Modeling code behaviour

Relies and uses
Slides from B.
Srivathsan

Chennai Mathematical Institute

Outline

- **Module 1:** Modeling simple code
- **Module 2:** Modeling hardware circuits
- **Module 3:** Modeling data dependent programs
- **Module 4:** Modeling concurrent systems

Module 1:

Modeling code behaviour

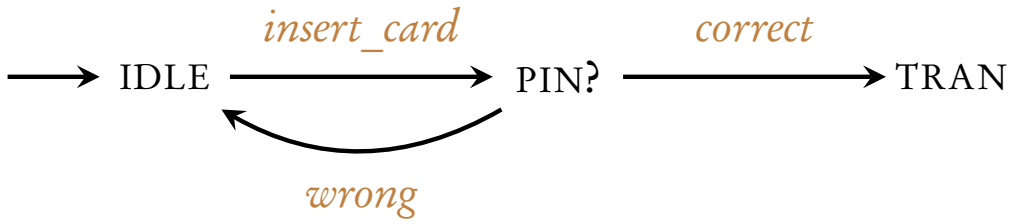

ATM

$\longrightarrow$ IDLE

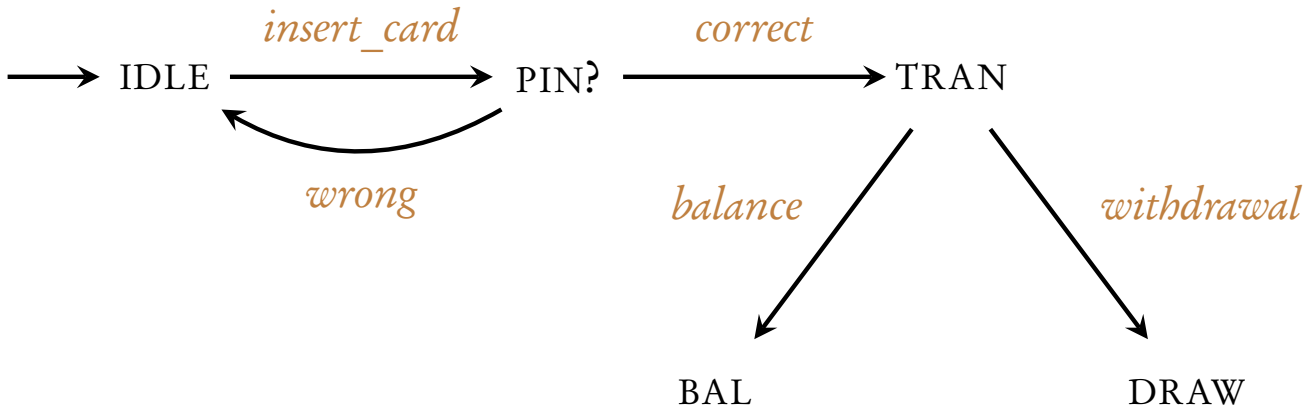

ATM


insert_card
IDLE
PIN?

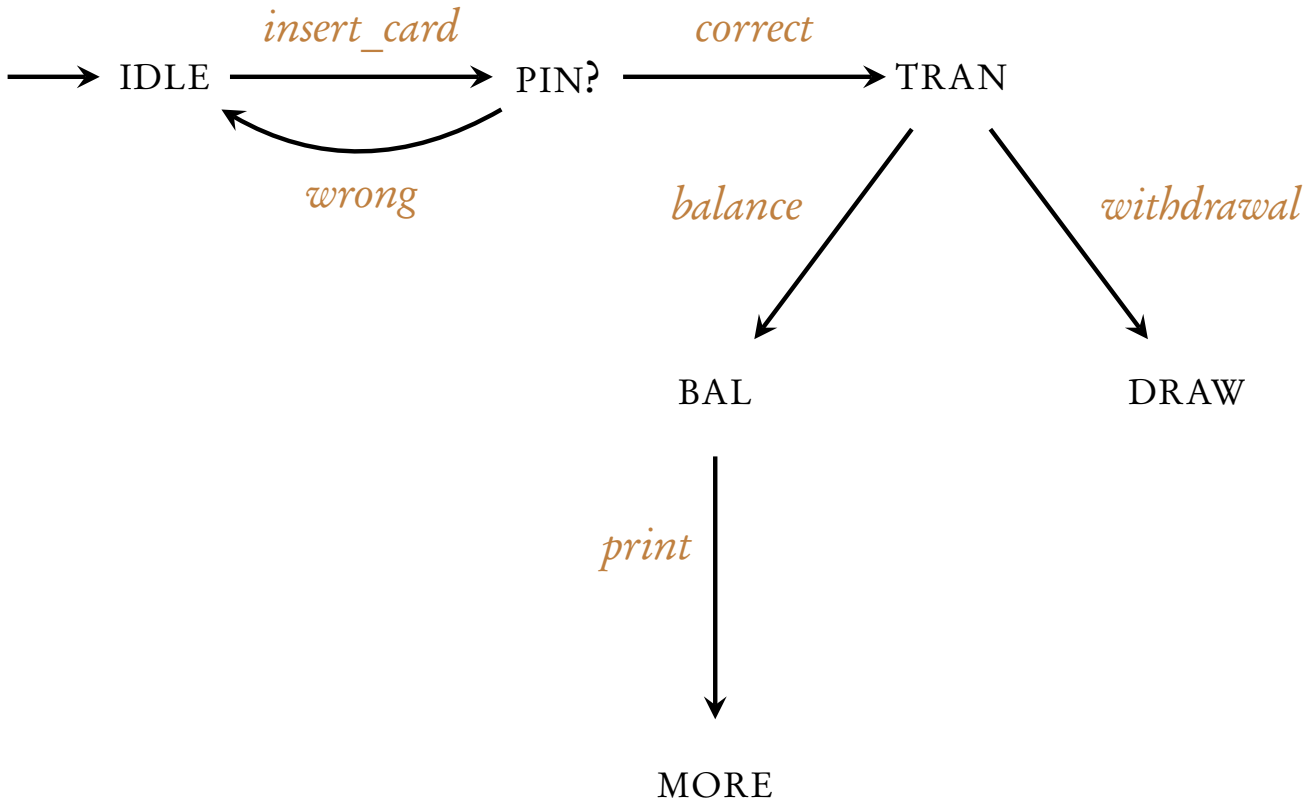

ATM


IDLE
insert_card
PIN?
correct
TRAN
wrong

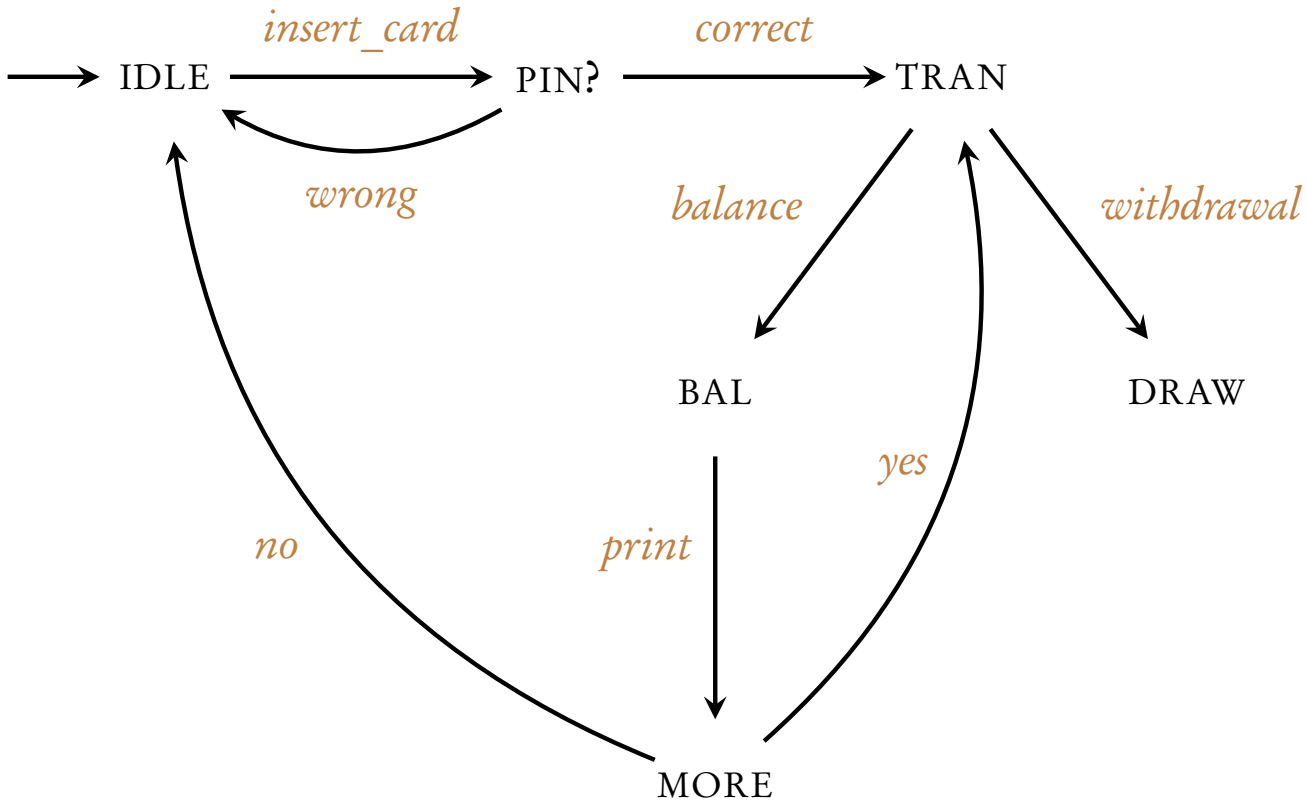

ATM


IDLE
insert_card
PIN?
correct
TRAN
wrong
balance
withdrawal
BAL
DRAW

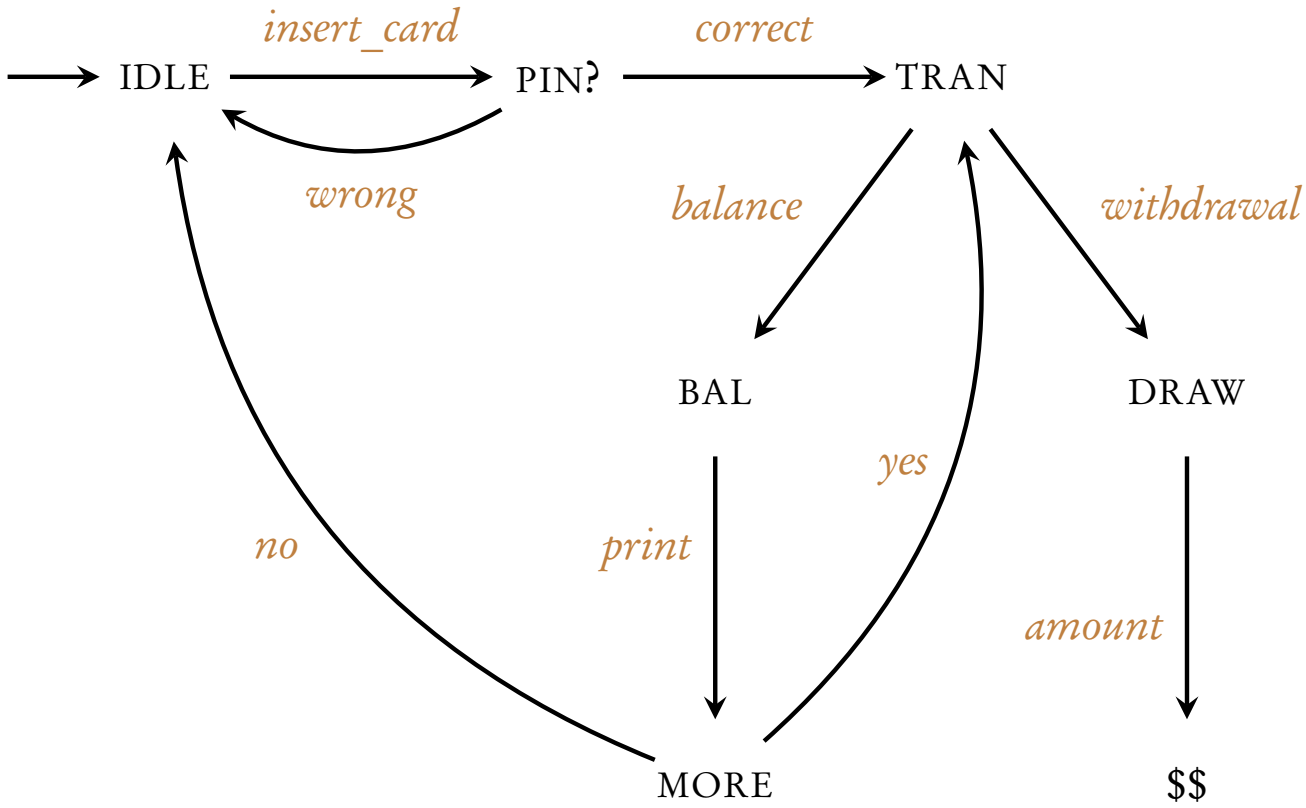

ATM


IDLE
insert_card
PIN?
correct
TRAN
wrong
balance
withdrawal
BAL
DRAW
print
MORE

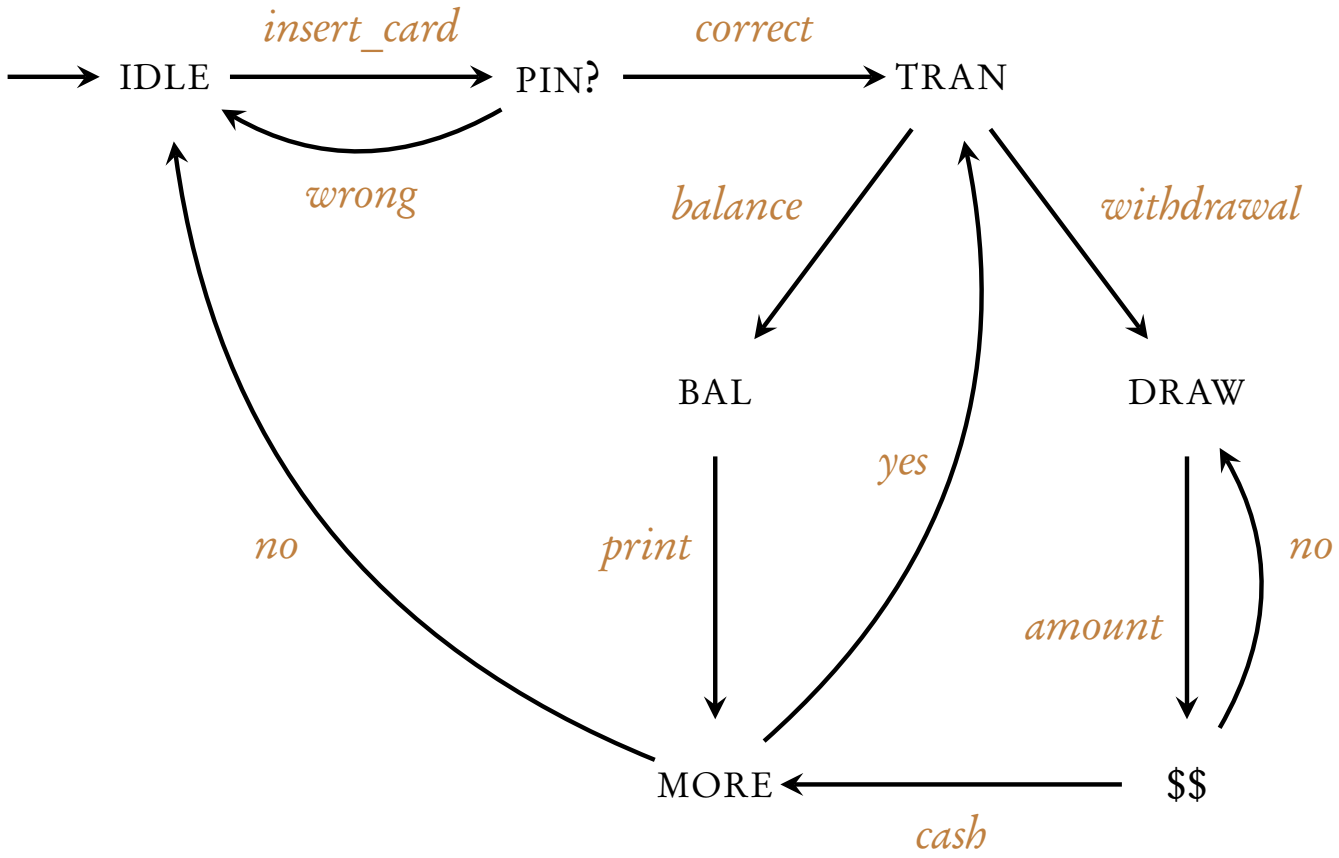

ATM


IDLE
PIN?
TRAN
BAL
DRAW
MORE
insert_card
correct
wrong
balance
withdrawal
yes
no
print

