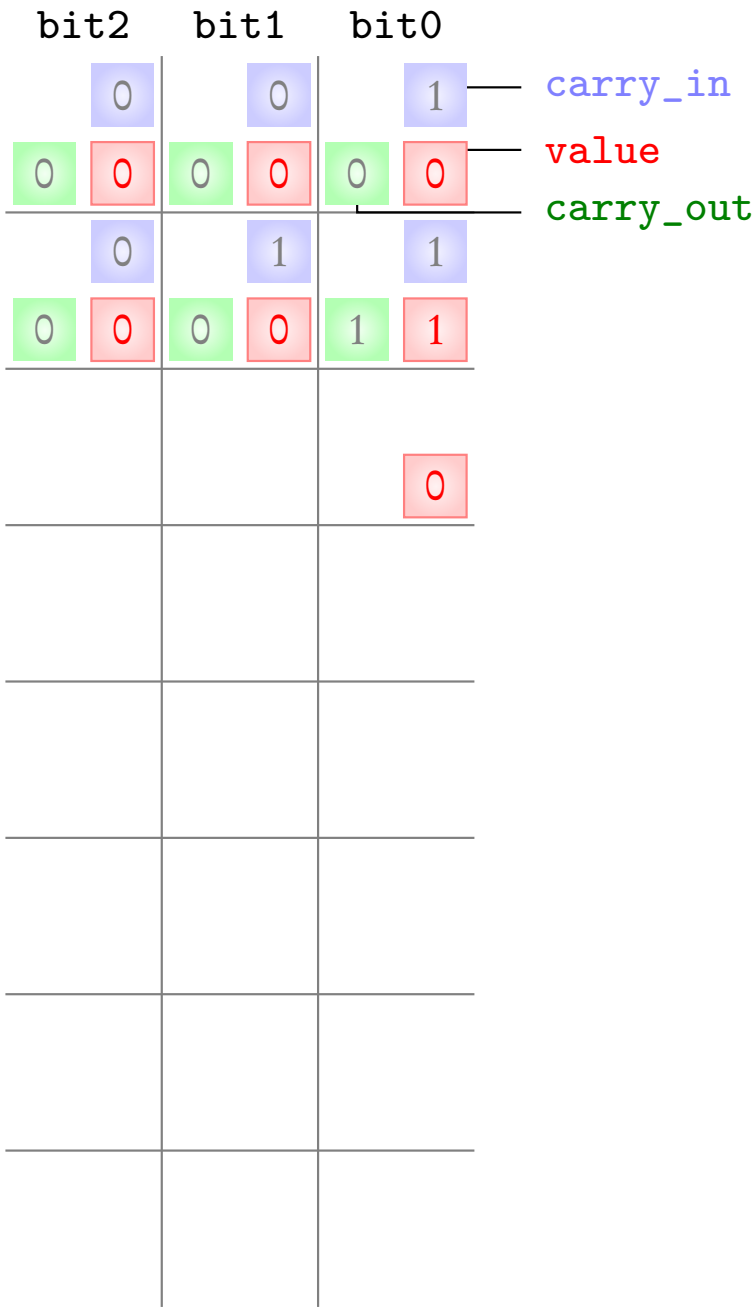

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


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



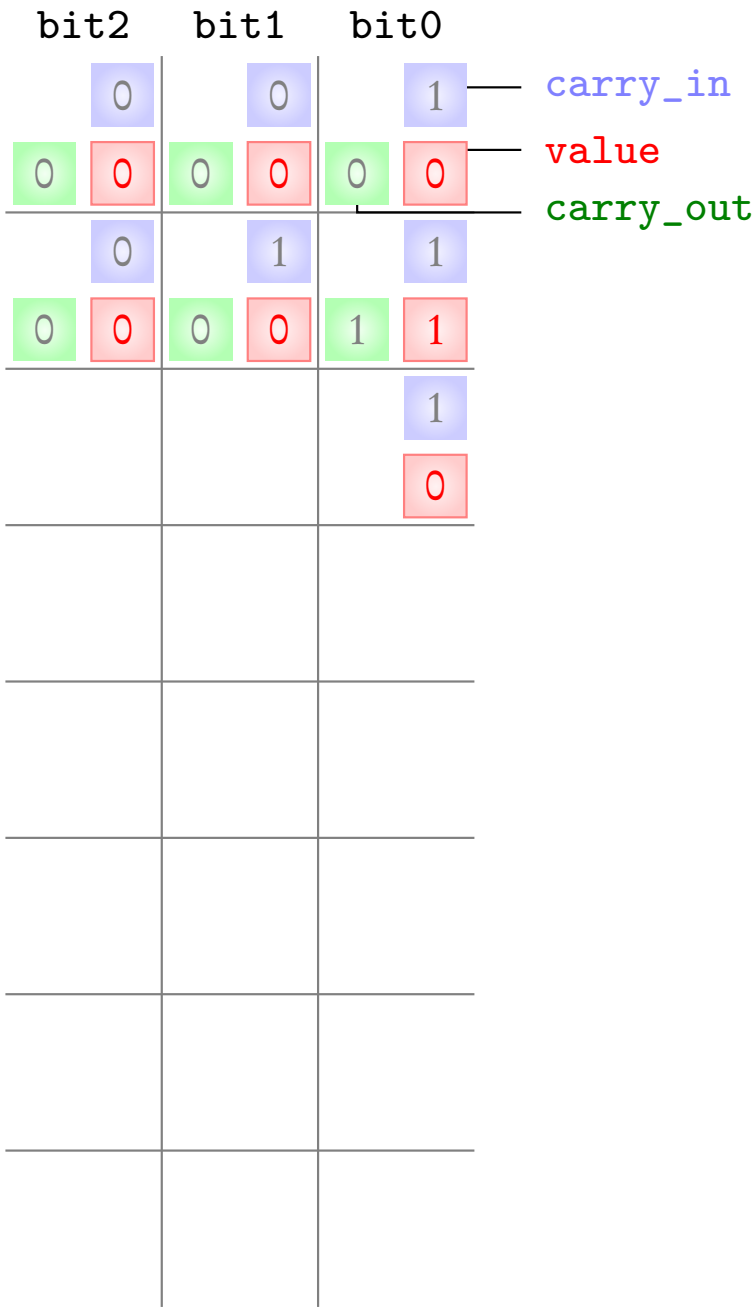

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


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



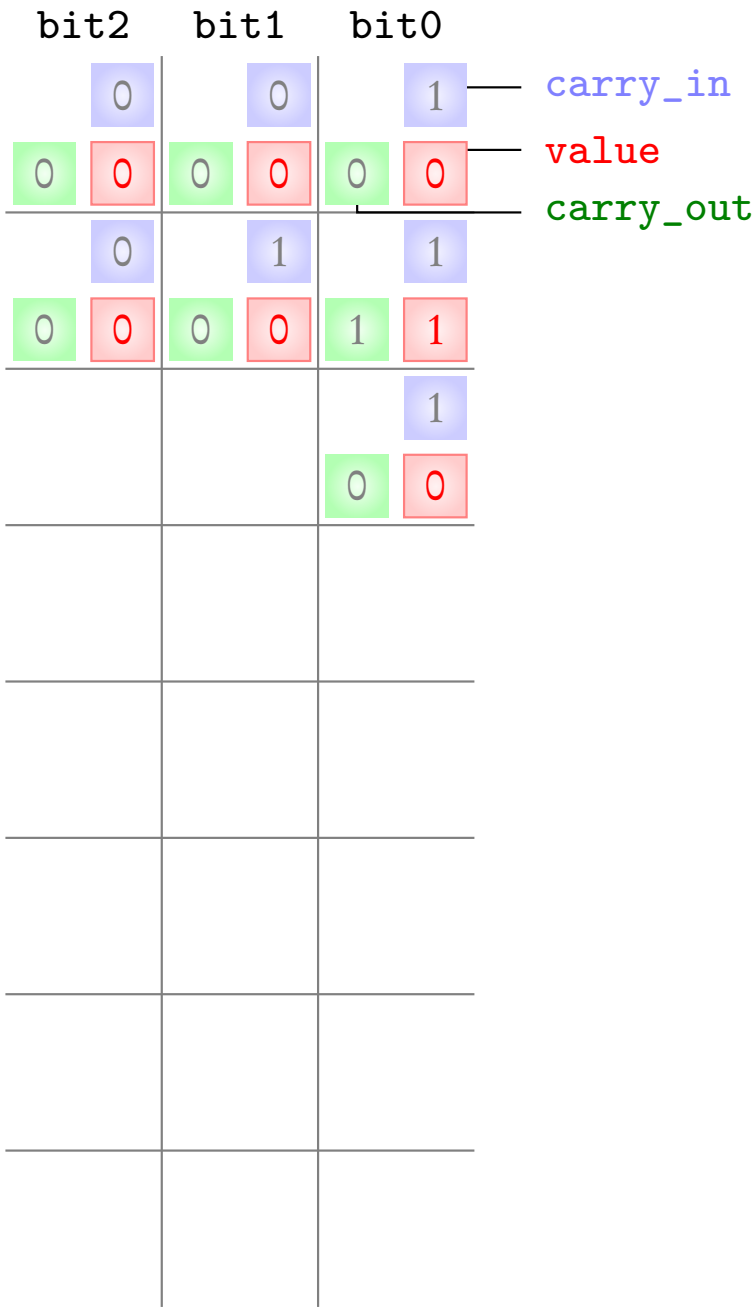

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


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



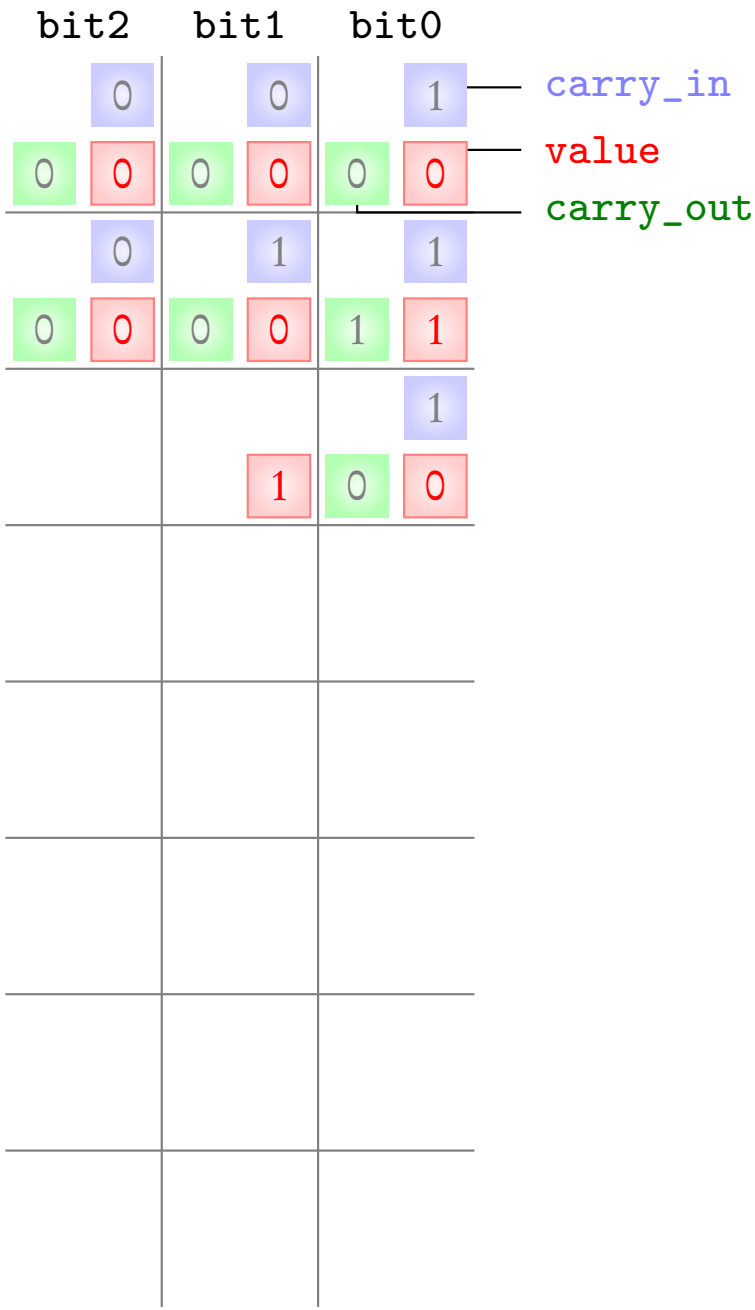

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


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



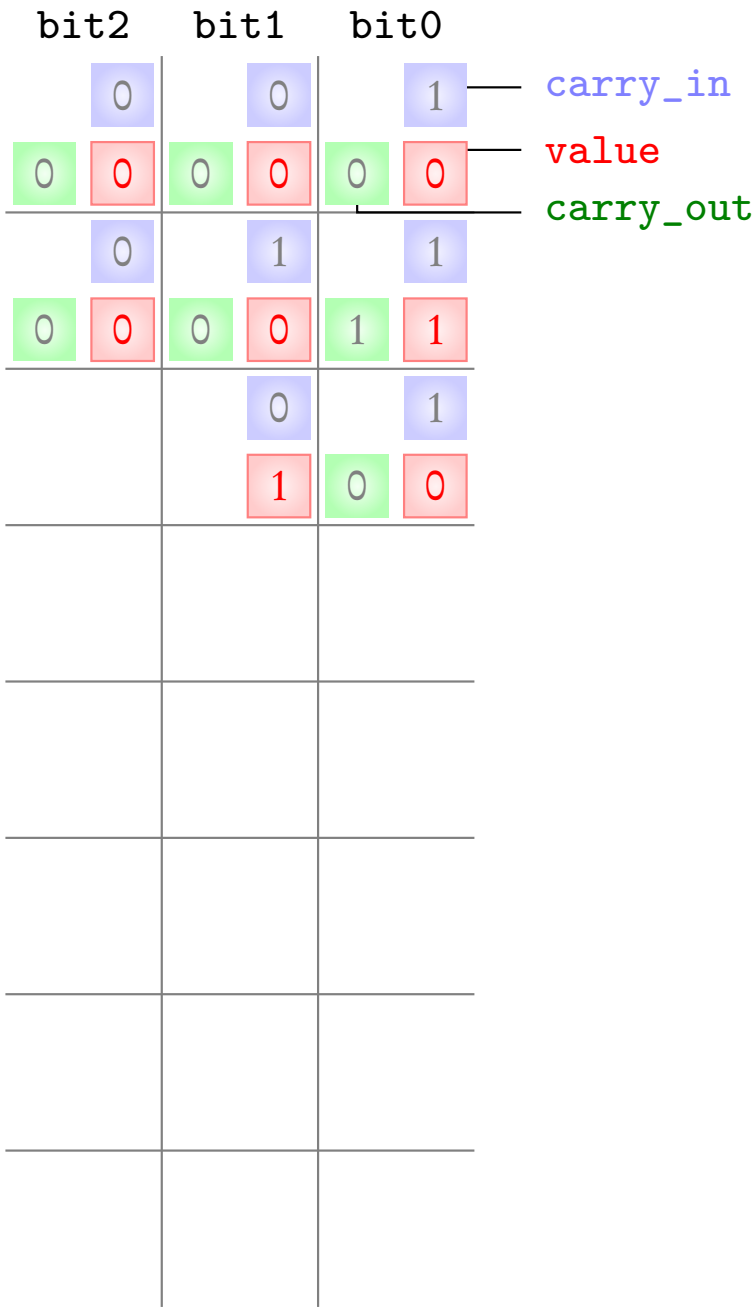

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


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



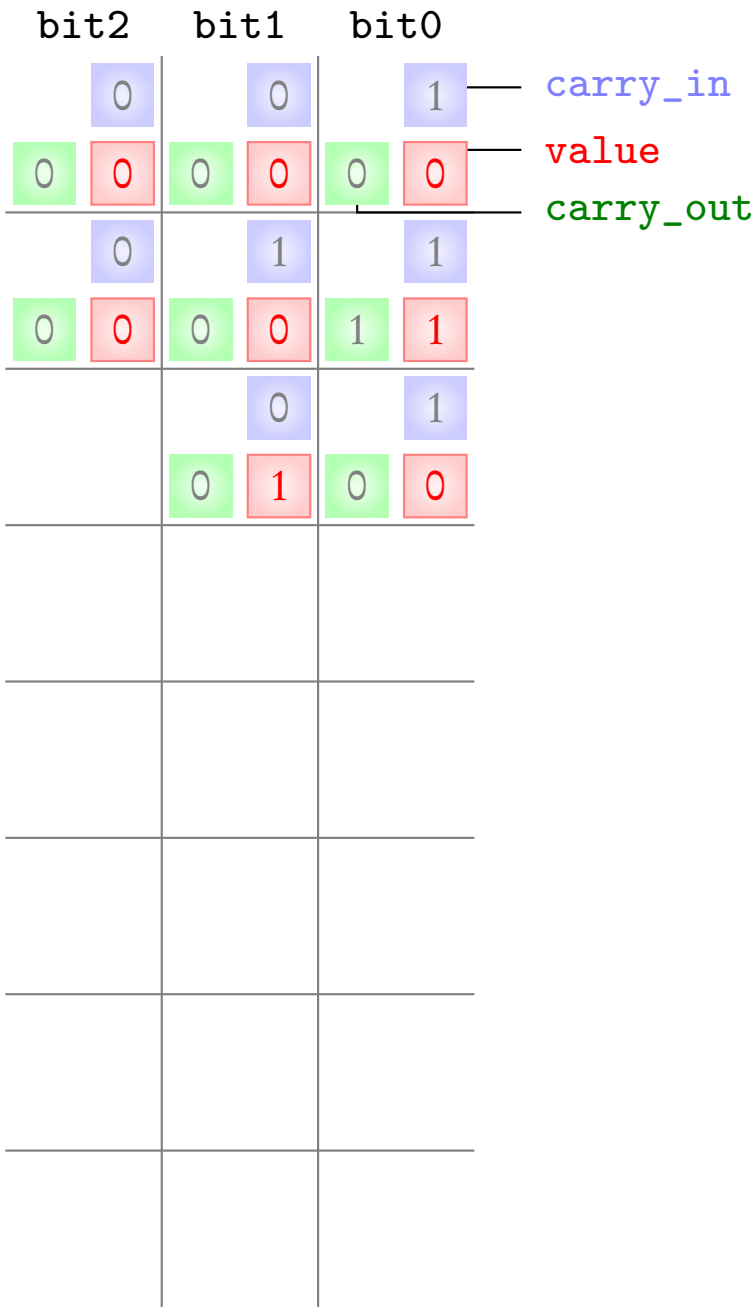

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


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



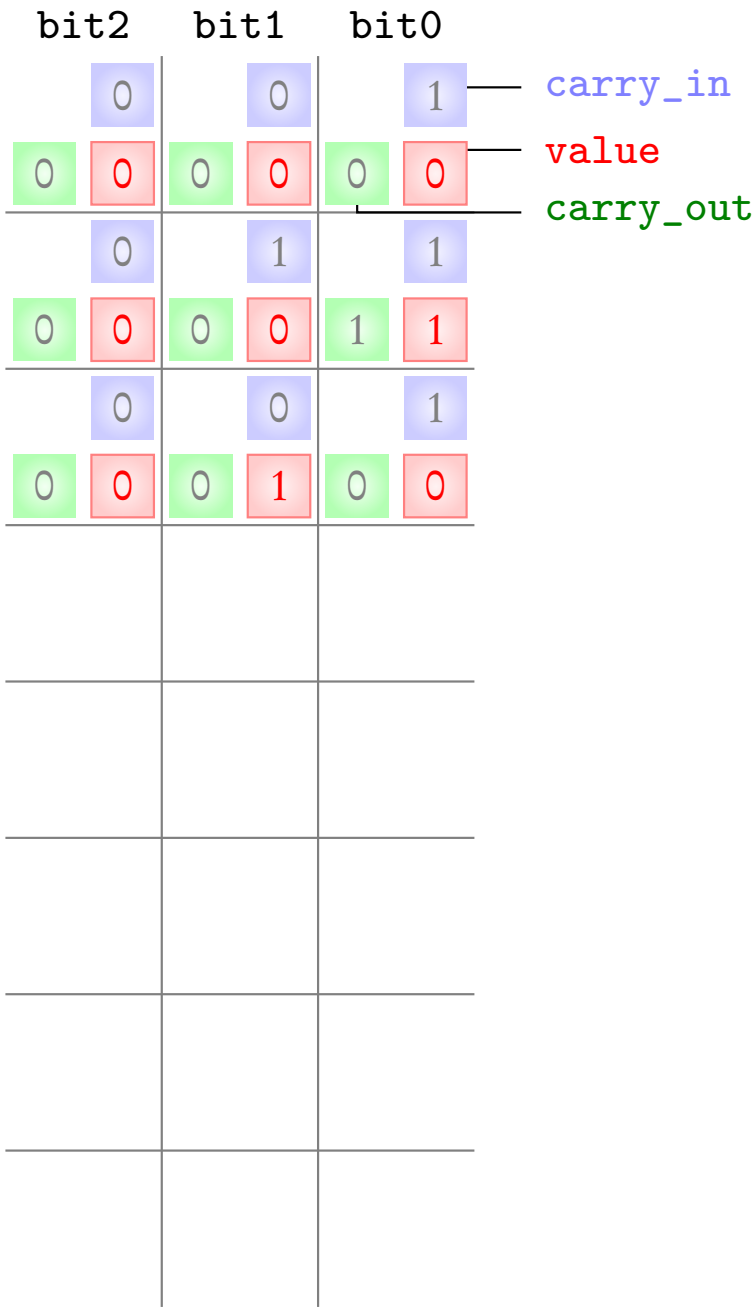

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


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



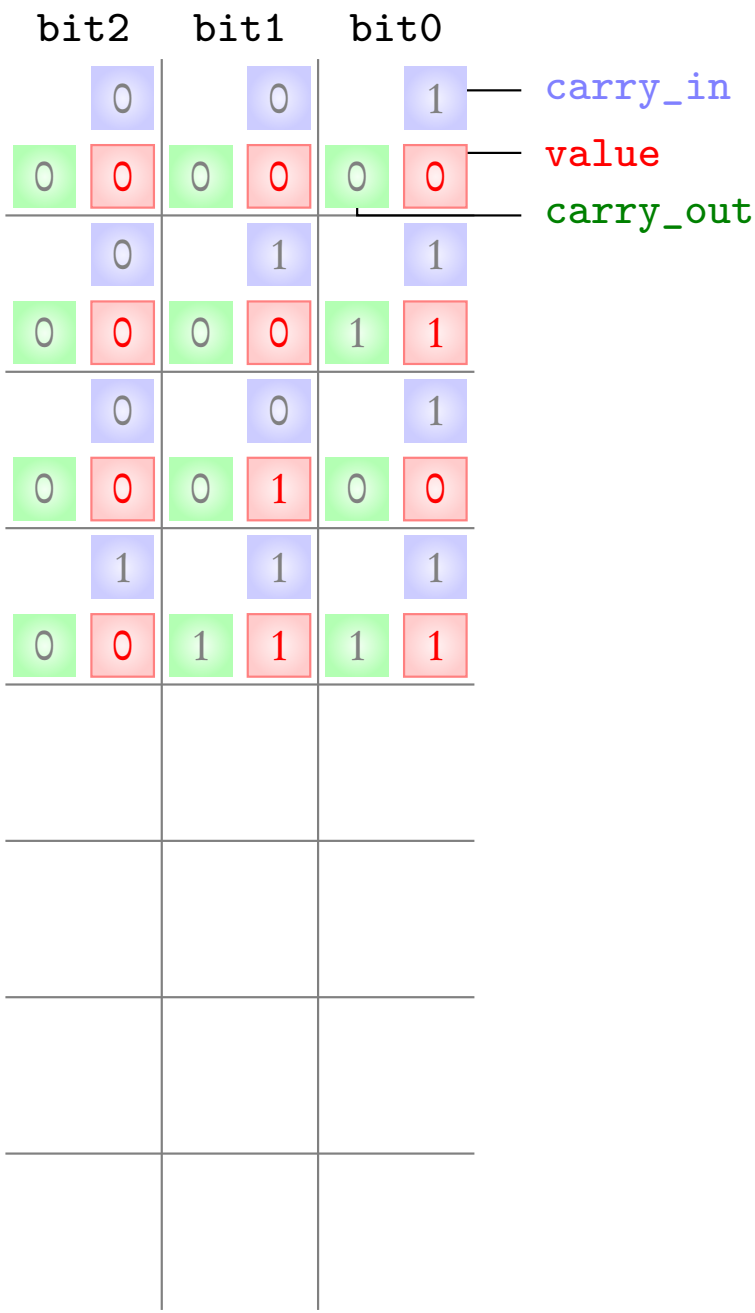

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


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