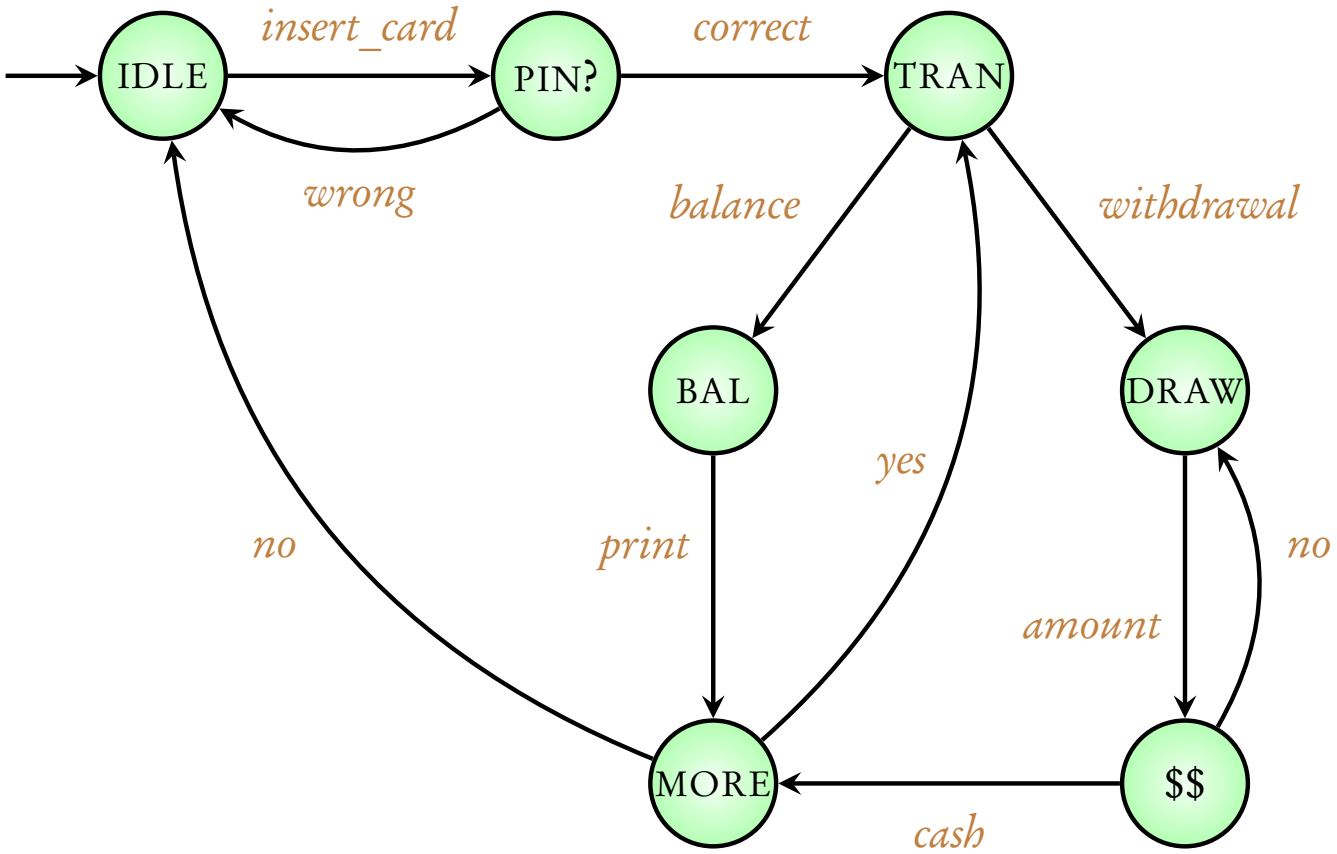

ATM

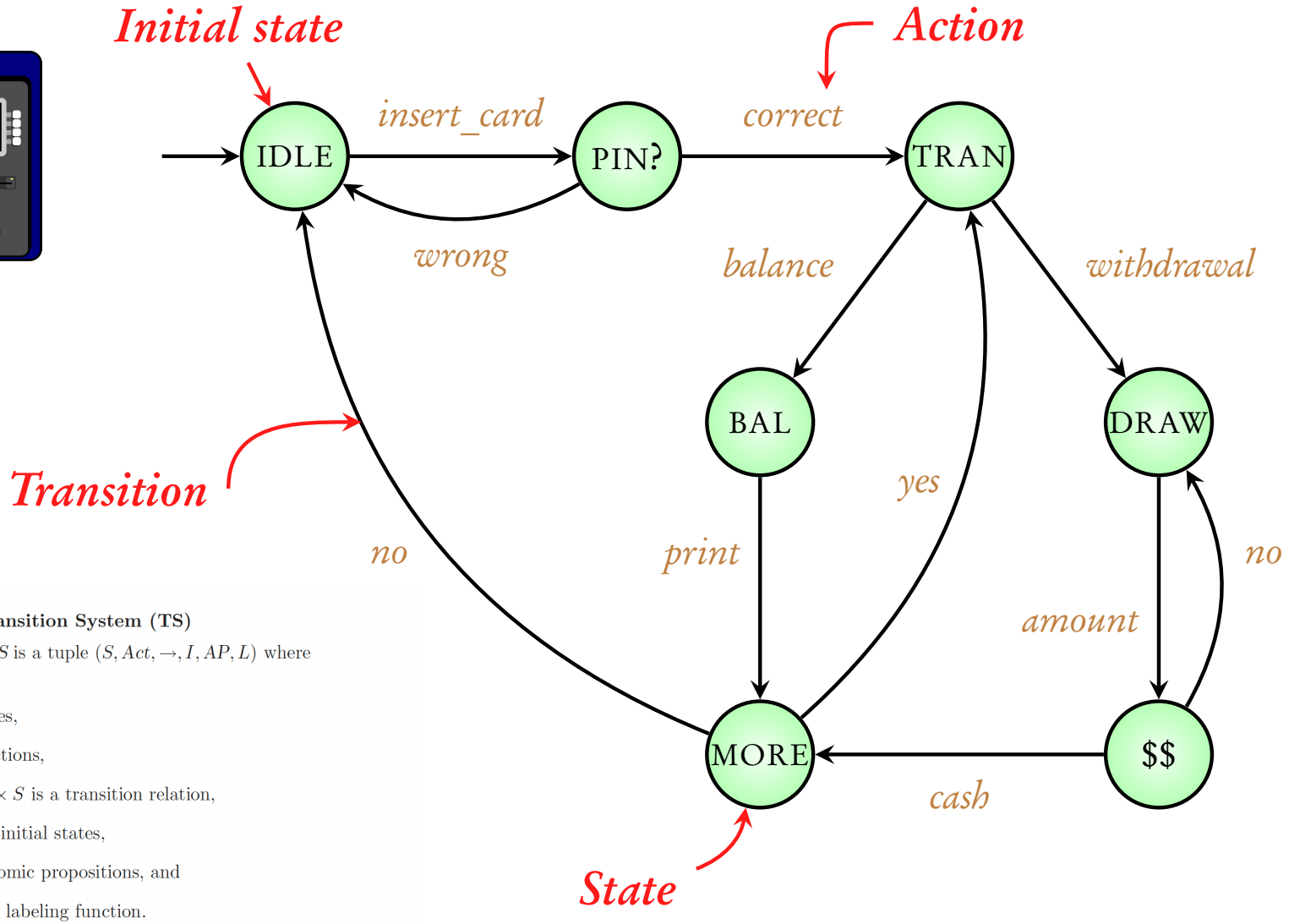

IDLE
PIN?
TRAN
BAL
DRAW
MORE
$$
insert_card
correct
wrong
balance
withdrawal
yes
no
print
amount


ATM


IDLE
PIN?
TRAN
BAL
DRAW
MORE
$$
insert_card
correct
wrong
balance
withdrawal
yes
no
print
amount
no
cash


ATM


IDLE
PIN?
TRAN
BAL
DRAW
MORE
$$
insert_card
correct
wrong
balance
withdrawal
yes
no
print
amount
no
cash



Definition 2.1. Transition System (TS)

A *transition system* TS is a tuple $(S, Act, \rightarrow, I, AP, L)$ where

- S is a set of states,
- Act is a set of actions,
- $\longrightarrow \subseteq S \times Act \times S$ is a transition relation,
- $I \subseteq S$ is a set of initial states,
- AP is a set of atomic propositions, and
- $L : S \rightarrow 2^{AP}$ is a labeling function.

TS is called *finite* if S, Act, and AP are finite.

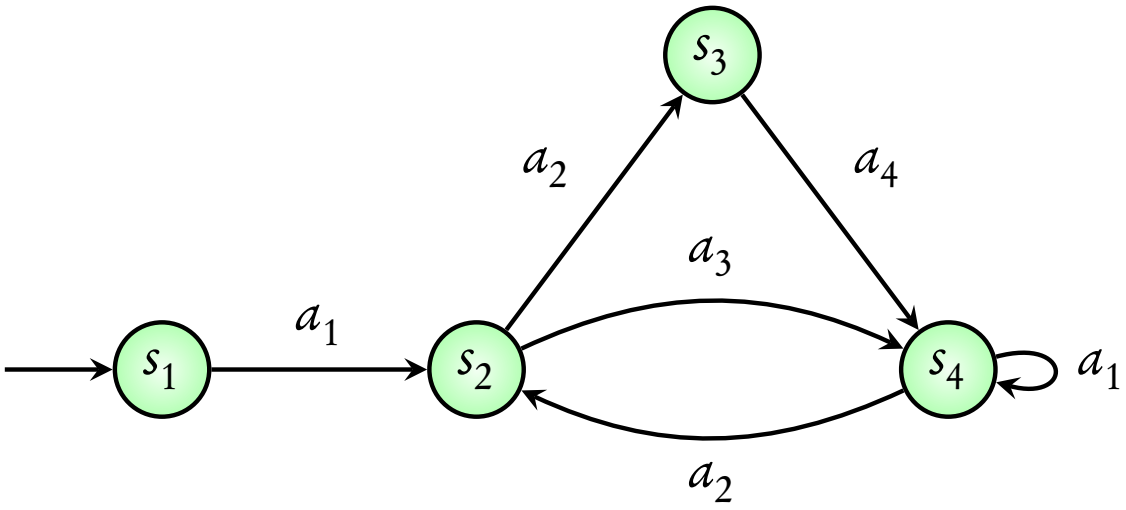
The labeling function L relates a set L(s) $\in$ 2AP of atomic propositions to any state s.

L(s) intuitively stands for exactly those atomic propositions a $\in$ AP which are satisfied by state s.

Given that Φ is a propositional logic formula,
then s satisfies the formula Φ if
the evaluation induced by L(s) makes the formula Φ true; that is:

$$s \models \Phi \quad \text{iff} \quad L(s) \models \Phi.$$

Transition system



States

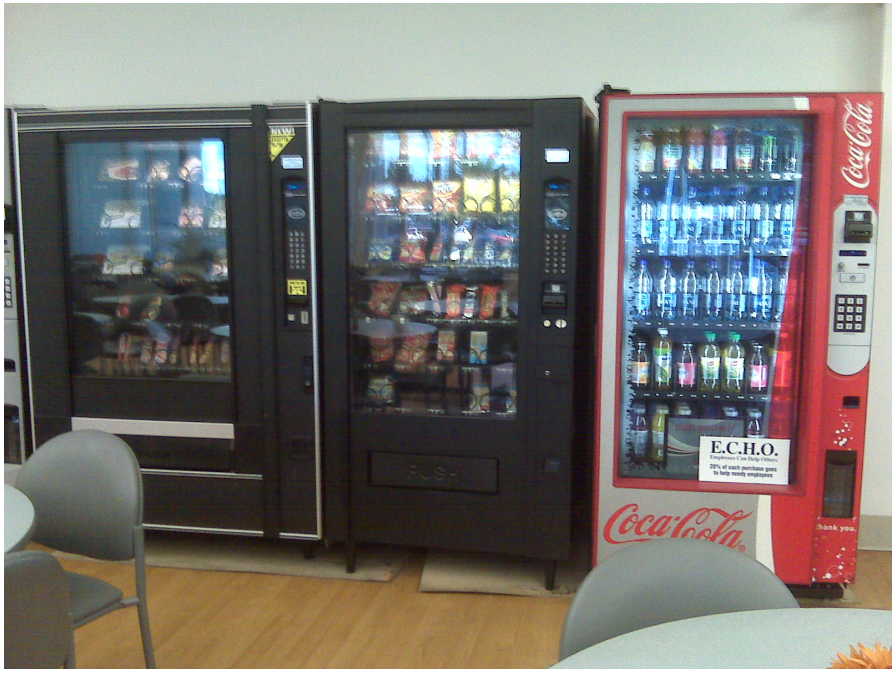
Actions

Transitions

Initial state

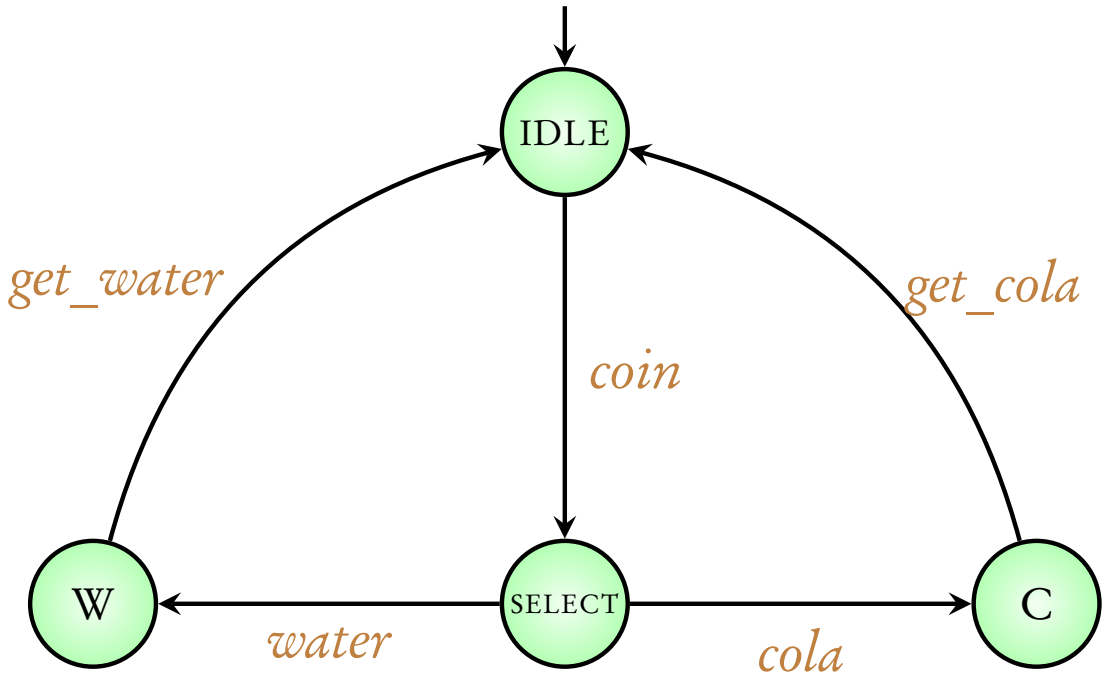
Other names: Finite-state machines, State-transition graphs

Modeling a vending machine

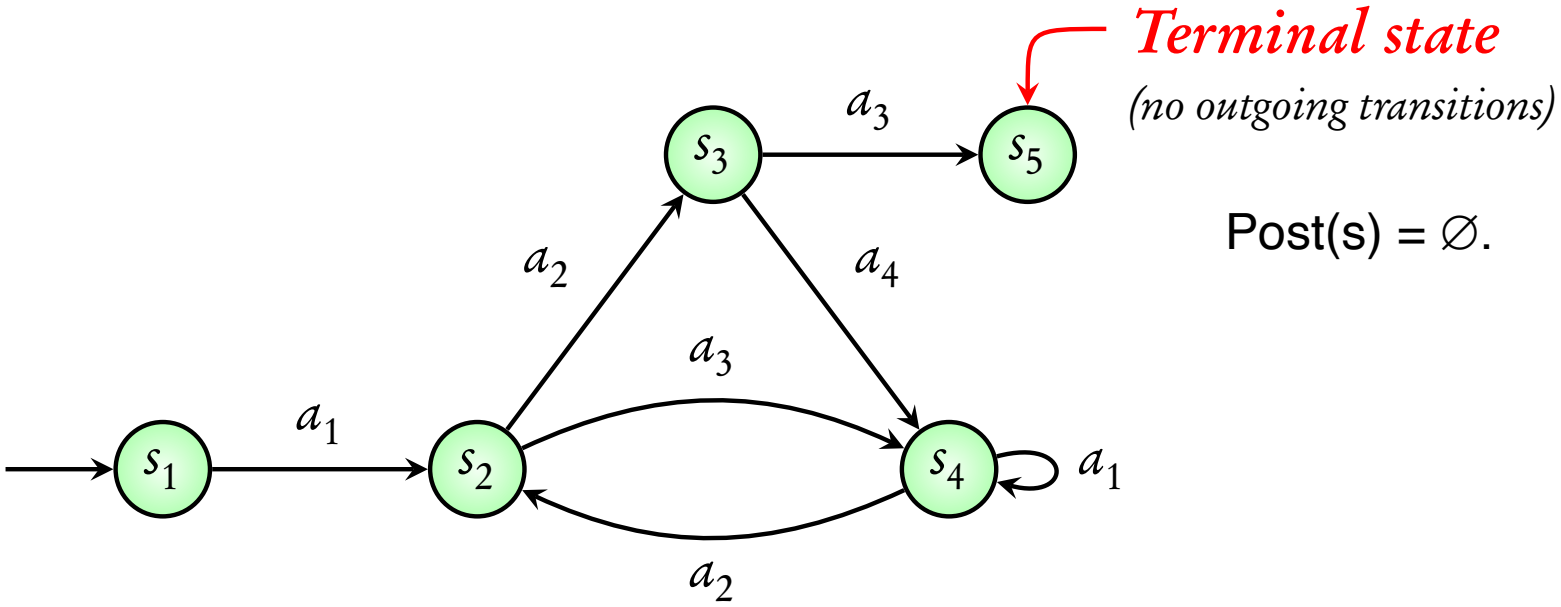


"Vending machines at hospital" by PCHS-NJROTC - Own work
Licensed under CC BY - SA 3.0 via Wikimedia Commons

L(idle) = ∅, L(W) = L(C) = { paid , drink }, L(select) = { paid }.