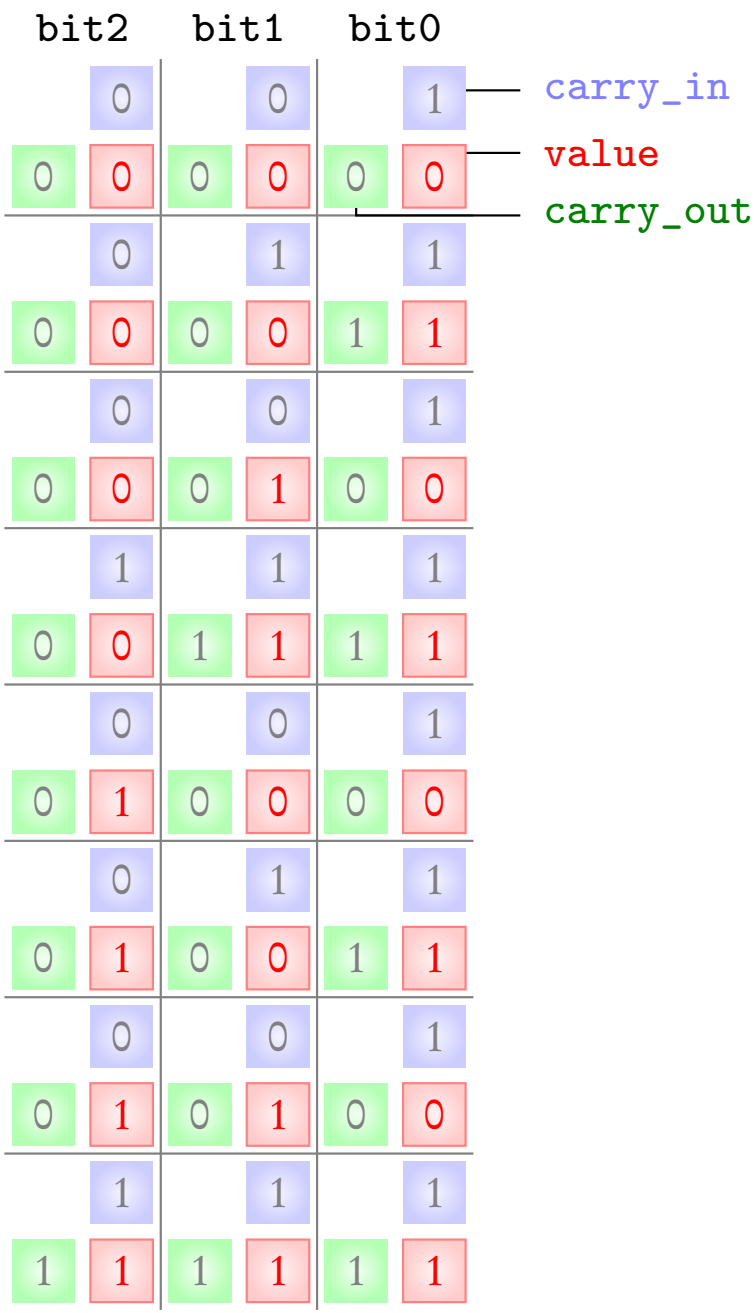

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


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



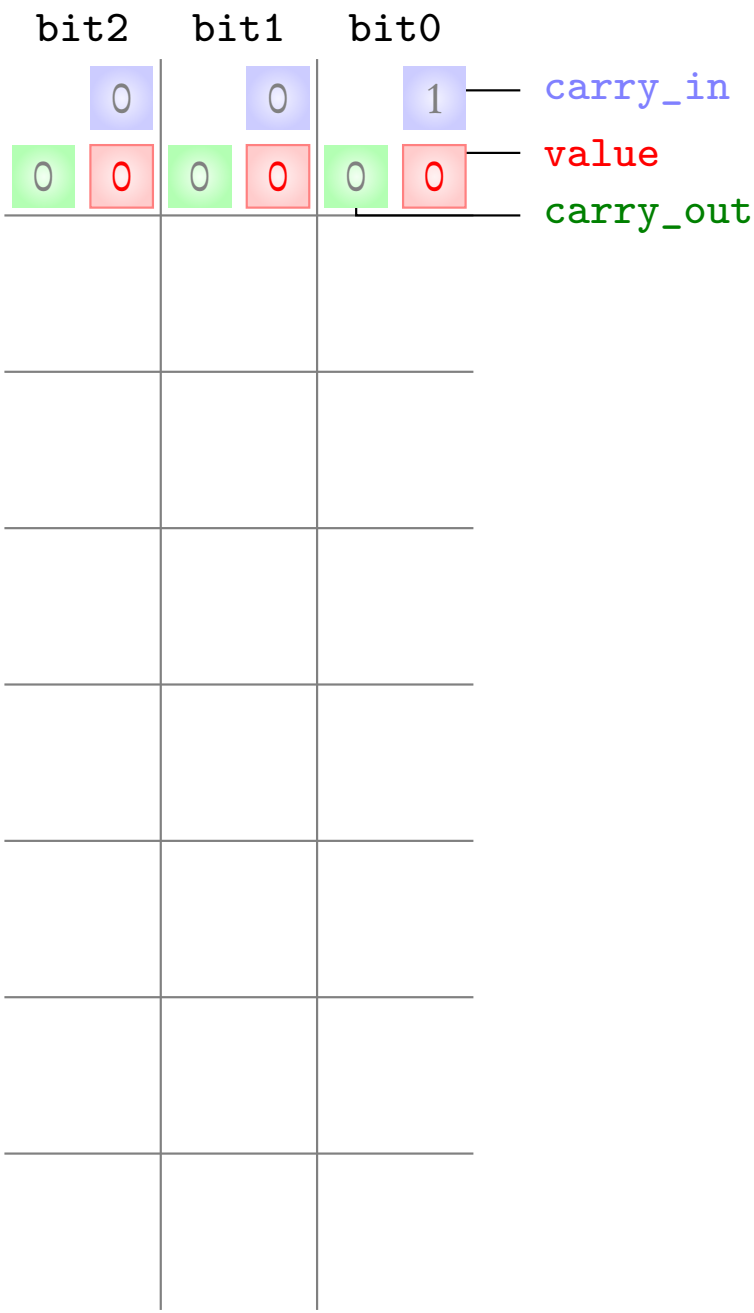

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


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

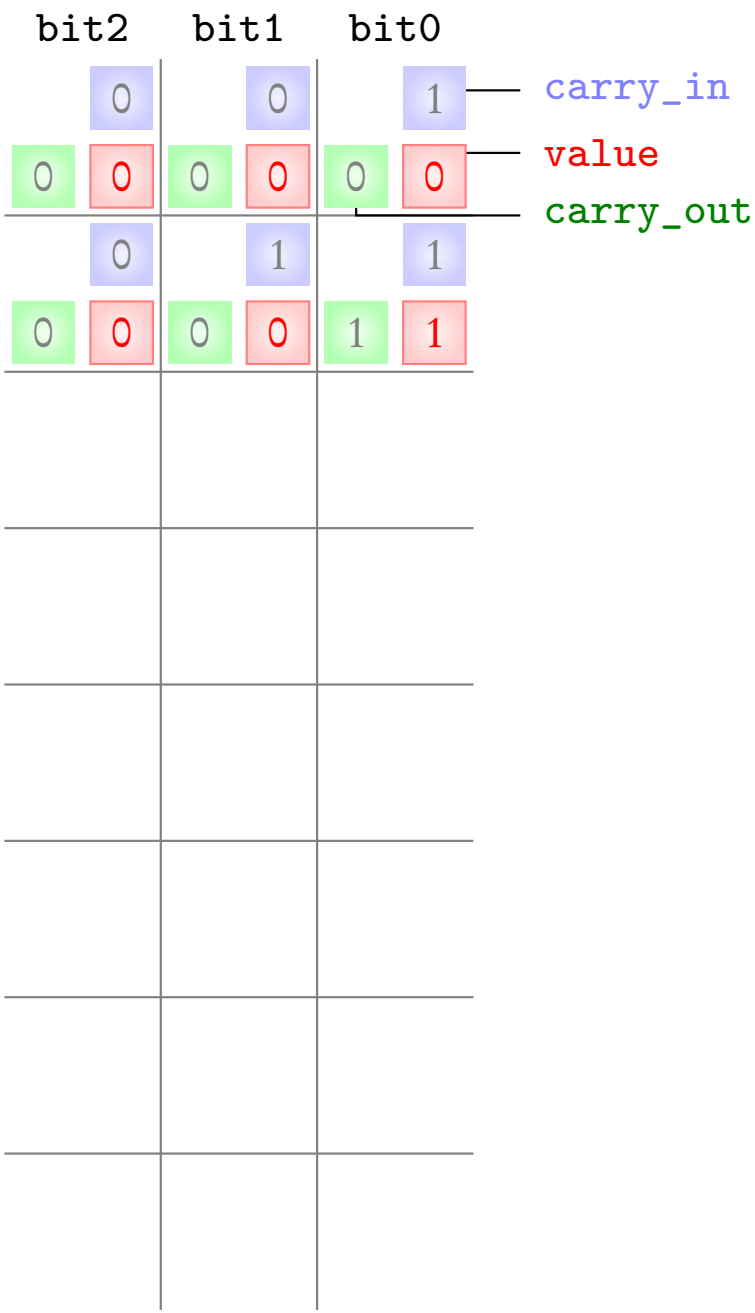


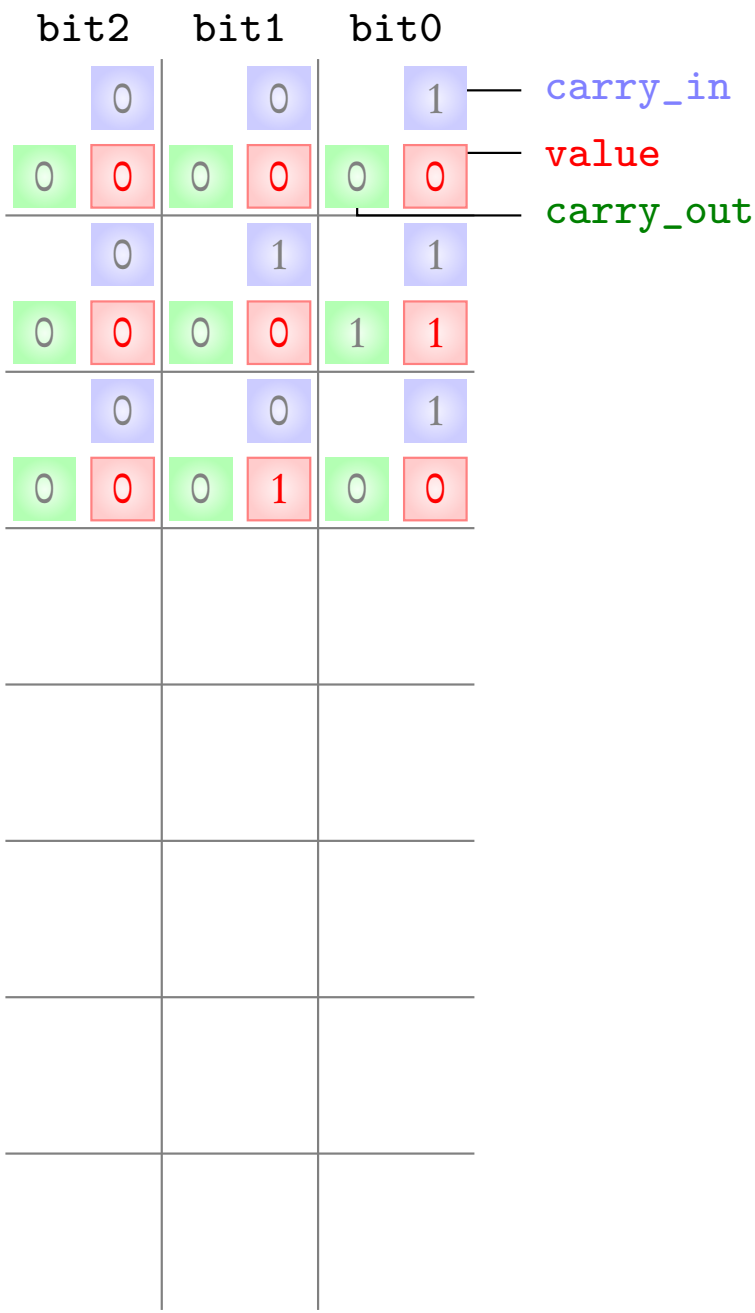

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


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

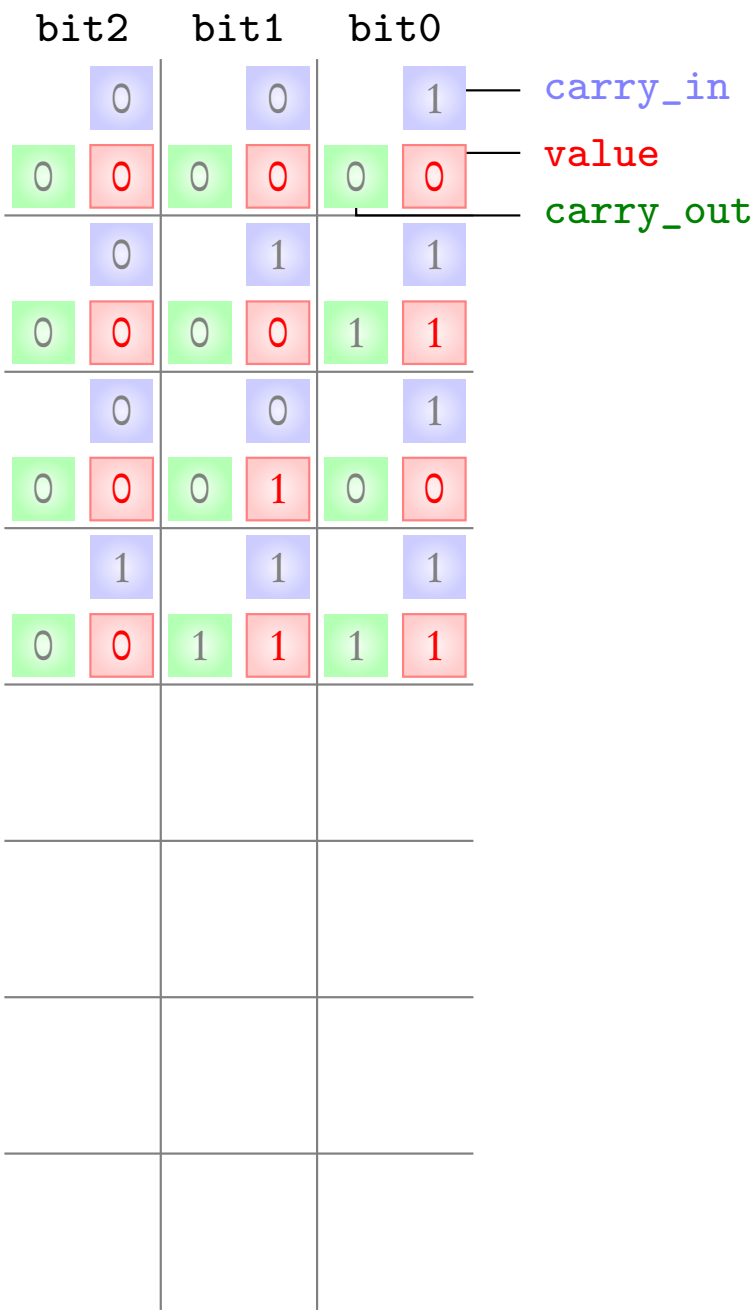


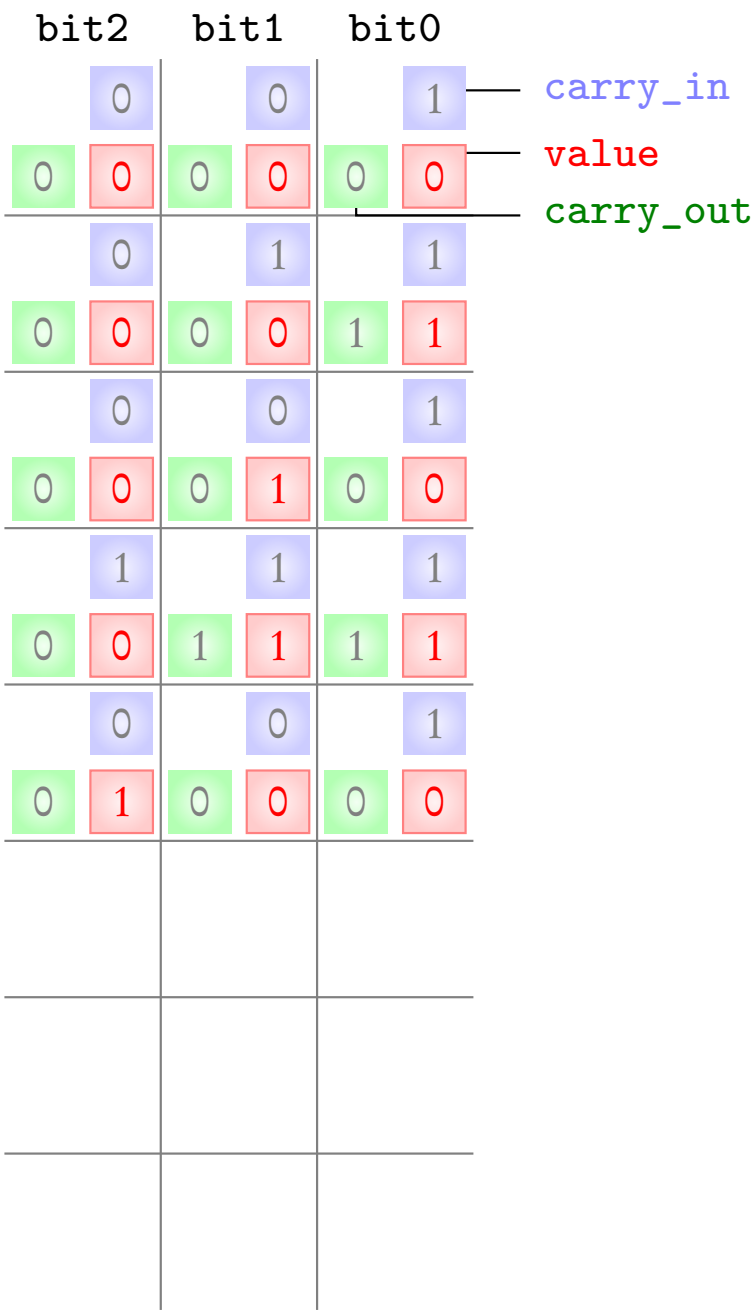

```
MODULE counter_cell(carry_in)
VAR