Coming next: some **terminology**



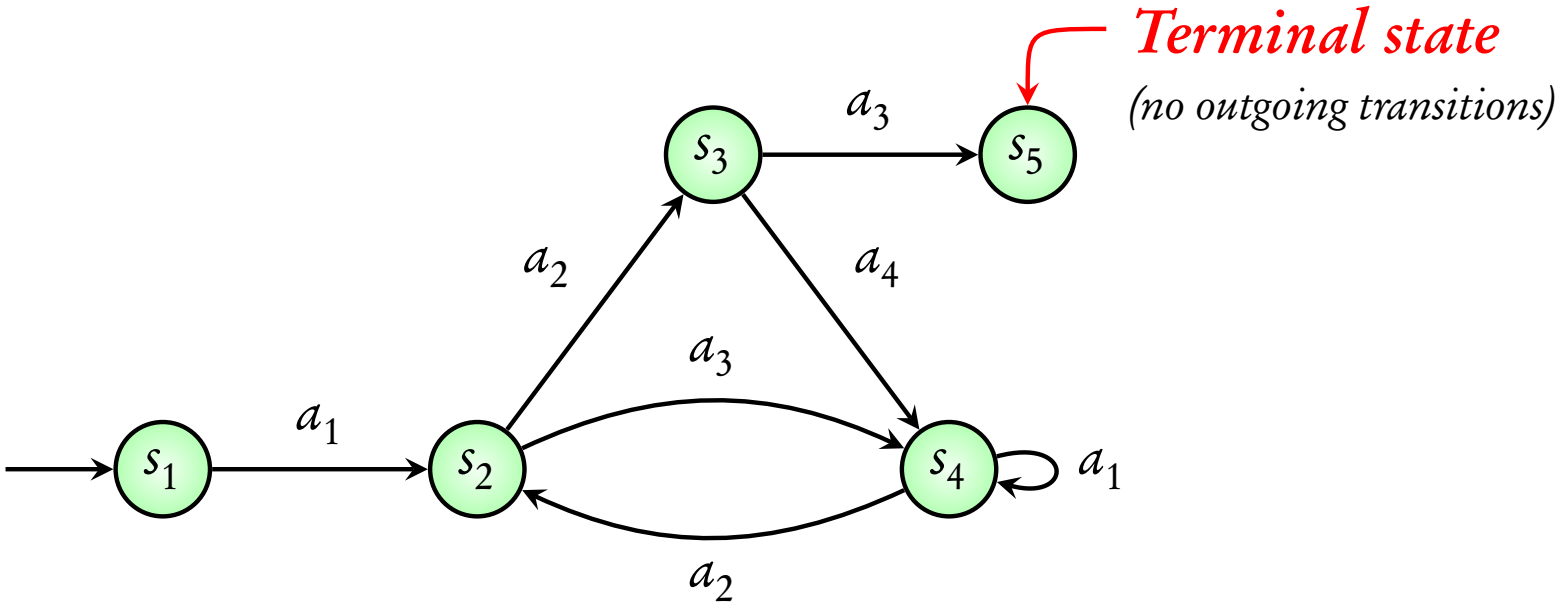
Definition 2.3. Direct Predecessors and Successors

Let $TS = (S, Act, \rightarrow, I, AP, L)$ be a transition system. For $s \in S$ and $\alpha \in Act$, the set of direct *α-successors* of s is defined as:

$$Post(s, \alpha) = \left\{ s' \in S \mid s \xrightarrow{\alpha} s' \right\}, \quad Post(s) = \bigcup_{\alpha \in Act} Post(s, \alpha).$$

The set of *α-predecessors* of s is defined by:

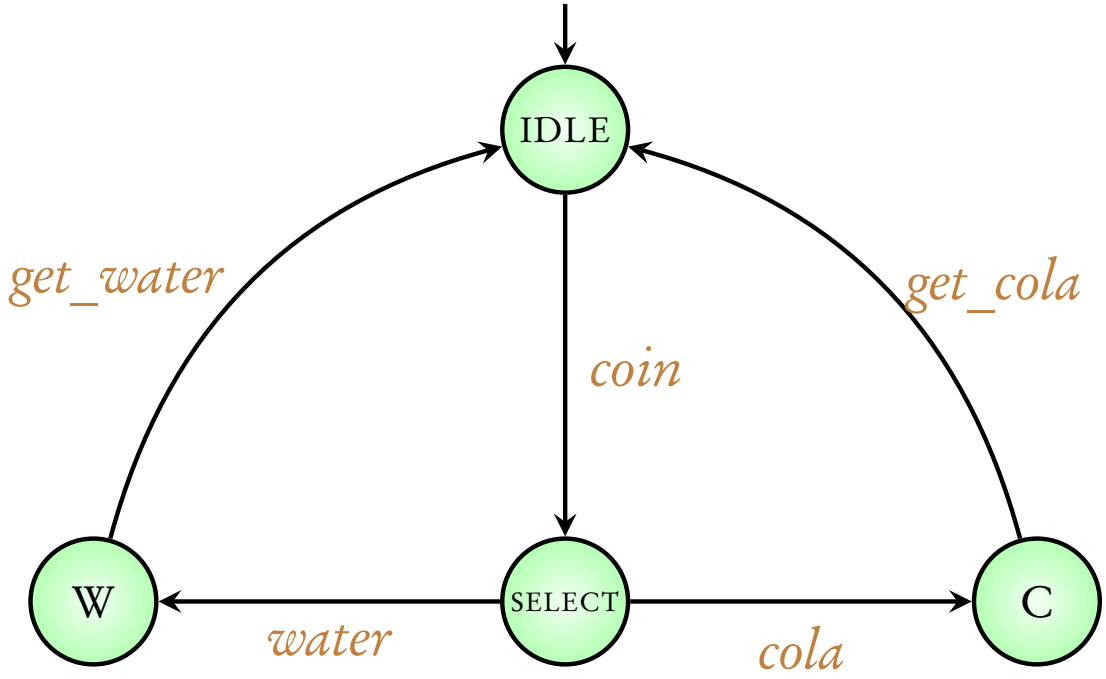
$$Pre(s, \alpha) = \left\{ s' \in S \mid s' \xrightarrow{\alpha} s \right\}, \quad Pre(s) = \bigcup_{\alpha \in Act} Pre(s, \alpha).$$



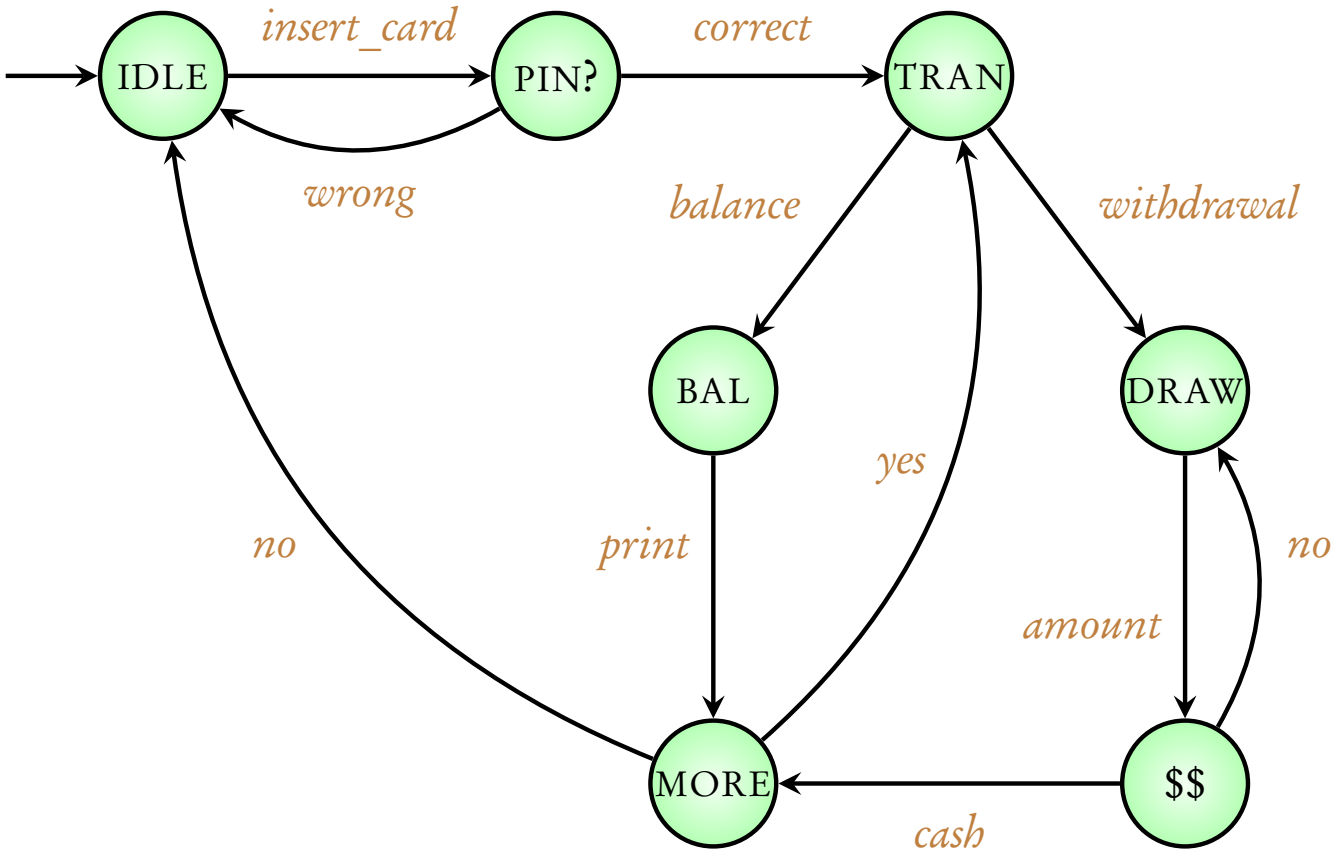
Execution: $s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_3} s_4 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_4 \xrightarrow{a_2} s_2 \xrightarrow{a_3} s_4 \ldots$

$s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_4} s_4 \xrightarrow{a_1} s_4 \xrightarrow{a_1} s_4 \xrightarrow{a_1} s_4 \ldots$

$s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_4} s_4 \xrightarrow{a_2} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} s_5$



Execution: IDLE $\xrightarrow{coin}$ SELECT $\xrightarrow{water}$ W $\xrightarrow{get\_water}$ IDLE $\xrightarrow{coin}$ SELECT $\xrightarrow{cola}$ $\cdots$



Execution: IDLE $\xrightarrow{insert\_card}$ PIN $\xrightarrow{wrong}$ IDLE $\xrightarrow{insert\_card}$ PIN $\cdots$

Executions formally

Definition 2.6. Execution Fragment

Let $TS = (S, Act, \rightarrow, I, AP, L)$ be a transition system. A *finite* execution fragment ϱ of TS is an alternating sequence of states and actions ending with a state

$$\varrho \;=\; s_0\, \alpha_1\, s_1\, \alpha_2 \ldots \alpha_n\, s_n \text{ such that } s_i \xrightarrow{\alpha_{i+1}} s_{i+1} \text{ for all } 0 \leqslant i < n,$$

where $n \geqslant 0$. We refer to n as the length of the execution fragment ϱ. An *infinite* execution fragment ρ of TS is an infinite, alternating sequence of states and actions:

$$\rho \;=\; s_0\, \alpha_1\, s_1\, \alpha_2\, s_2\, \alpha_3 \ldots \text{ such that } s_i \xrightarrow{\alpha_{i+1}} s_{i+1} \text{ for all } 0 \leqslant i.$$

Definition 2.7. Maximal and Initial Execution Fragment

A *maximal* execution fragment is either a finite execution fragment that ends in a terminal state, or an infinite execution fragment. An execution fragment is called *initial* if it starts in an initial state, i.e., if $s_0 \in I$. ■

Definition 2.9. Execution

An *execution* of transition system TS is an initial, maximal execution fragment.

Module 2:

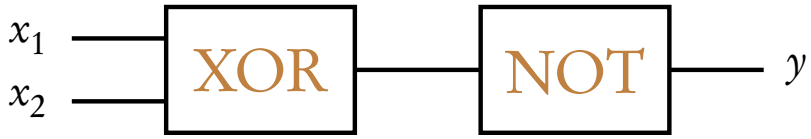
Modeling hardware circuits

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

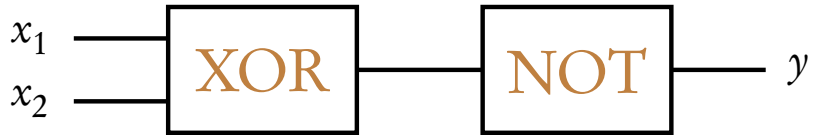
x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

x	y
0	1
1	0

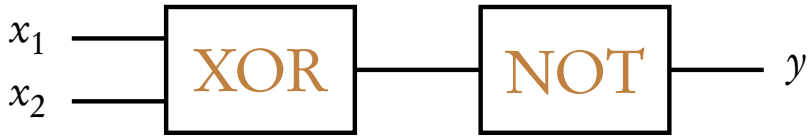
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0


x_1
x_2
XOR
NOT
y

$$y = \text{NOT}\,(\,\text{XOR}\,(x_1, x_2)\,)$$

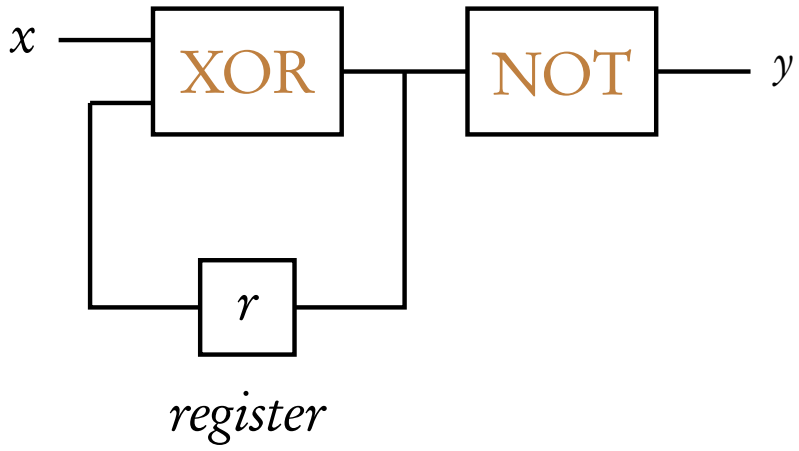


$$y = \text{NOT}\,(\,\text{XOR}\,(x_1, x_2)\,)$$



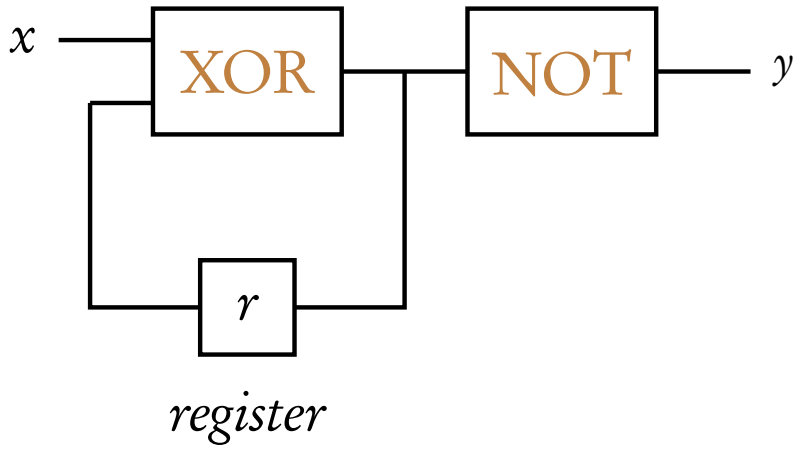
x_1	x_2	y
0	0	1
0	1	0
1	0	0
1	1	1

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



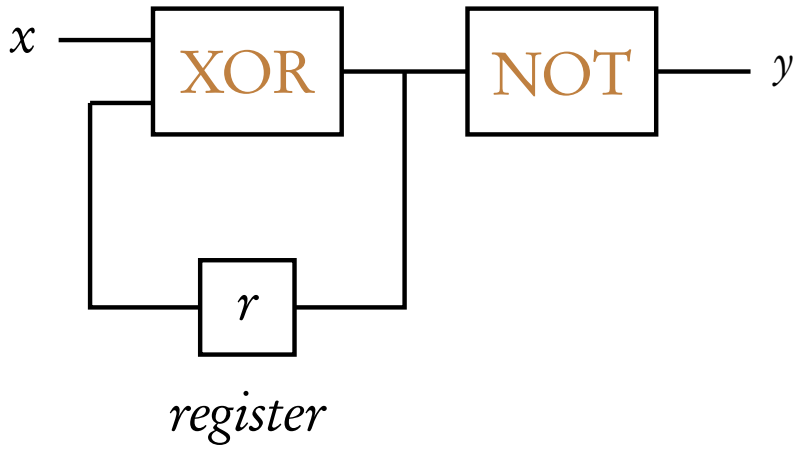
$$y = \text{NOT}\ (\ \text{XOR}\ (x, r)\ )$$