    value:boolean;
ASSIGN
    init(value):=FALSE;
    next(value):= value xor carry_in;
DEFINE
    carry_out := carry_in & value;


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

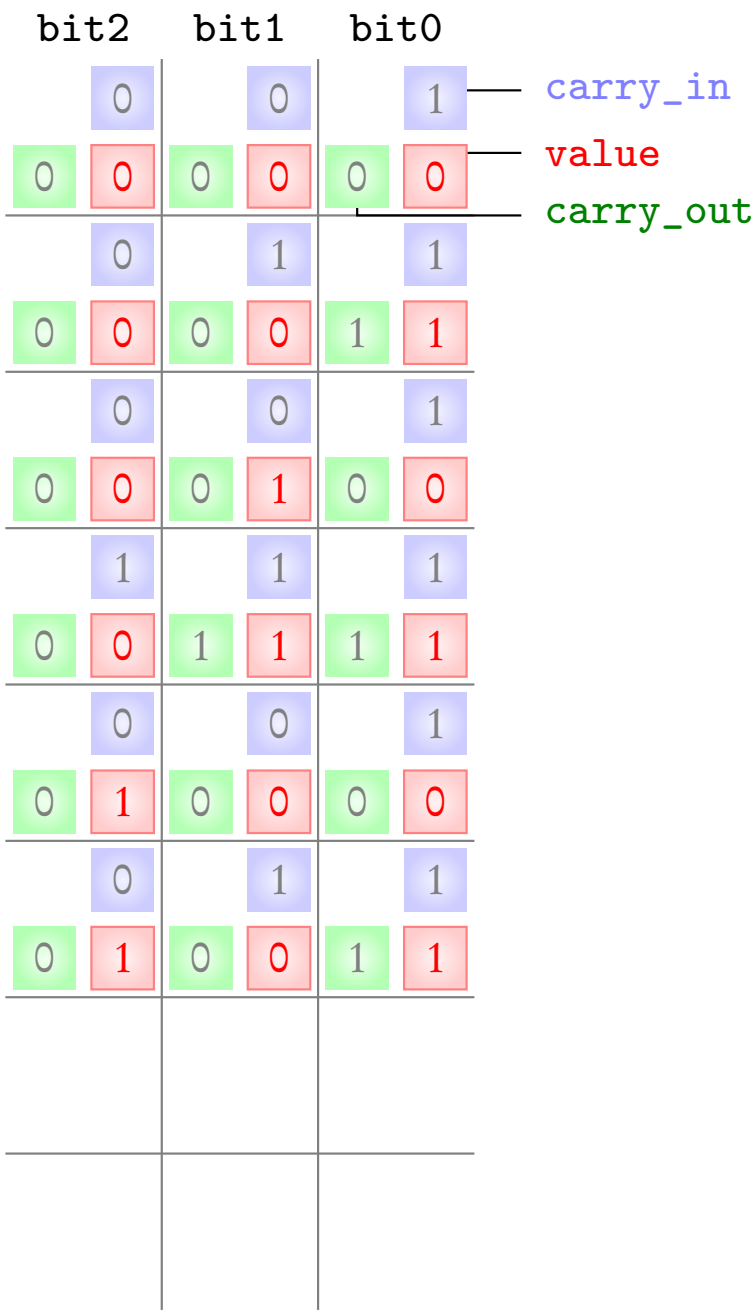

bit2
bit1
bit0
0
0
1
carry_in
0
0
0
0
0
0
value
0
carry_out

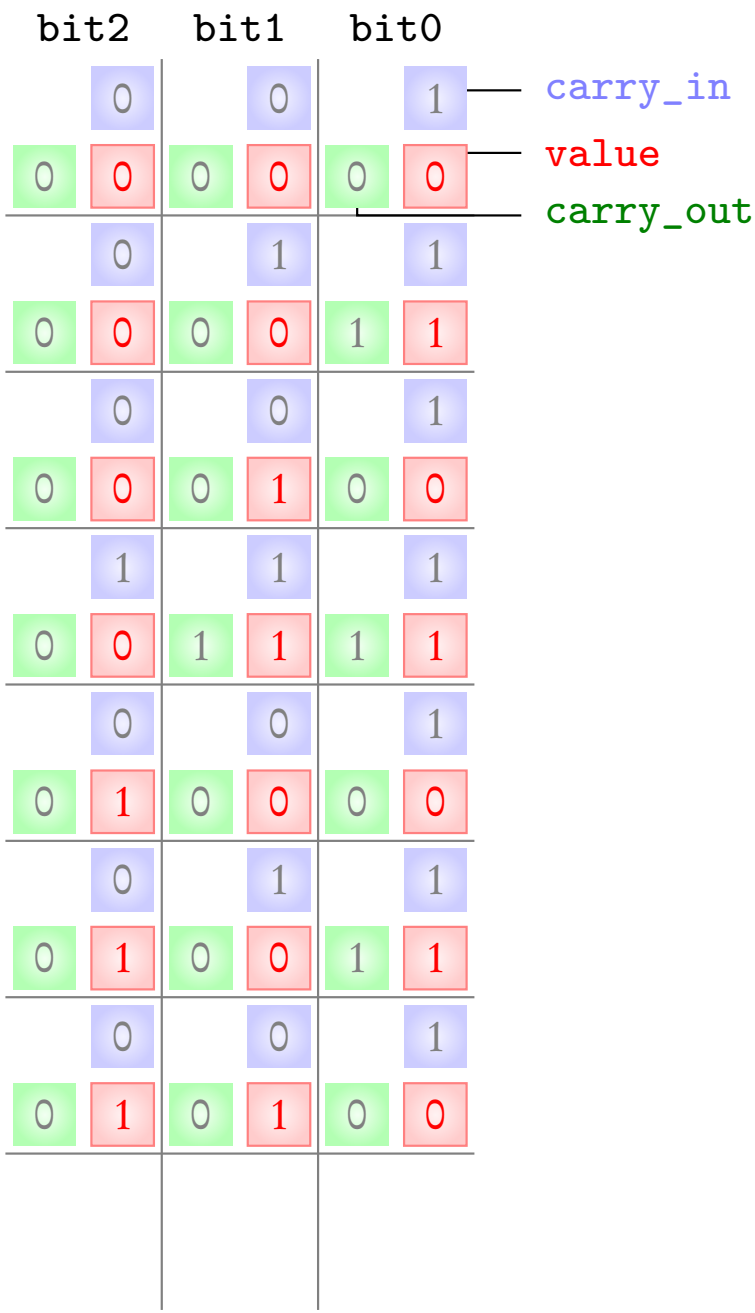

bit2
bit1
bit0
carry_in
value
carry_out
0
0
1
0
0
0
0
0
0
0
1
1
0
0
0
0
1
1

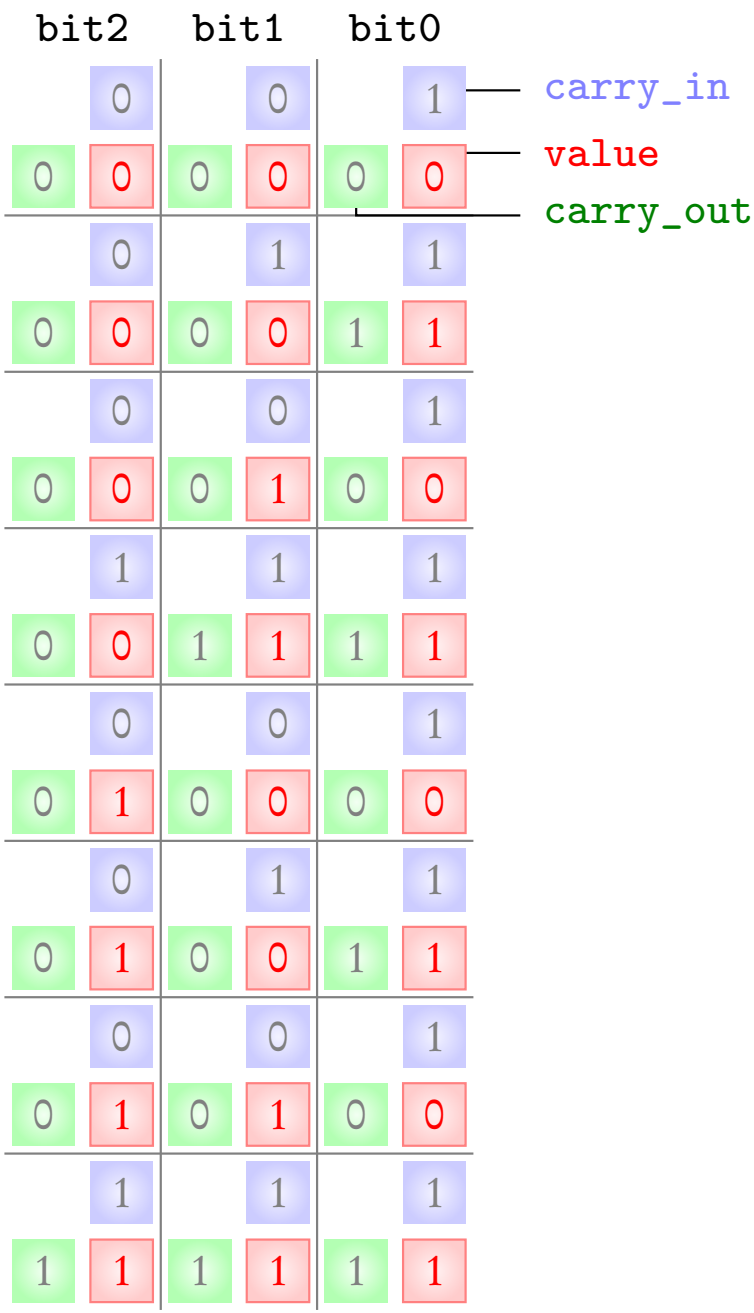

bit2
bit1
bit0
carry_in
value
carry_out
0 0 0 0 0 1
0 0 0 0 0 0
0 0 1 1 1 1
0 0 0 0 1 1
0 0 0 1 0 1
0 0 0 1 0 0

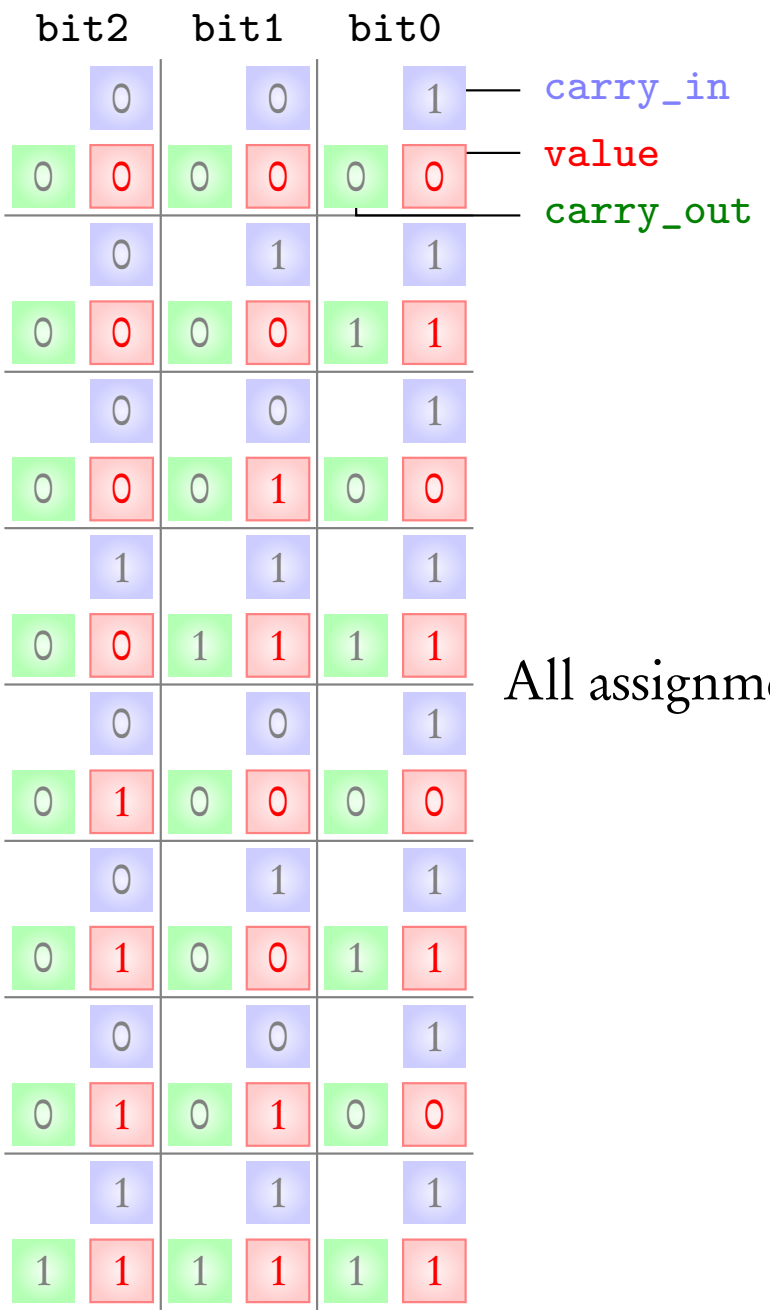

bit2
bit1
bit0
carry_in
value
carry_out


bit2
bit1
bit0
carry_in
value
carry_out


bit2
bit1
bit0
carry_in
value
carry_out


bit2
bit1
bit0
carry_in
value
carry_out


bit2
bit1
bit0
carry_in
value
carry_out



Synchronous composition

All assignments to all `MODULES` occur simultaneously

(more about this later)

Simple circuit

Use of `DEFINE`

Hierarchical designs

Use of `MODULE`

Counter

Synchronous composition of modules

Unit-2: Model-checker NuSMV

B. Srivathsan

Chennai Mathematical Institute

NPTEL-course

July - November 2015

Module 4:

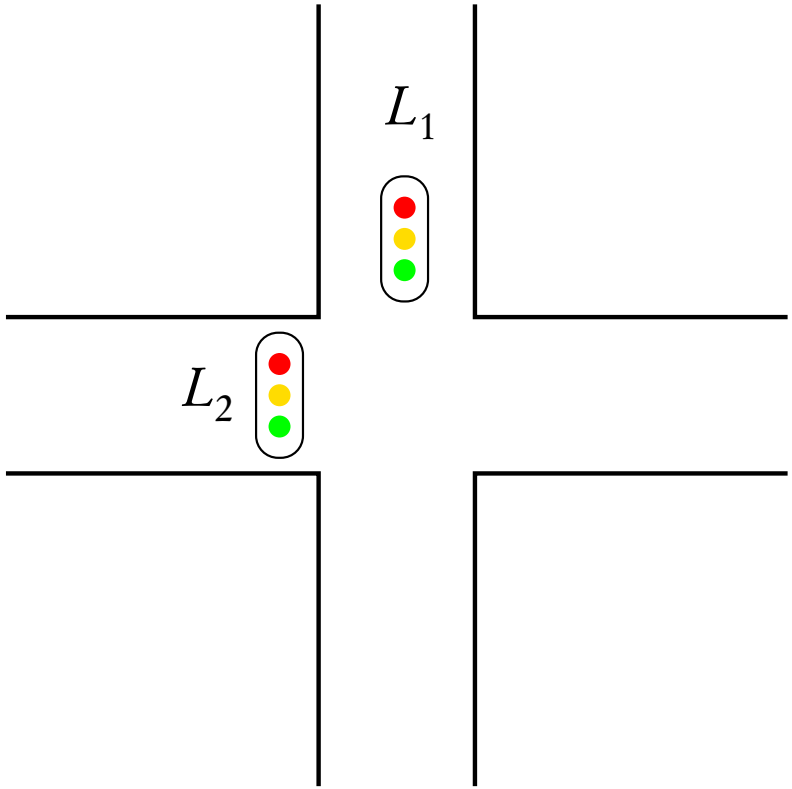
Modeling concurrent systems

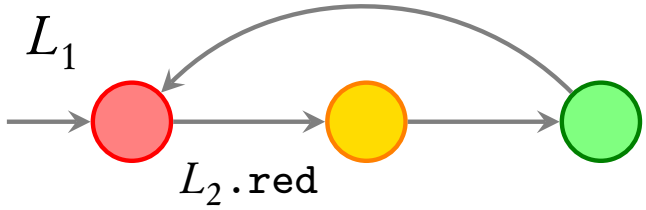