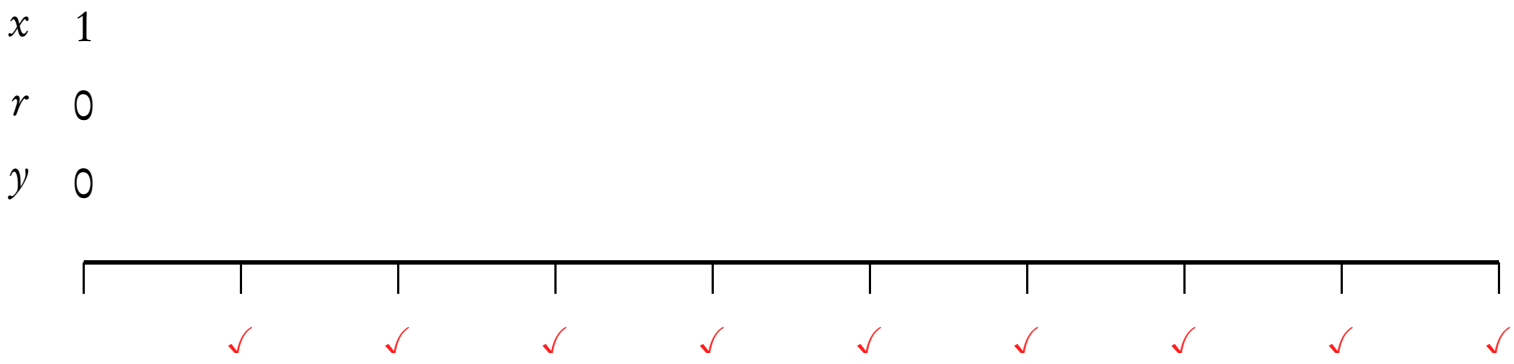
$$r_{next} = \text{XOR}\ (x, r)$$



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

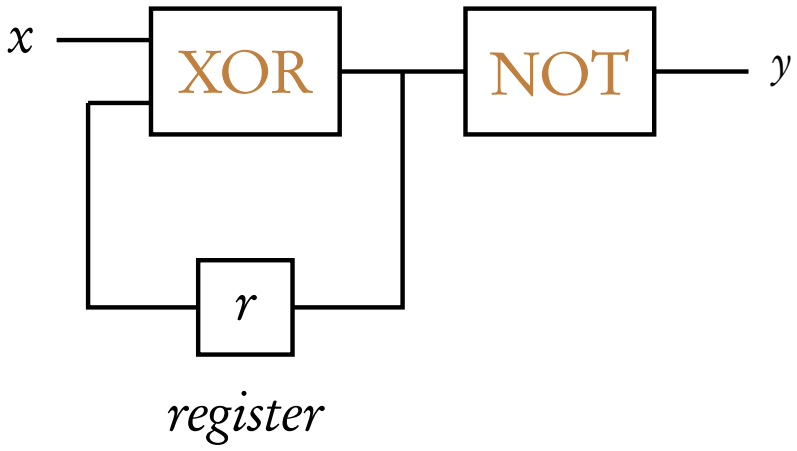
$$r_{next} = \text{XOR}\ (x, r)$$

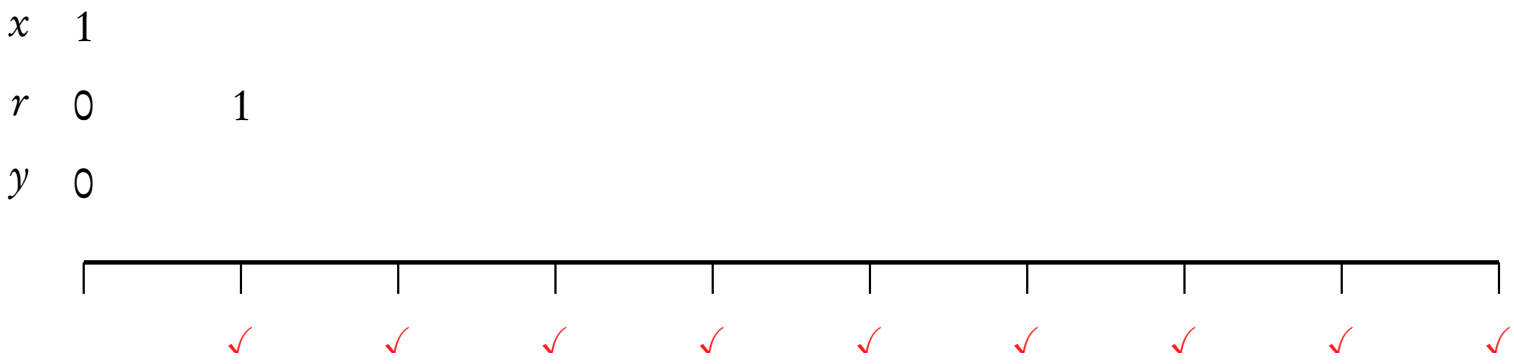




$$y = \text{NOT}\ (\ \text{XOR}\ (x, r)\ )$$

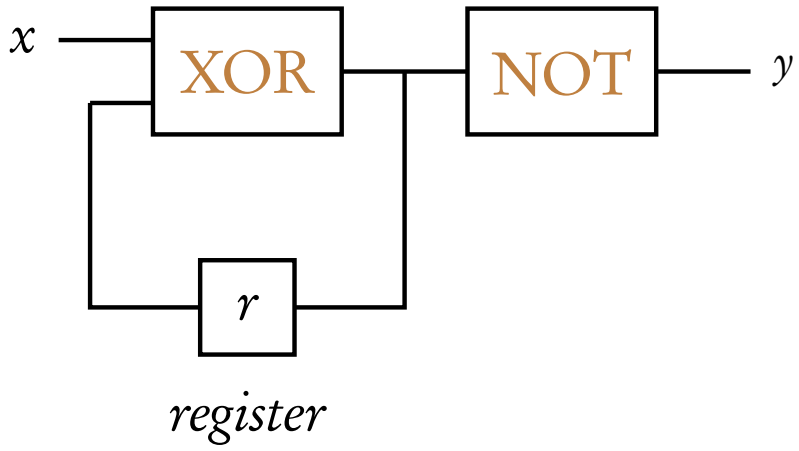
$$r_{next} = \text{XOR}\ (x, r)$$





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

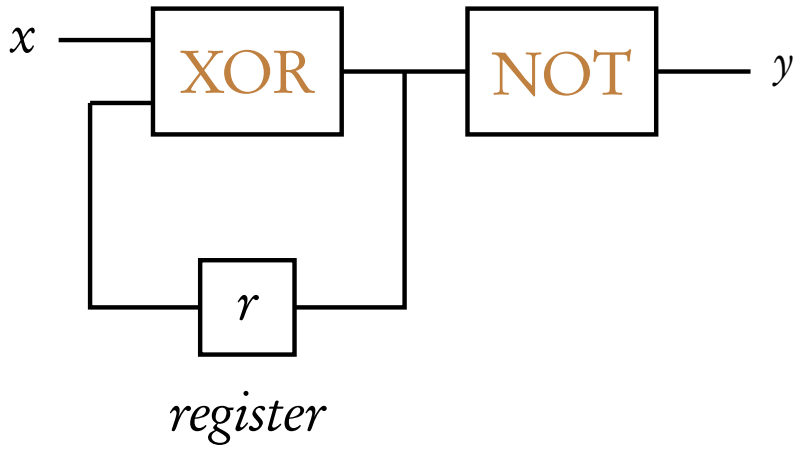
$$r_{next} = \text{XOR}\ (x, r)$$

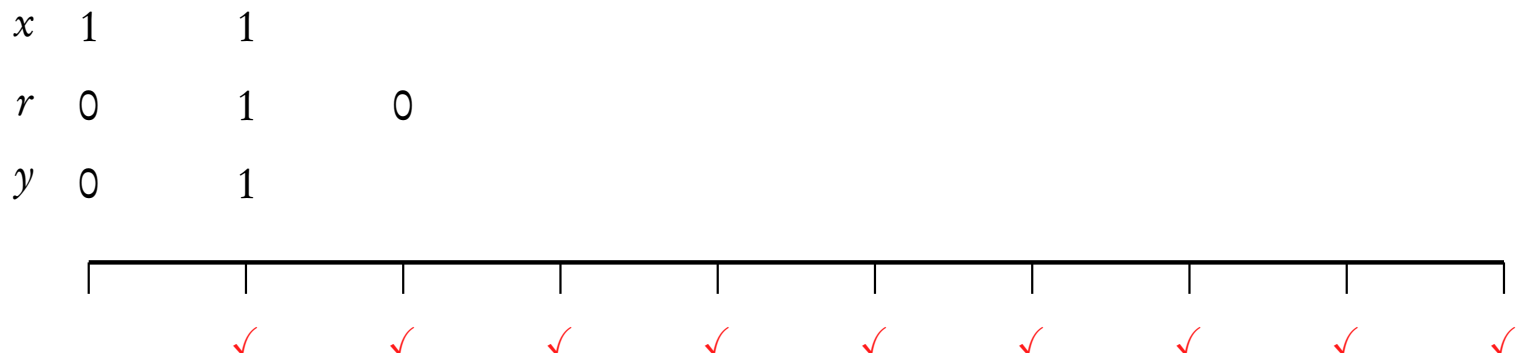


x	1	1
r	0	1
y	0	1

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

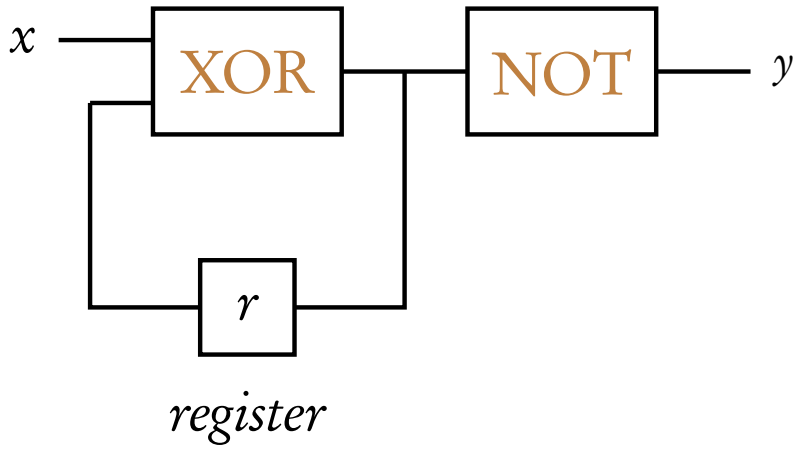
$$r_{next} = \text{XOR}\ (x, r)$$



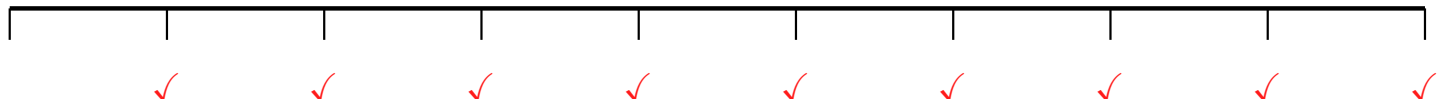


$$y = \text{NOT}\ (\ \text{XOR}\ (x, r)\ )$$

$$r_{next} = \text{XOR}\ (x, r)$$

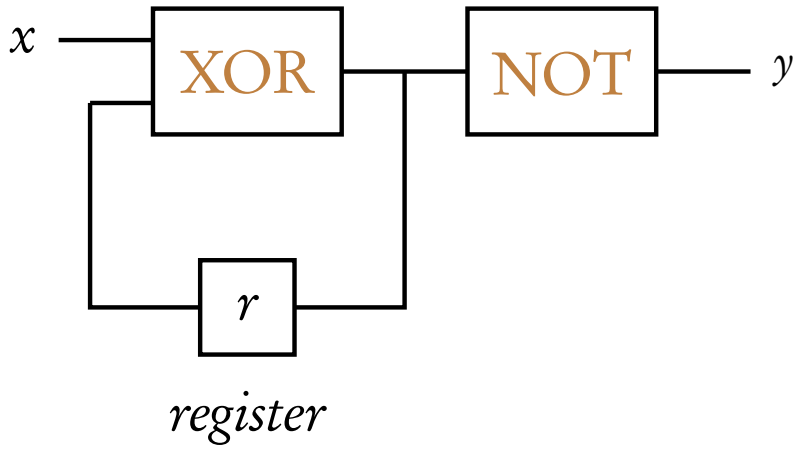


x	1	1	0
r	0	1	0
y	0	1	1

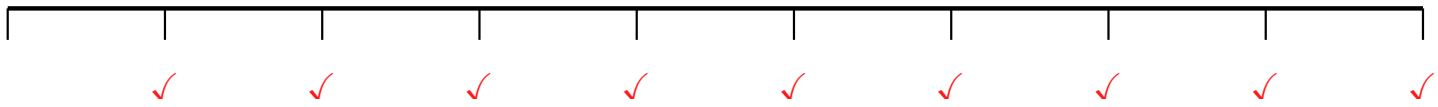


$$y = \text{NOT}\ (\ \text{XOR}\ (x, r)\ )$$

$$r_{next} = \text{XOR}\ (x, r)$$

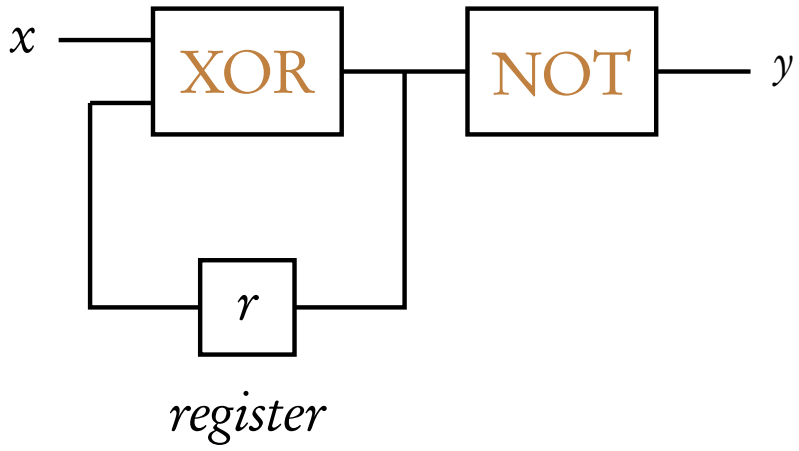


x	1	1	0	
r	0	1	0	0
y	0	1	1	

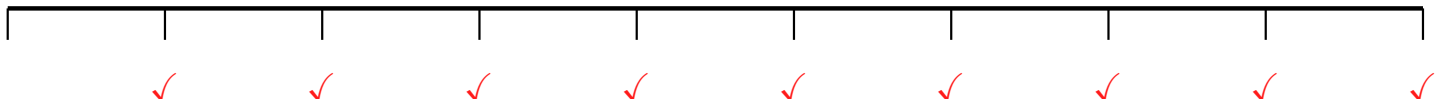


$$y = \text{NOT}\ (\ \text{XOR}\ (x, r)\ )$$

$$r_{next} = \text{XOR}\ (x, r)$$

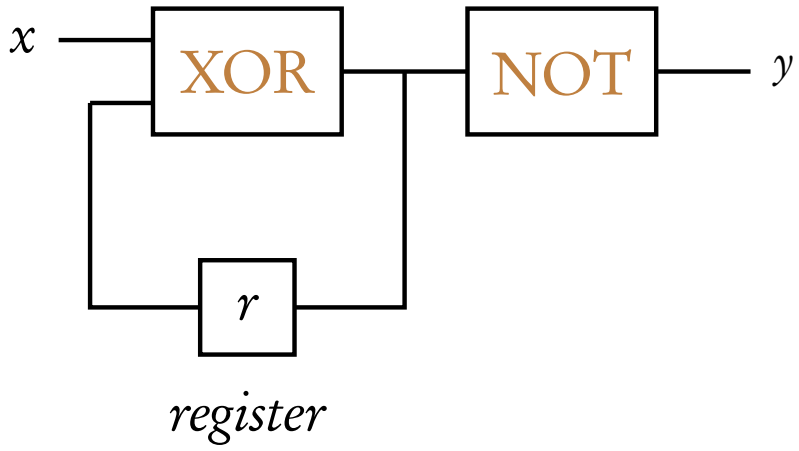


x	1	1	0	1
r	0	1	0	0
y	0	1	1	0

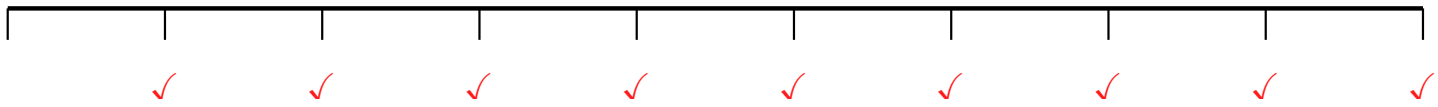


$$y = \text{NOT}\ (\ \text{XOR}\ (x, r)\ )$$

$$r_{next} = \text{XOR}\ (x, r)$$

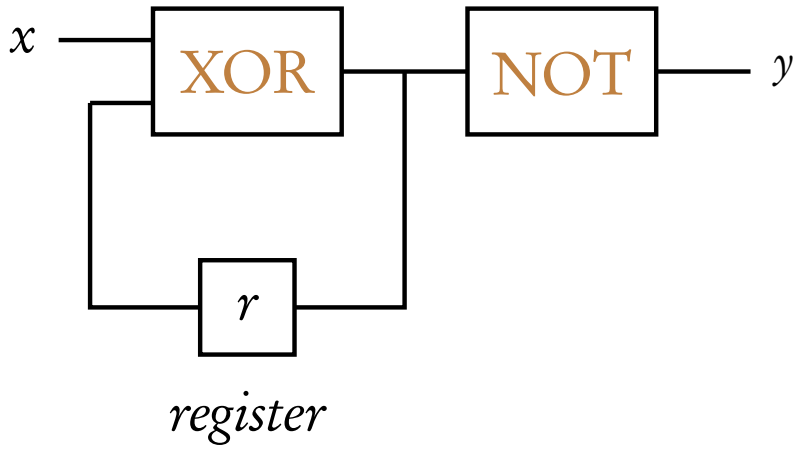


x	1	1	0	1	
r	0	1	0	0	1
y	0	1	1	0	

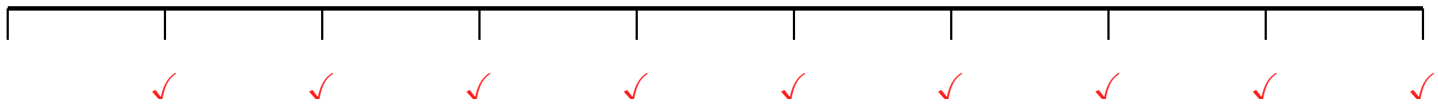


$$y = \text{NOT}\ (\ \text{XOR}\ (x, r)\ )$$