Synchronous vs. Asynchronous systems

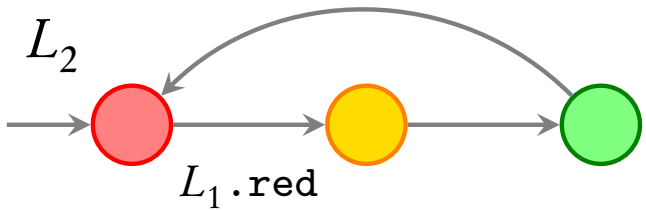
Acknowledgements:

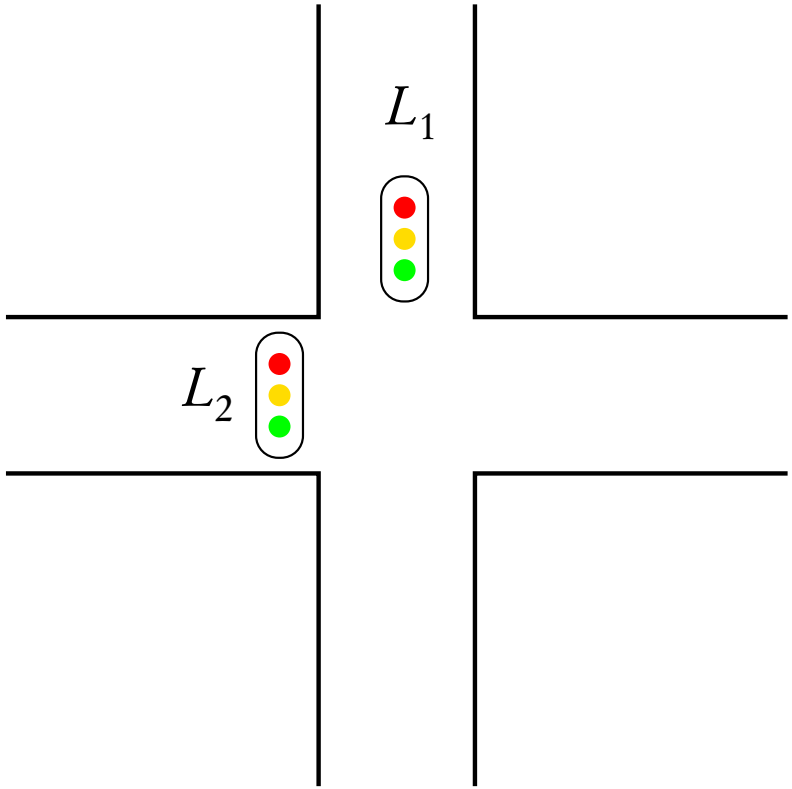
This part of module taken from lecture slides of

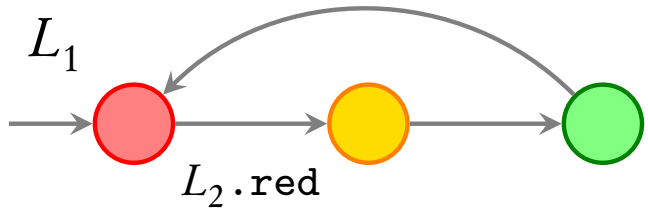
Prof. Supratik Chakraborty, IIT Bombay

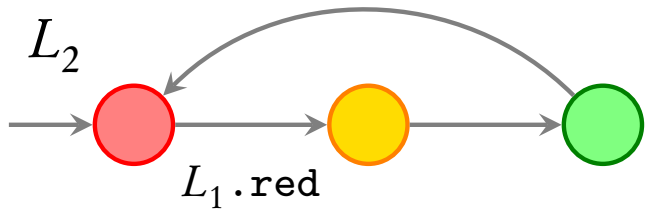

L_1
L_2


L_1
L_2.red


L_2
L_1.red







If a light is **red**, it can **stay red** for an **arbitrary period**

If it goes **yellow**, it should become **green within one cycle**

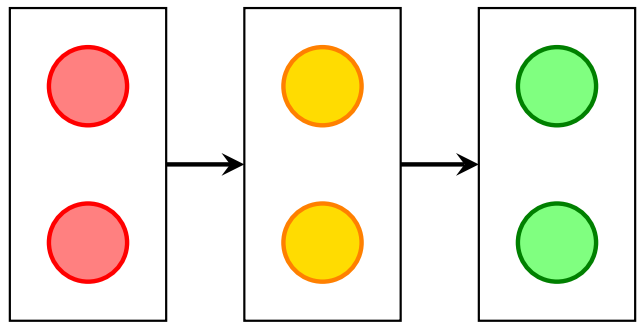
If it is **green**, it can **stay green** for an **arbitrary period**