$$r_{next} = \text{XOR}\ (x, r)$$

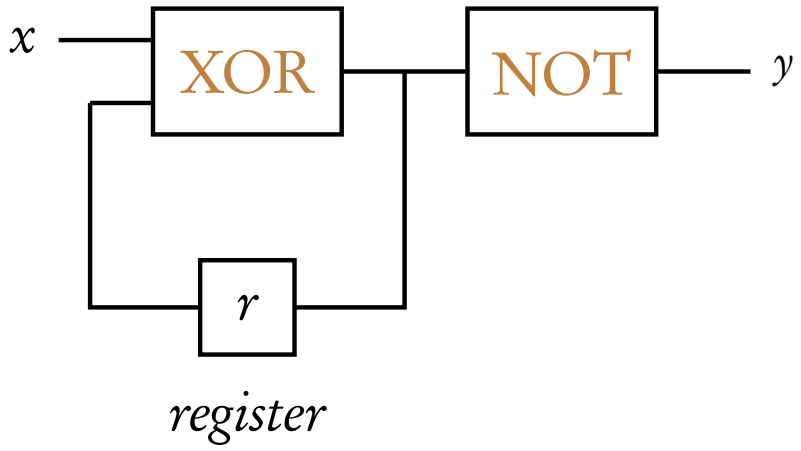


x	1	1	0	1	1
r	0	1	0	0	1
y	0	1	1	0	1



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

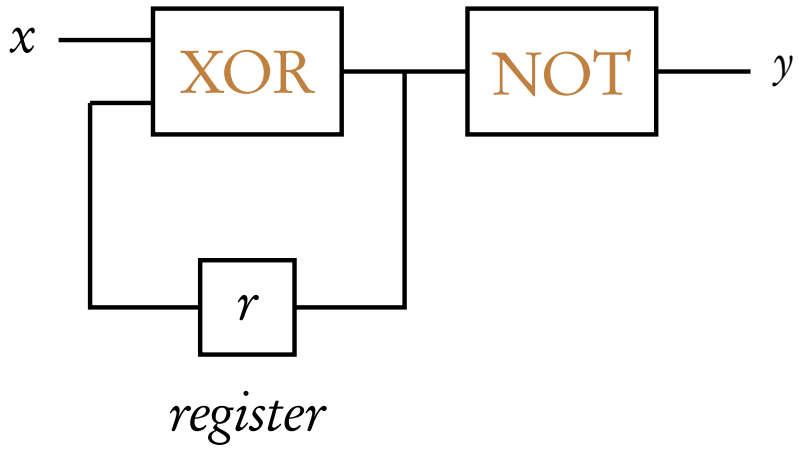
$$r_{next} = \text{XOR}\ (x, r)$$



x	1	1	0	1	1	1	0	0	1	0
r	0	1	0	0	1	0	1	1	1	0 $\cdots$
y	0	1	1	0	1	0	0	0	1	1

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

$$r_{next} = \text{XOR}\ (x, r)$$



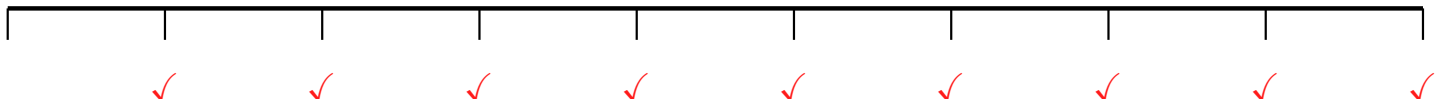
$x = 0, r = 0, y = 1$

$x = 1, r = 0, y = 0$

$x = 0, r = 1, y = 0$

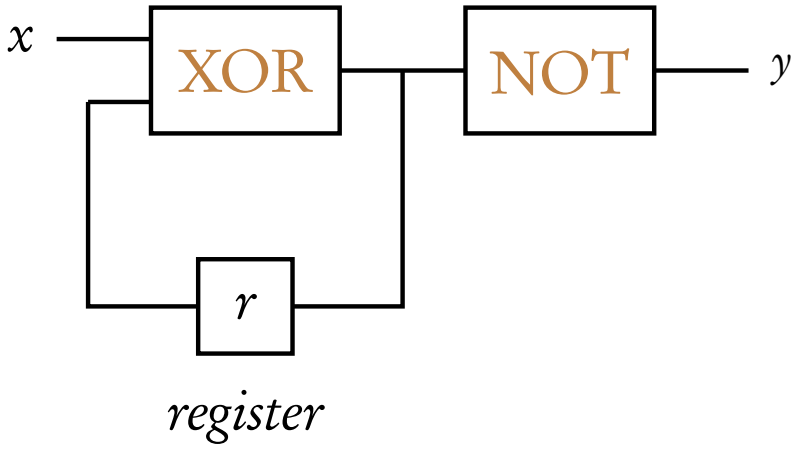
$x = 1, r = 1, y = 1$

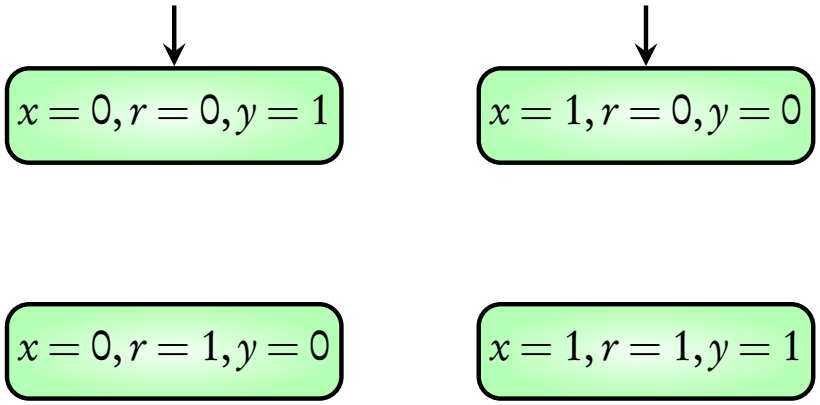
x	1		1		0		1		1		1		0		0		1		0	
r	0		1		0		0		1		0		1		1		1		0	$\cdots$
y	0		1		1		0		1		0		0		0		1		1	



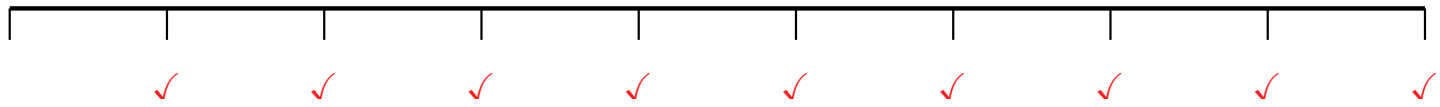
$$y = \text{NOT}\ (\ \text{XOR}\ (x, r)\ )$$

$$r_{next} = \text{XOR}\ (x, r)$$



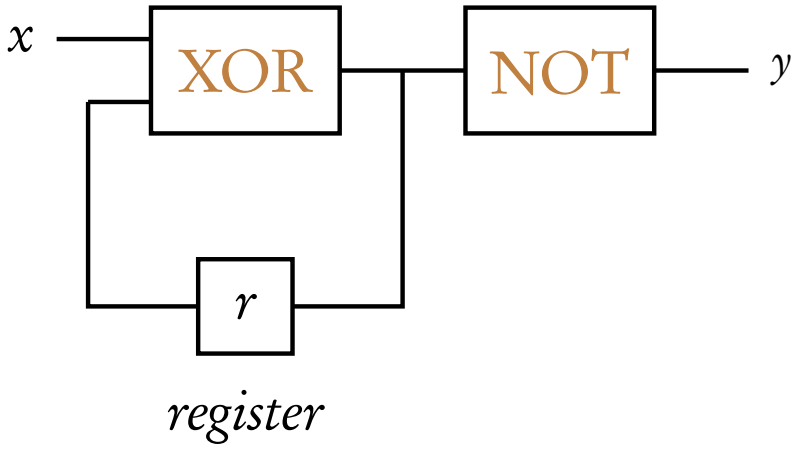


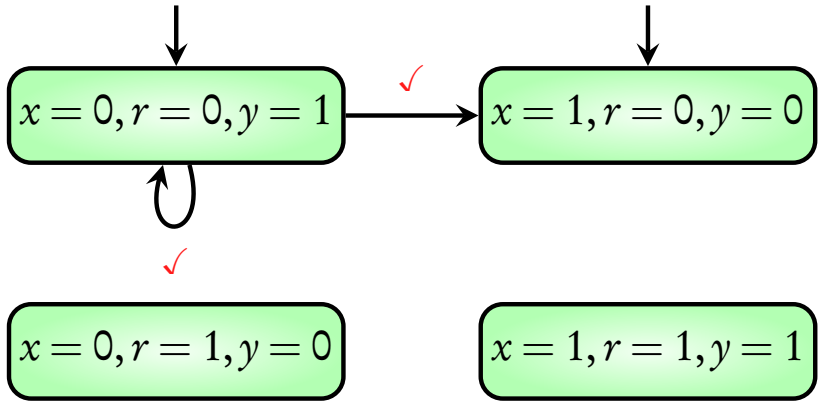
x	1		1		0		1		1		1		0		0		1		0	
r	0		1		0		0		1		0		1		1		1		0	• • •
y	0		1		1		0		1		0		0		0		1		1	



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

$$r_{next} = \text{XOR}\ (x, r)$$

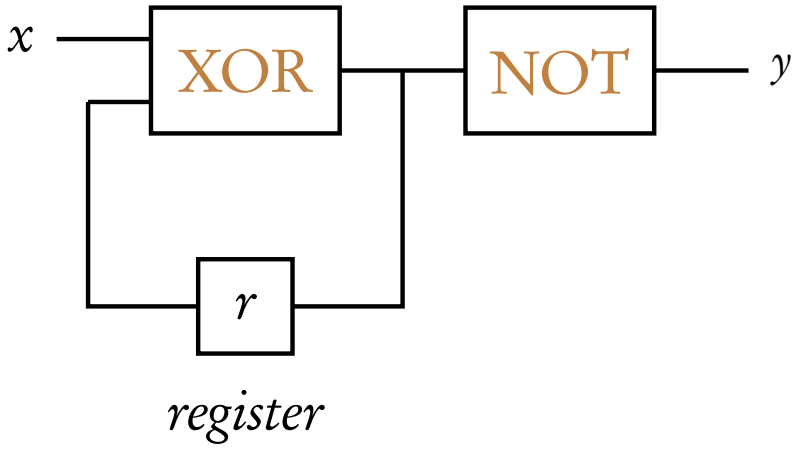


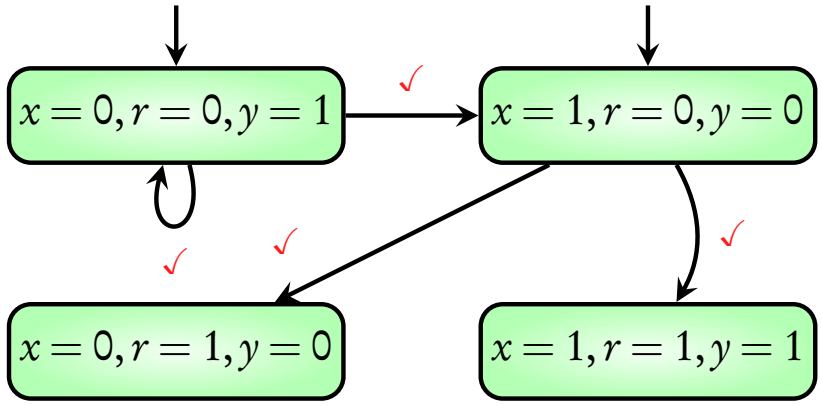


x	1	1	0	1	1	1	0	0	1	0	
r	0	1	0	0	1	0	1	1	1	0	•••
y	0	1	1	0	1	0	0	0	1	1	

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

$$r_{next} = \text{XOR}\ (x, r)$$

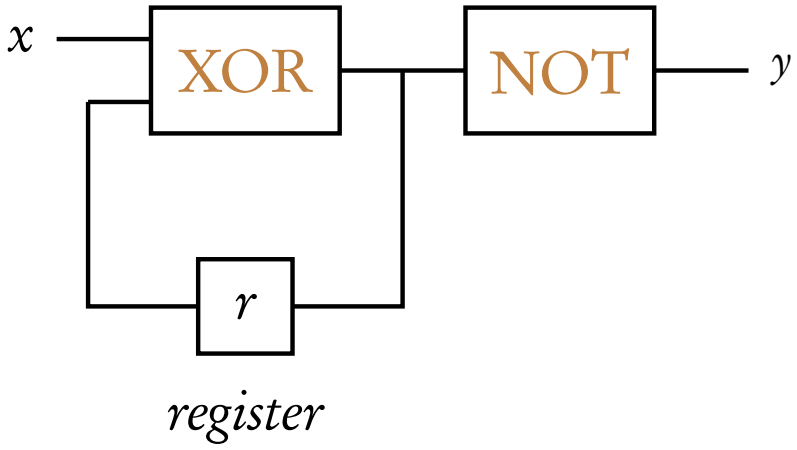


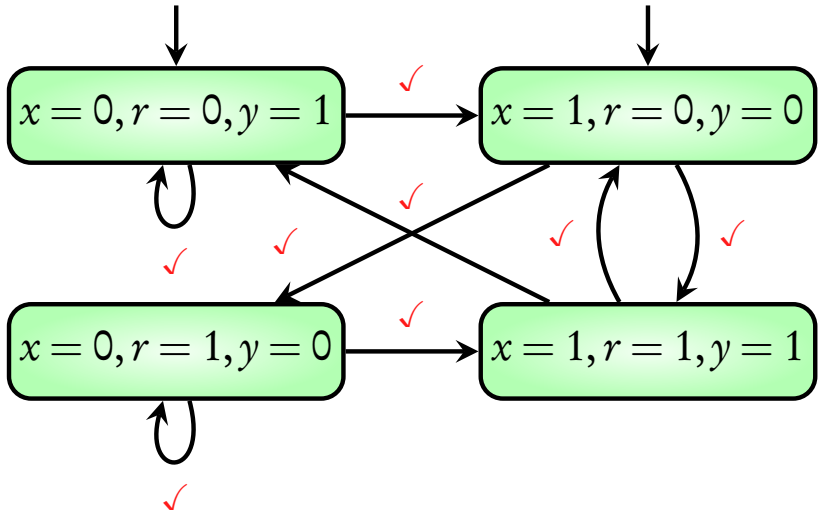


x	1		1		0		1	1	1	0	0	1	0	
r	0		1		0		0	1	0	1	1	1	0	• • •
y	0		1		1		0	1	0	0	0	1	1	