```
MODULE light(other)
VAR
    state: {r,y,g};
ASSIGN
    init(state) := r;
    next(state) := case
                     state=r & other=r : {r, y};
                     state=y : g;
                     state=g : {g, r};
                     TRUE : state;
                   esac;

MODULE main
VAR
    tl1: light(tl2.state);  tl2: light(tl1.state);
```

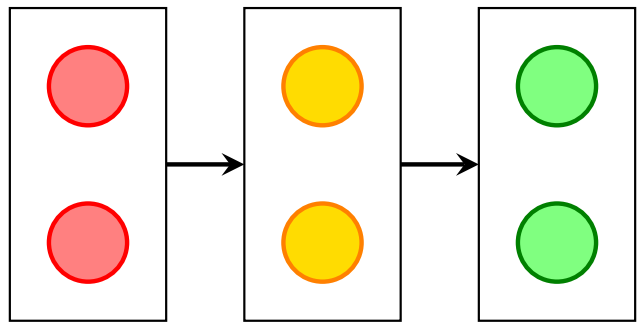
Synchronous composition

```
MODULE light(other)
VAR
    state: {r,y,g};
ASSIGN
    init(state) := r;
    next(state) := case
                     state=r & other=r : {r, y};
                     state=y : g;
                     state=g : {g, r};
                     TRUE : state;
                   esac;

MODULE main
VAR
    tl1: light(tl2.state);  tl2: light(tl1.state);
```

Synchronous composition



Synchronous composition

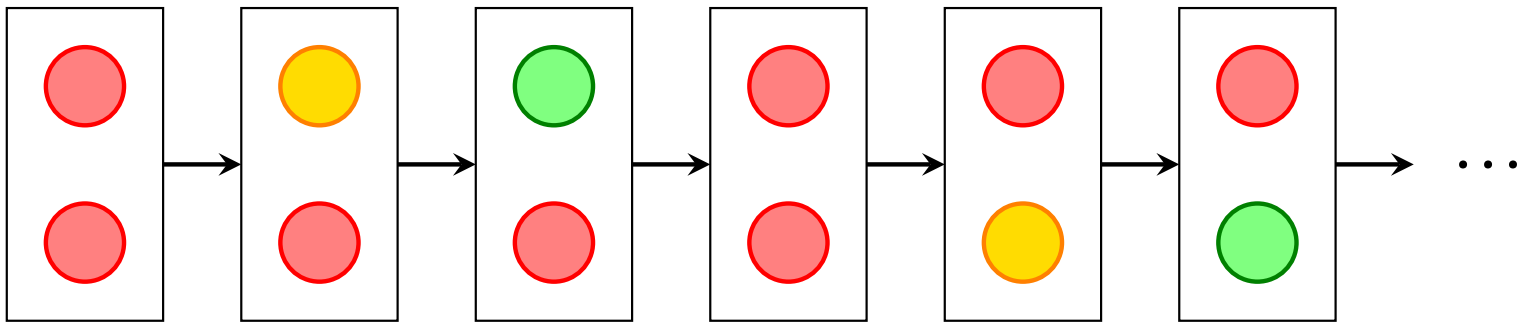


Both lights can **simultaneously become green!**

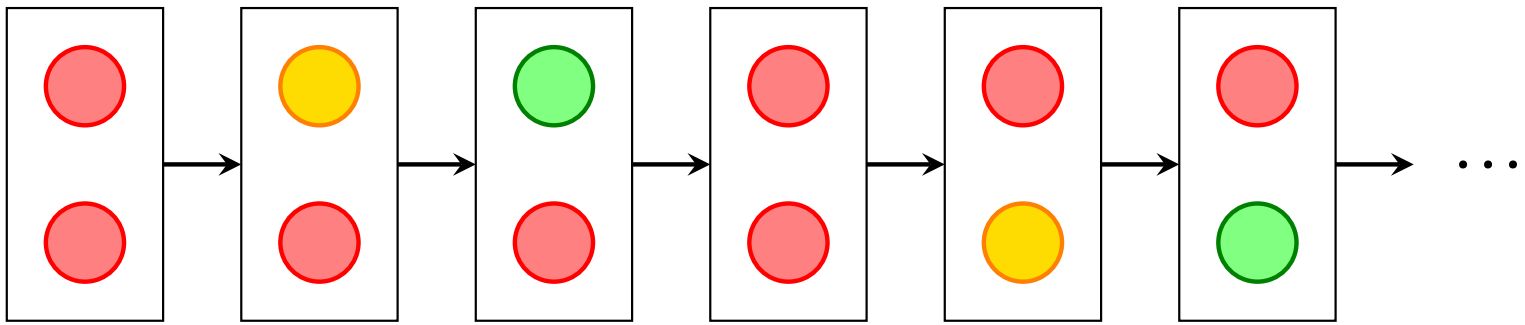
Asynchronous composition

```
MODULE light(other)
VAR
    state: {r,y,g};
ASSIGN
    init(state) := r;
    next(state) := case
                     state=r & other=r : {r, y};
                     state=y : g;
                     state=g : {g, r};
                     TRUE : state;
                   esac;