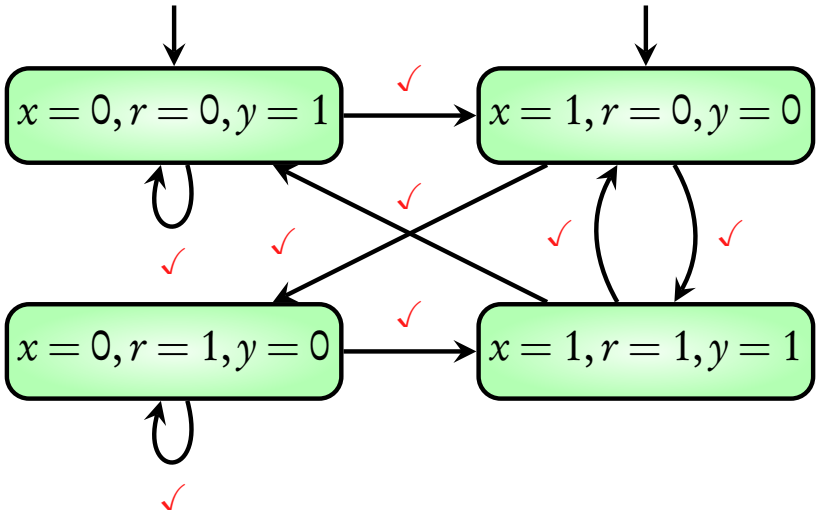
$$y = \text{NOT}\ (\ \text{XOR}\ (x, r)\ )$$

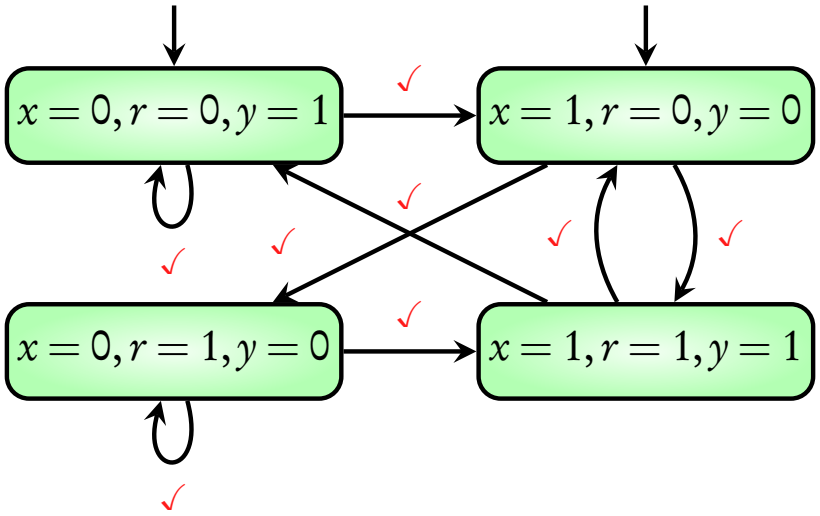
$$r_{next} = \text{XOR}\ (x, r)$$





x	1	1	0	1	1	1	0	0	1	0	
r	0	1	0	0	1	0	1	1	1	0	⋯
y	0	1	1	0	1	0	0	0	1	1	


$x = 0, r = 0, y = 1$
$x = 1, r = 0, y = 0$
$x = 0, r = 1, y = 0$
$x = 1, r = 1, y = 1$

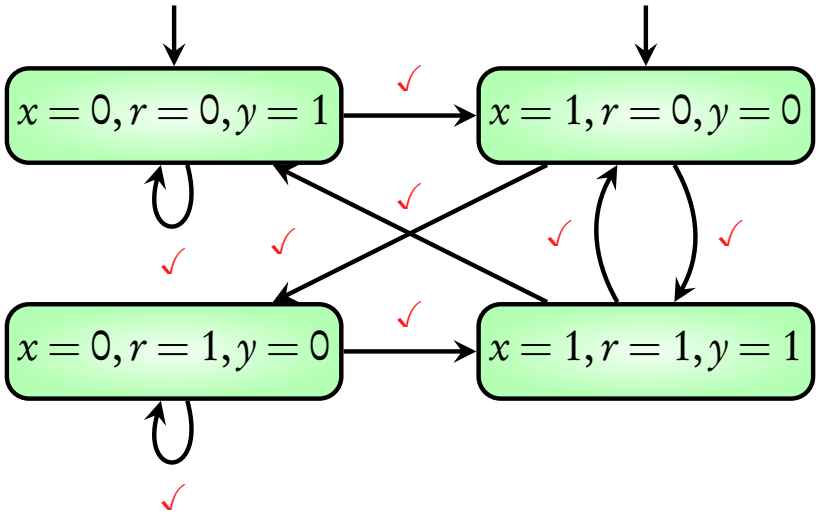


More than one initial state

States with **more than one transition** on an action

Non-deterministic

transition system



More than one initial state

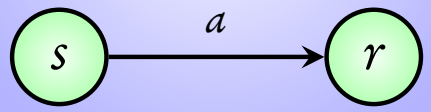
States with **more than one transition** on an action

Transition Systems

Deterministic

Single initial state

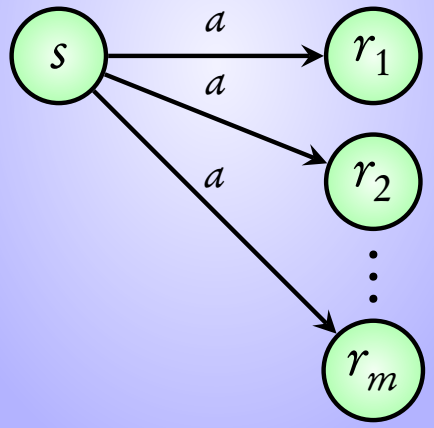
and



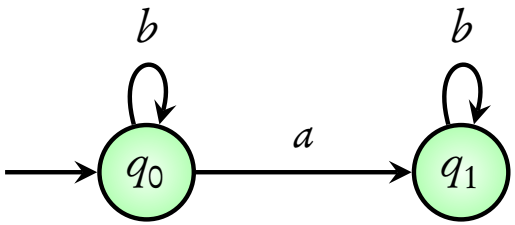
Non-deterministic

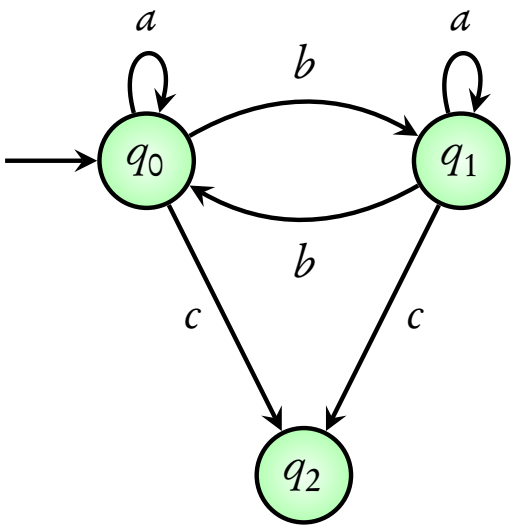
Multiple initial states

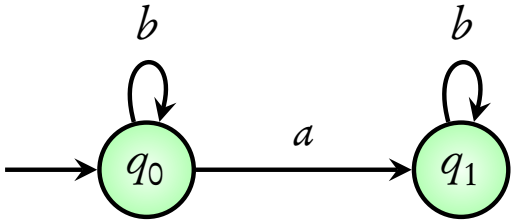
or



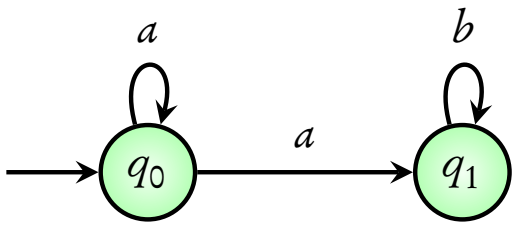
Deterministic

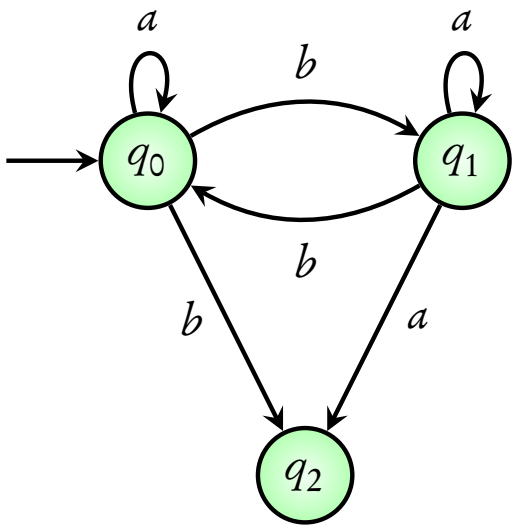


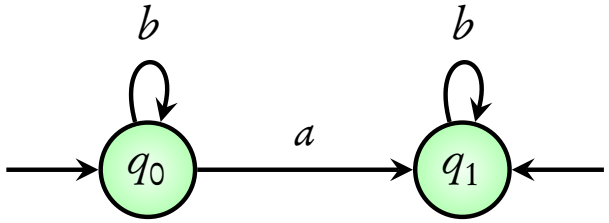




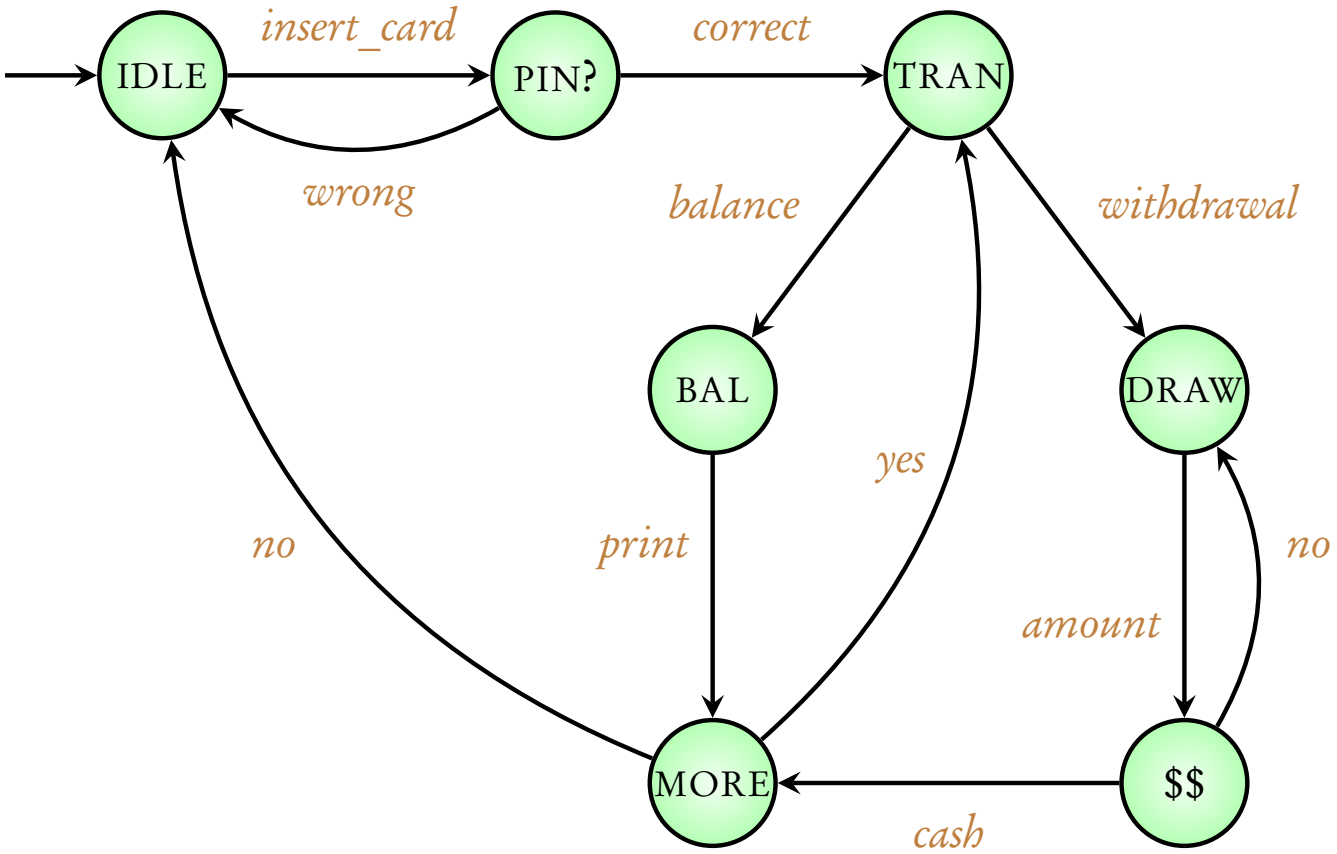
Non-deterministic



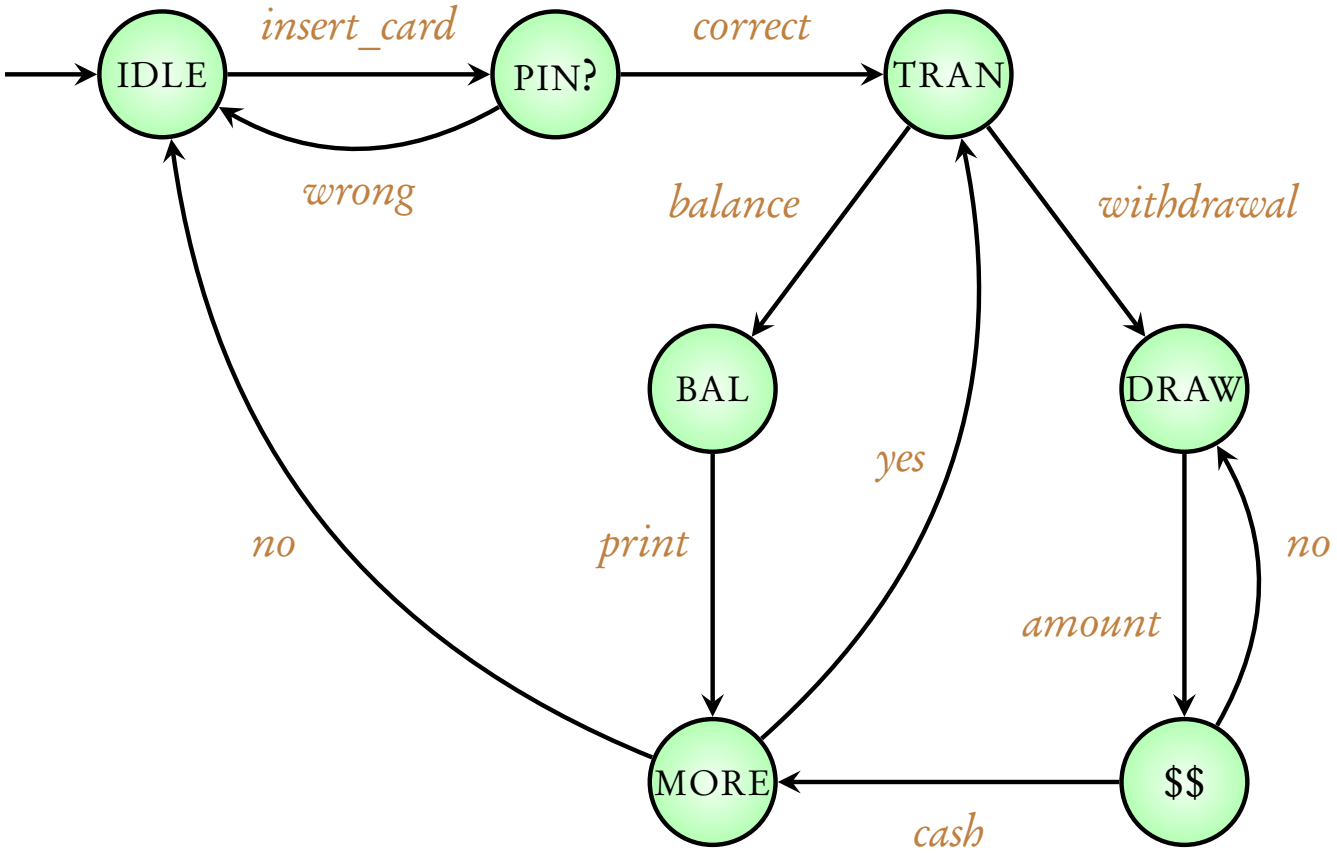




Model of ATM

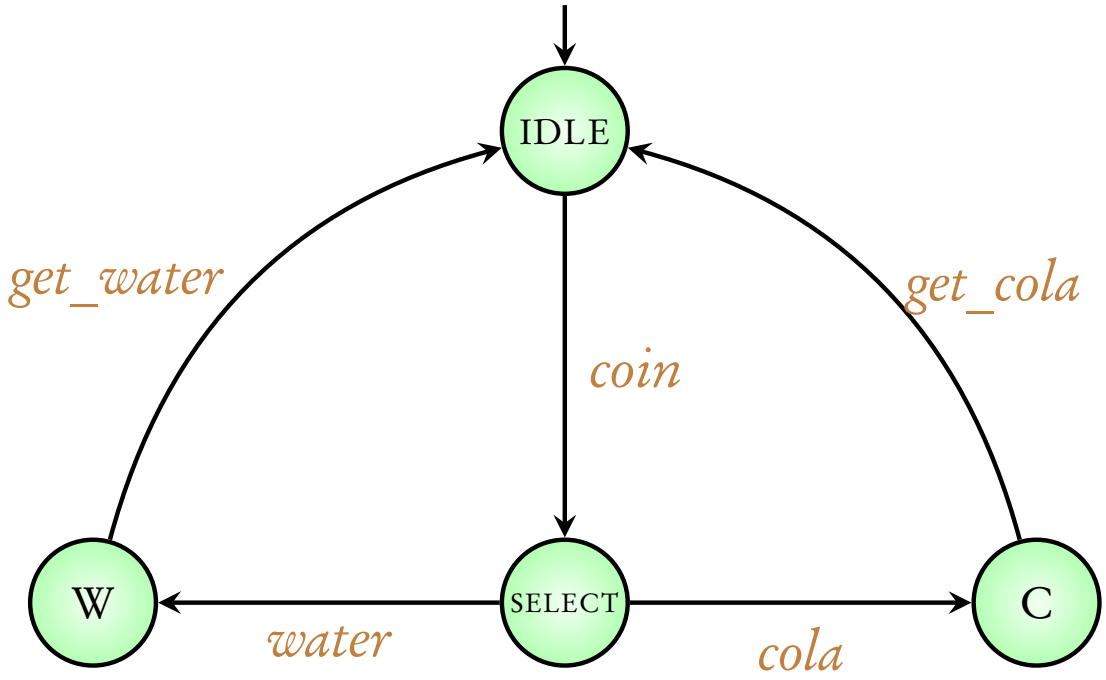

IDLE
insert_card
PIN?
correct
TRAN
wrong
balance
withdrawal
BAL
DRAW
yes
no
print
no
amount
MORE
$$
cash

Model of ATM

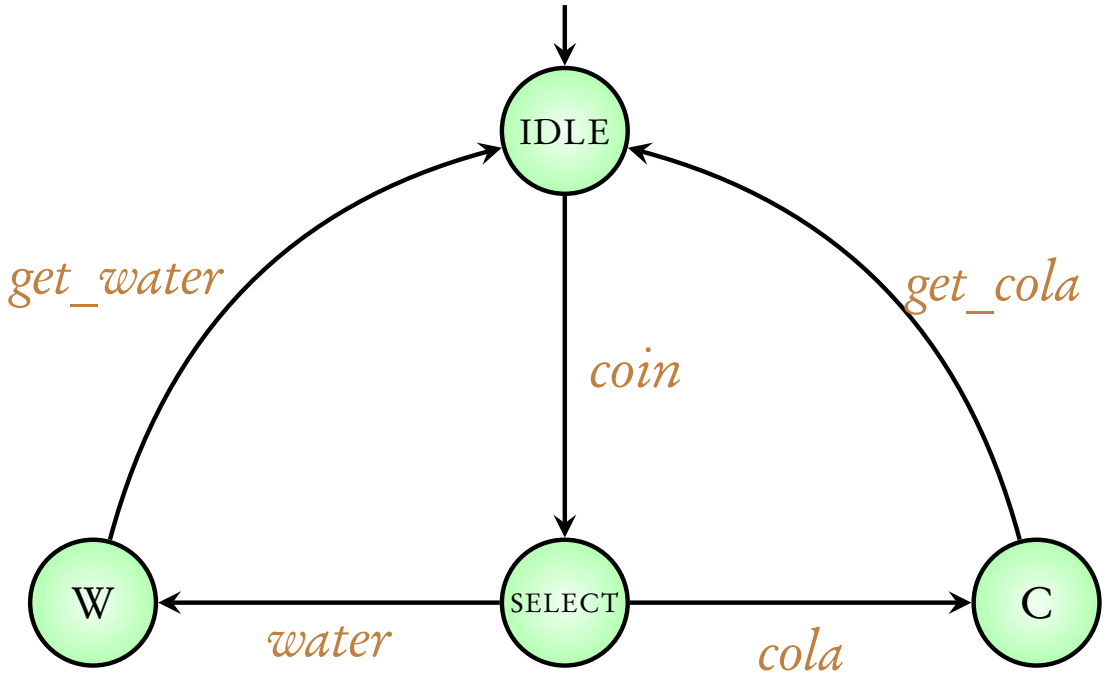


Deterministic transition system

Model of vending machine

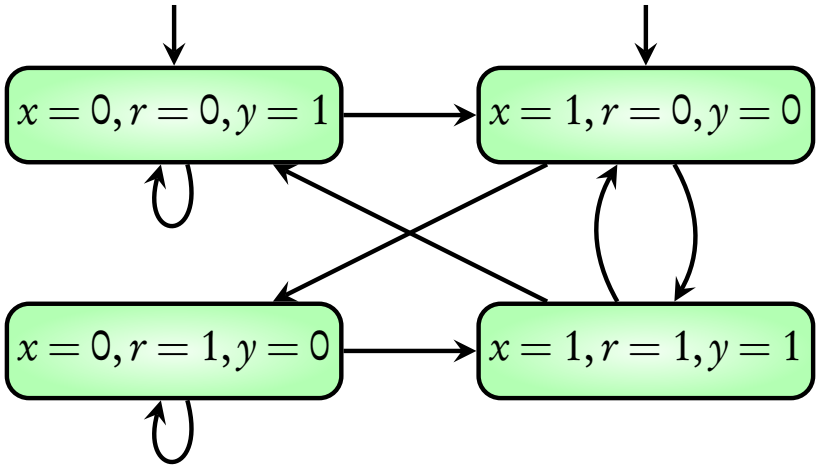


Model of vending machine

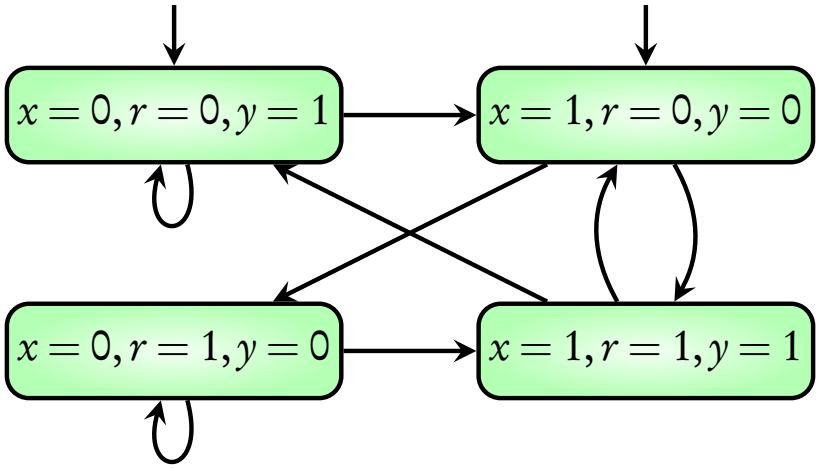


Deterministic transition system

Model of hardware circuit

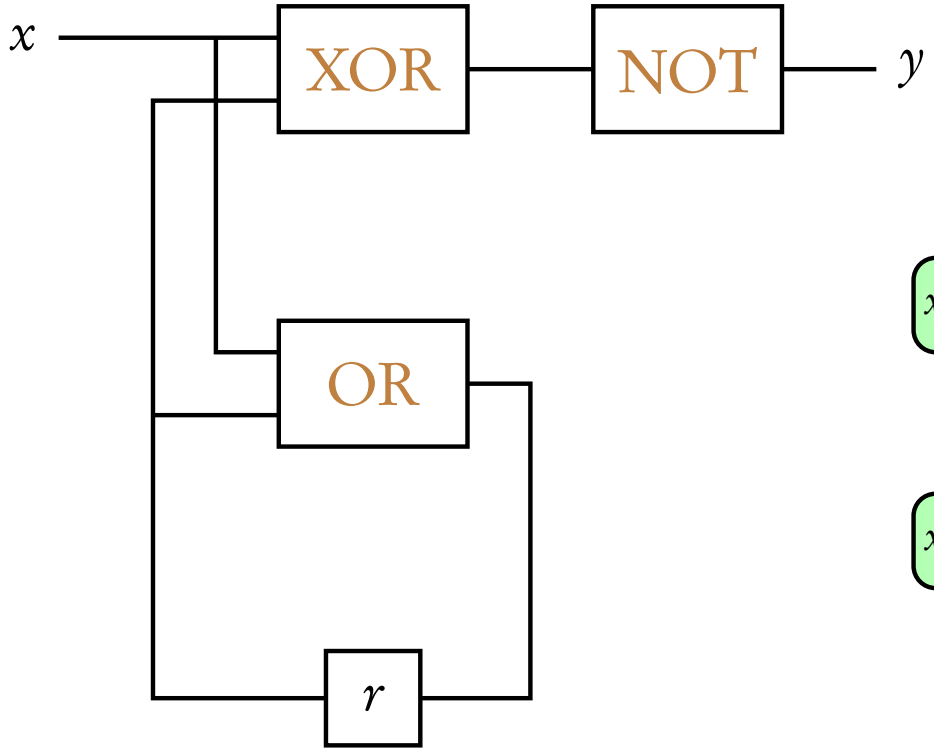


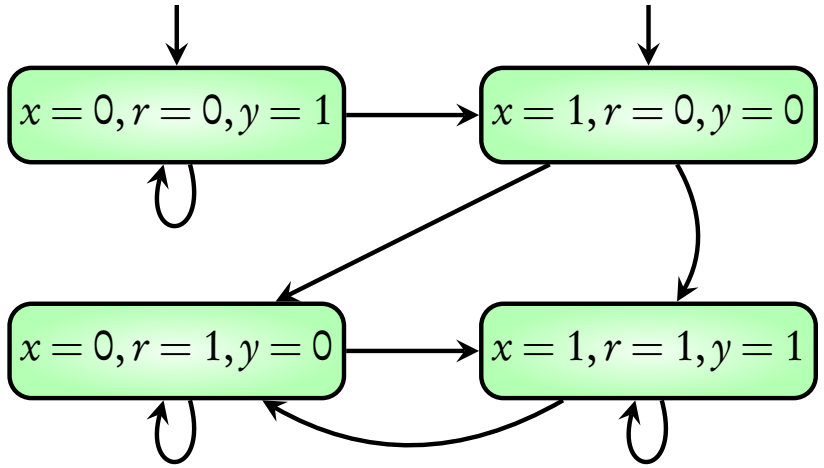
Model of hardware circuit



Non-deterministic transition system: to model **incomplete information**

Another Example





Module 3:

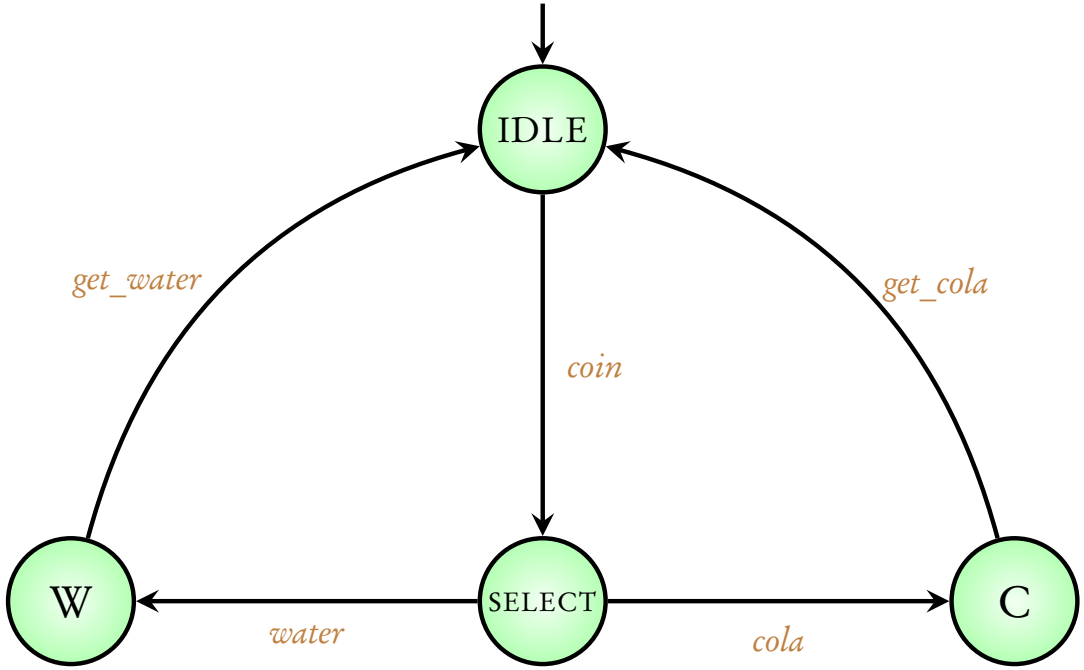
Modeling data-dependent programs

Data-dependent programs:

Variables + Conditional branching + Assignments

$$\textbf{if } x\%2 = 1 \textbf{ then } x := x + 1 \textbf{ else } x := 2{\cdot}x \textbf{ fi}.$$

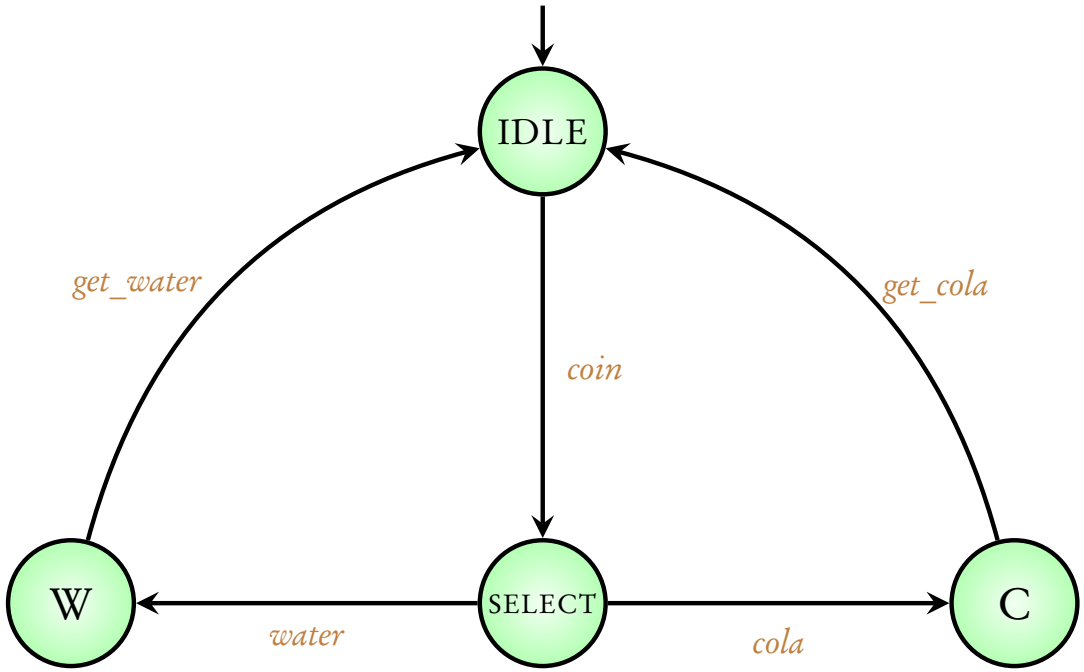
Coming next: vending machine revisited



Count the number of bottles and return coins if the machine is empty...