MODULE main
VAR
    tl1: process light(tl2.state);
    tl2: process light(tl1.state);
```

Asynchronous composition



Asynchronous composition



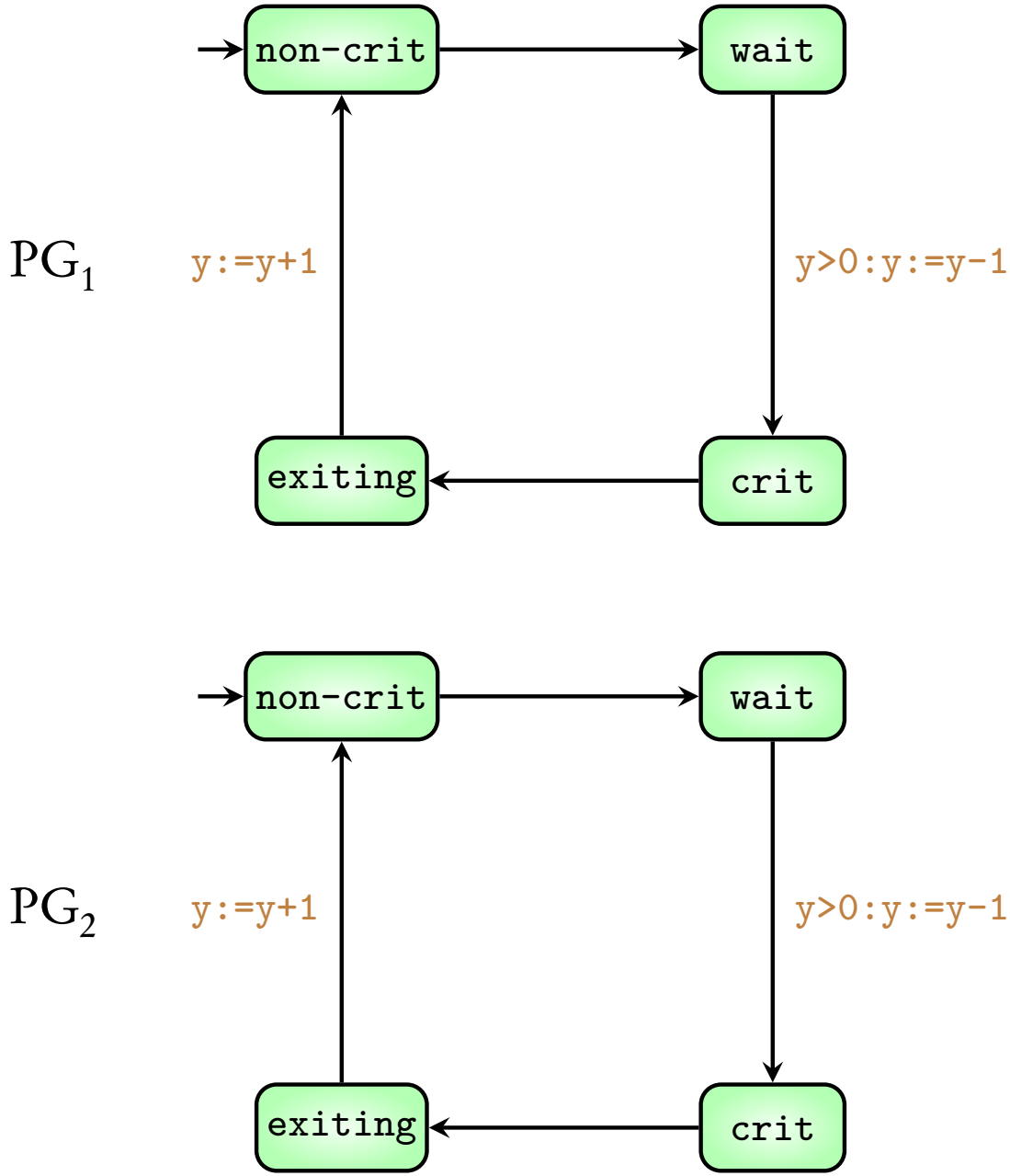
Only one light can become **green** at a time

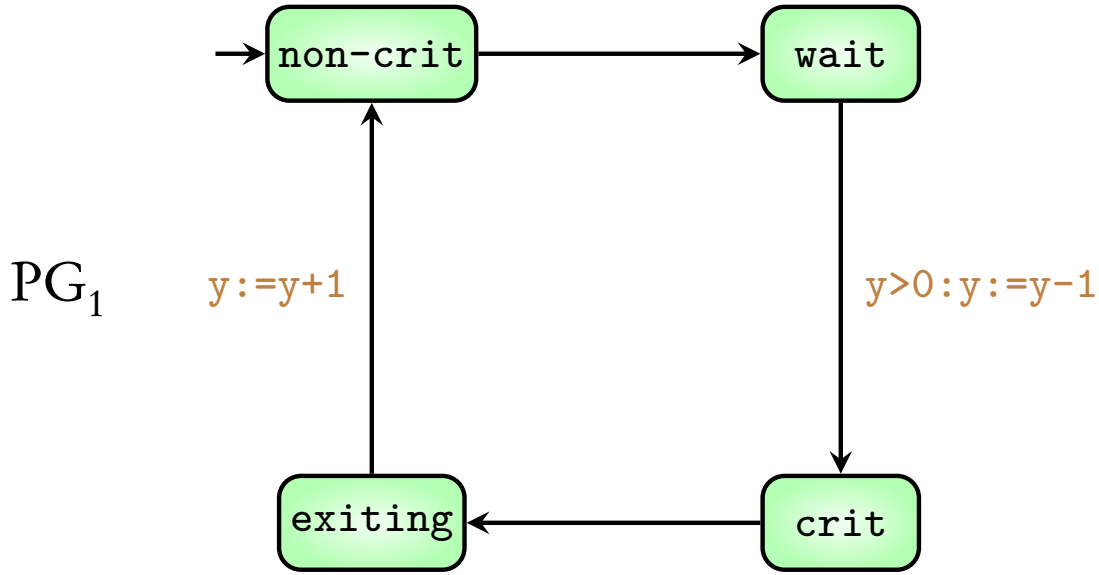
- **Synchronous:**
  - all assignments to all modules made **simultaneously**
  - suitable when all modules are synchronized to a **global clock**
- **Asynchronous:**
  - execution of modules is **interleaved**
  - at a time, **only one** module executes
  - choice of next module to be executed is **non-deterministic**
  - suitable when **no assumptions** can be made **about communication delay** between modules

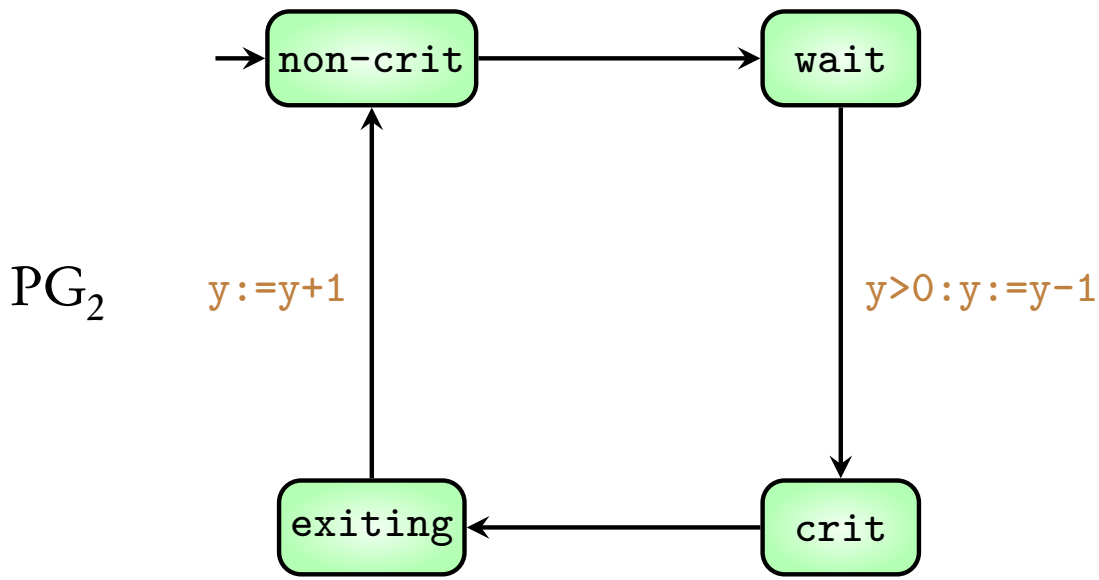
Synchronous
vs.
Asynchronous
systems

Synchronous vs. Asynchronous systems

Mutual Exclusion


PG1
non-crit
wait
y:=y+1
y>0:y:=y-1
exiting
crit
PG2
non-crit
wait
y:=y+1
y>0:y:=y-1
exiting
crit


PG1
non-crit
wait
y:=y+1
y>0:y:=y-1
exiting
crit


PG2
non-crit
wait
y:=y+1
y>0:y:=y-1
exiting
crit

NuSMV demo

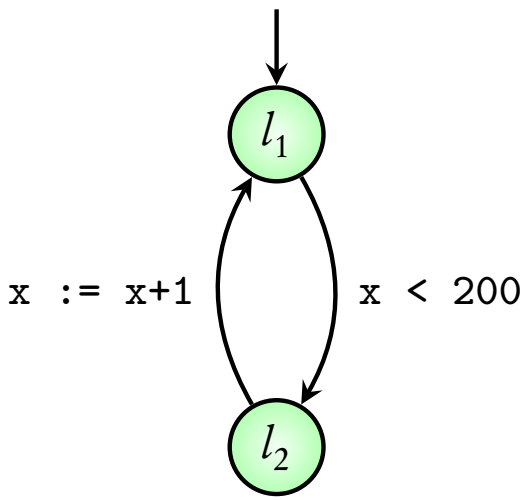
Synchronous vs. Asynchronous systems

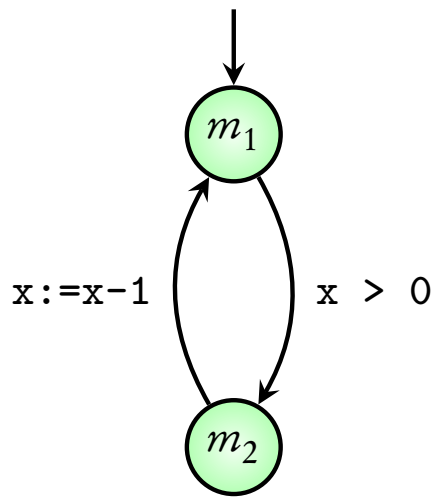
Mutual Exclusion

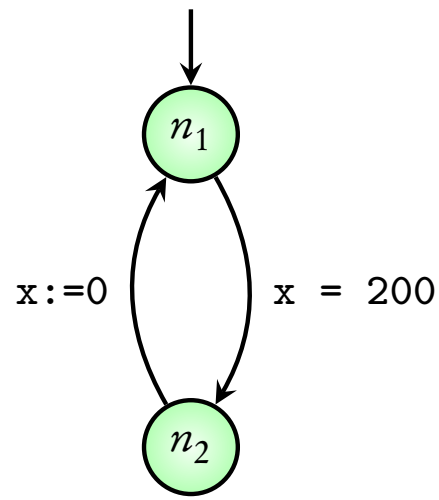
```
while x < 200
    x := x+1
```

```
while x>0
    x := x-1
```

```
while x=200
    x := 0
```



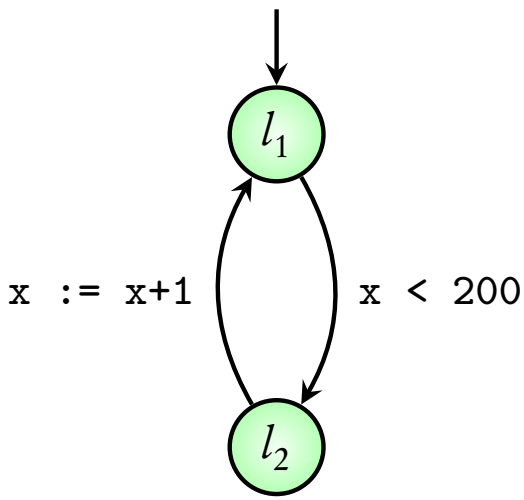


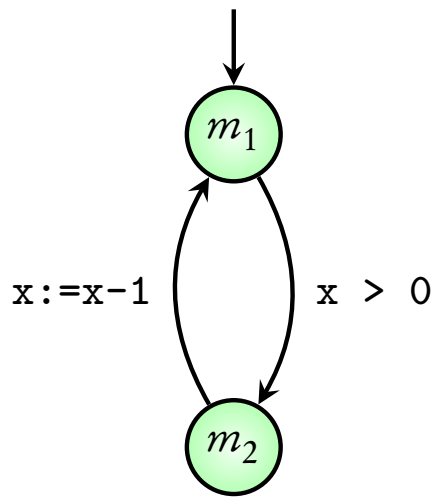


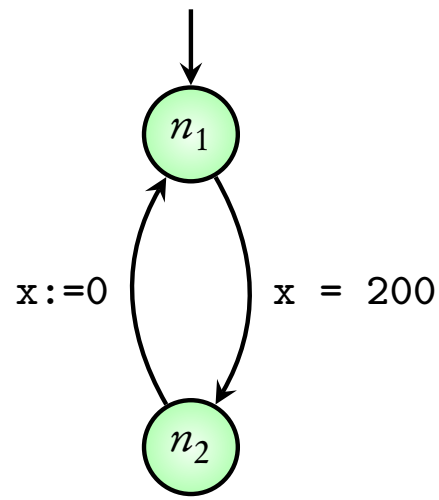
```
while x < 200
    x := x+1
```

```
while x>0
    x := x-1
```

```
while x=200
    x := 0
```







NuSMV demo

Synchronous vs. Asynchronous systems

Mutual Exclusion

Concurrent programs example

Unit-2: Model-checker NuSMV

B. Srivathsan

Chennai Mathematical Institute

NPTEL-course

July - November 2015

Summary

- **Module 1:** Role of model-checking tools
- **Module 2:** Simple verification examples in NuSMV
- **Module 3:** Hardware circuits in NuSMV
- **Module 4:** Modeling concurrent systems in NuSMV

Important concepts: G and F requirements, Synchronous and asynchronous composition