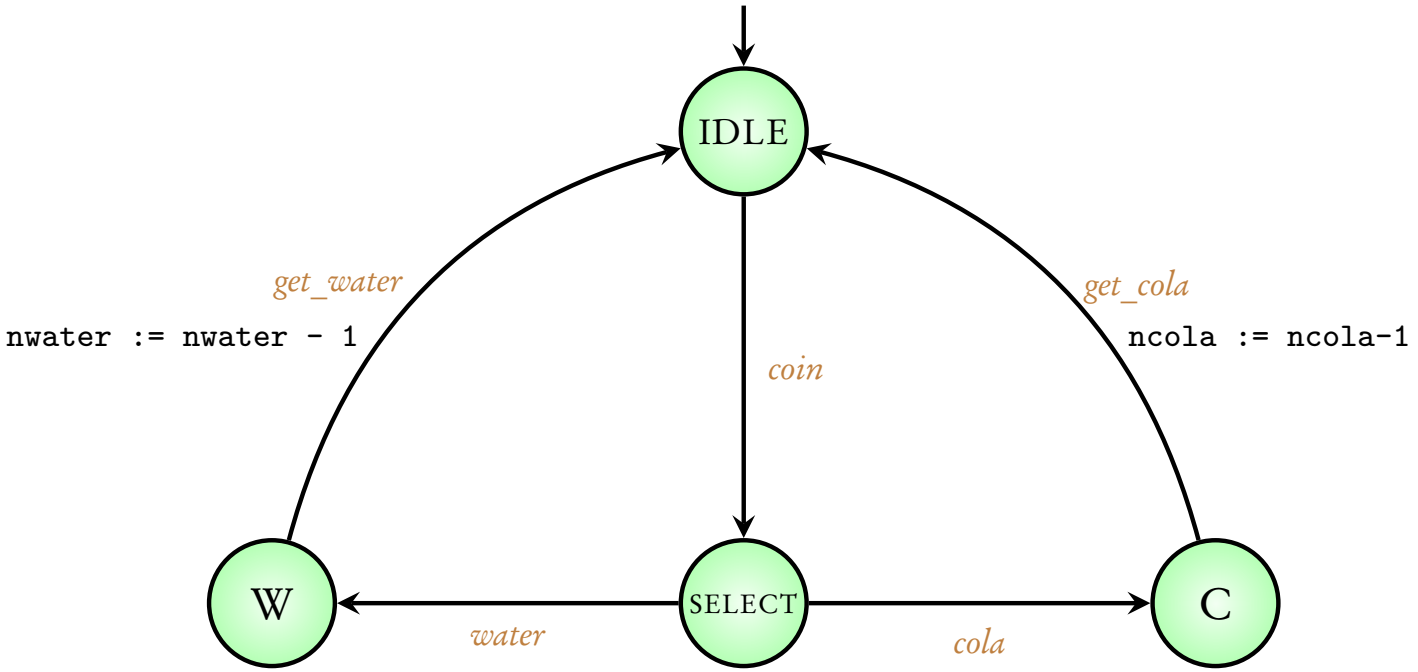
Variables:

`nwater, ncola, max`



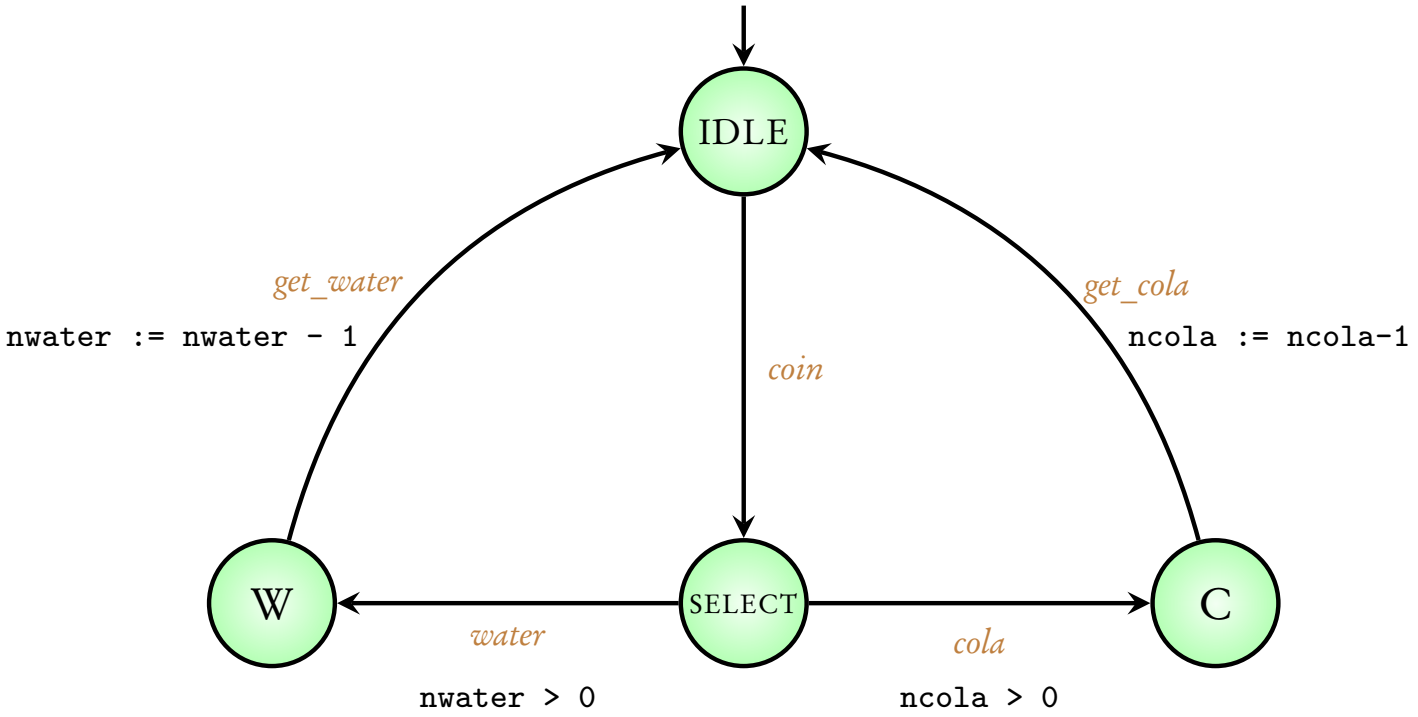
Variables:

`nwater, ncola, max`



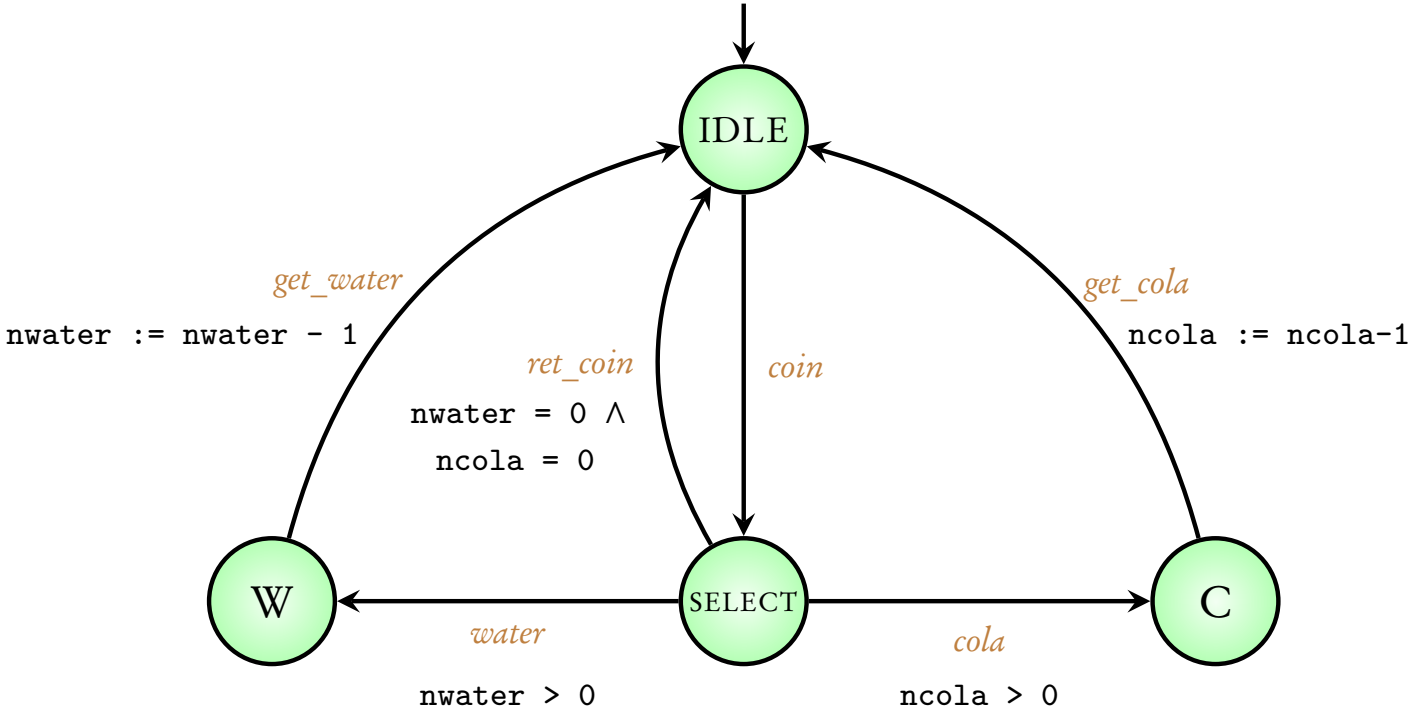
Variables:

`nwater, ncola, max`



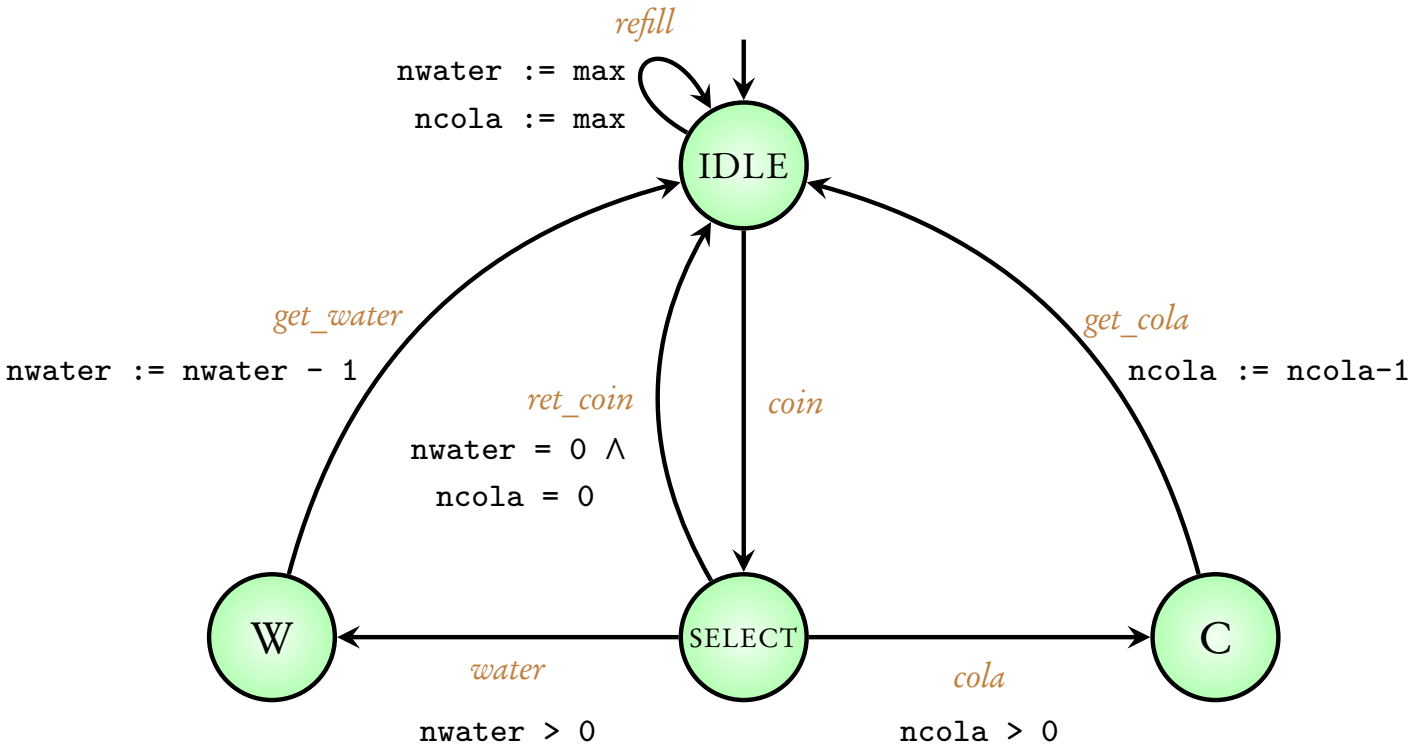
Variables:

`nwater, ncola, max`



Variables:

`nwater, ncola, max`

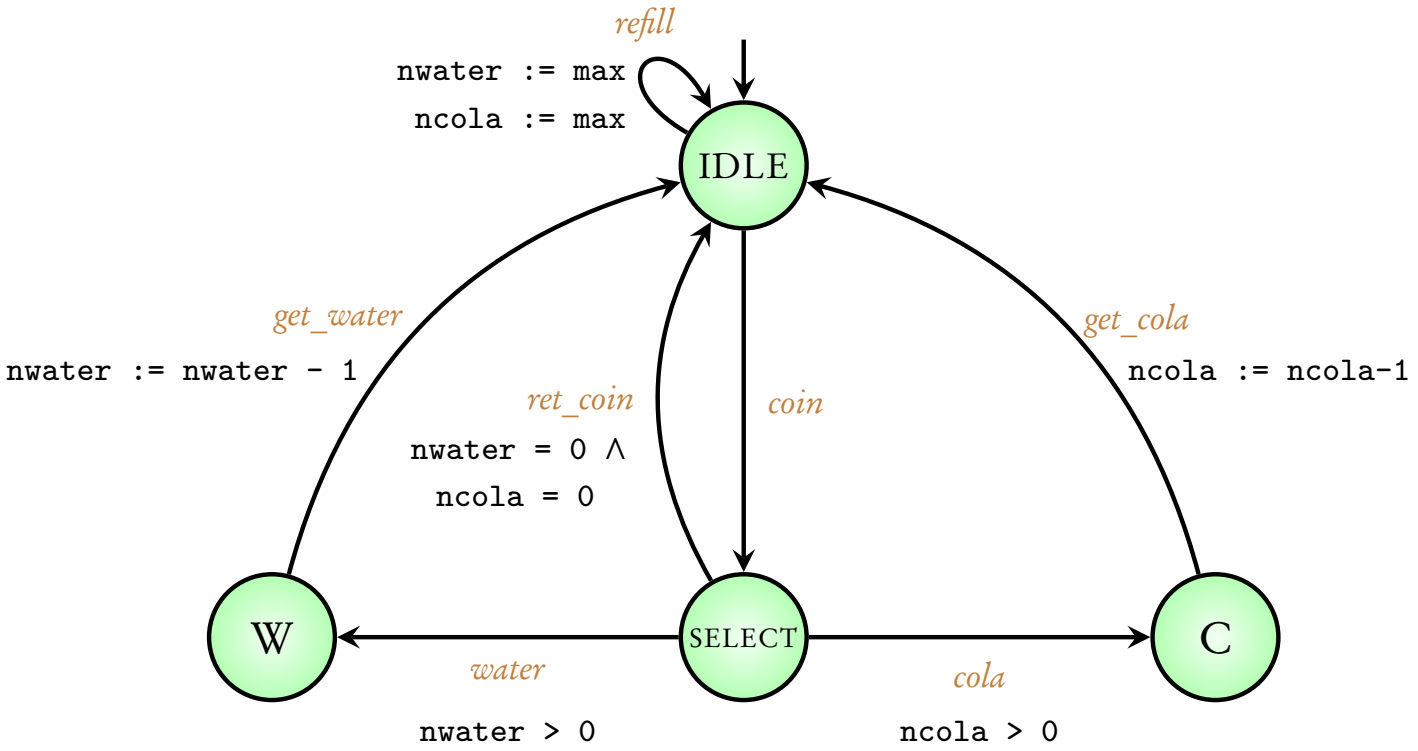


Initial condition:

`nwater = max, ncola = max`

Variables:

`nwater, ncola, max`

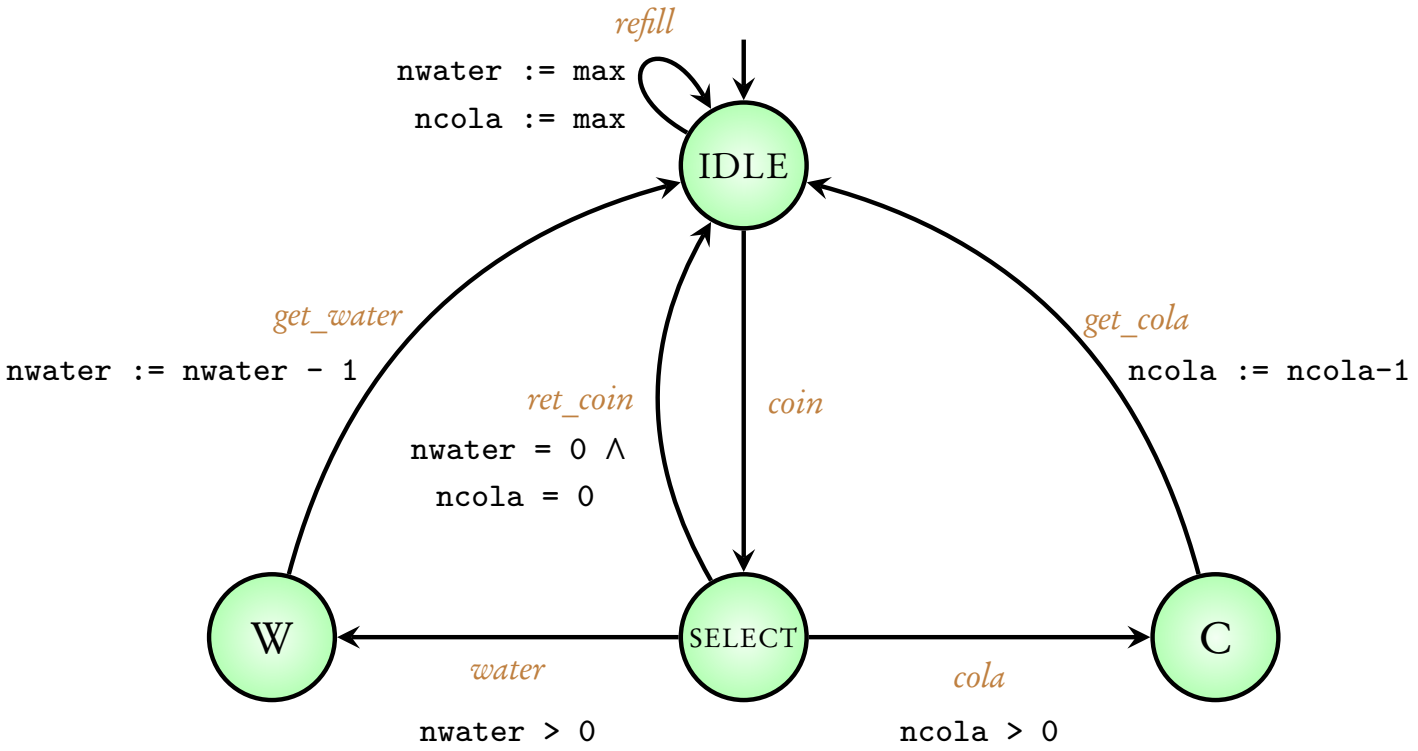


Initial condition:

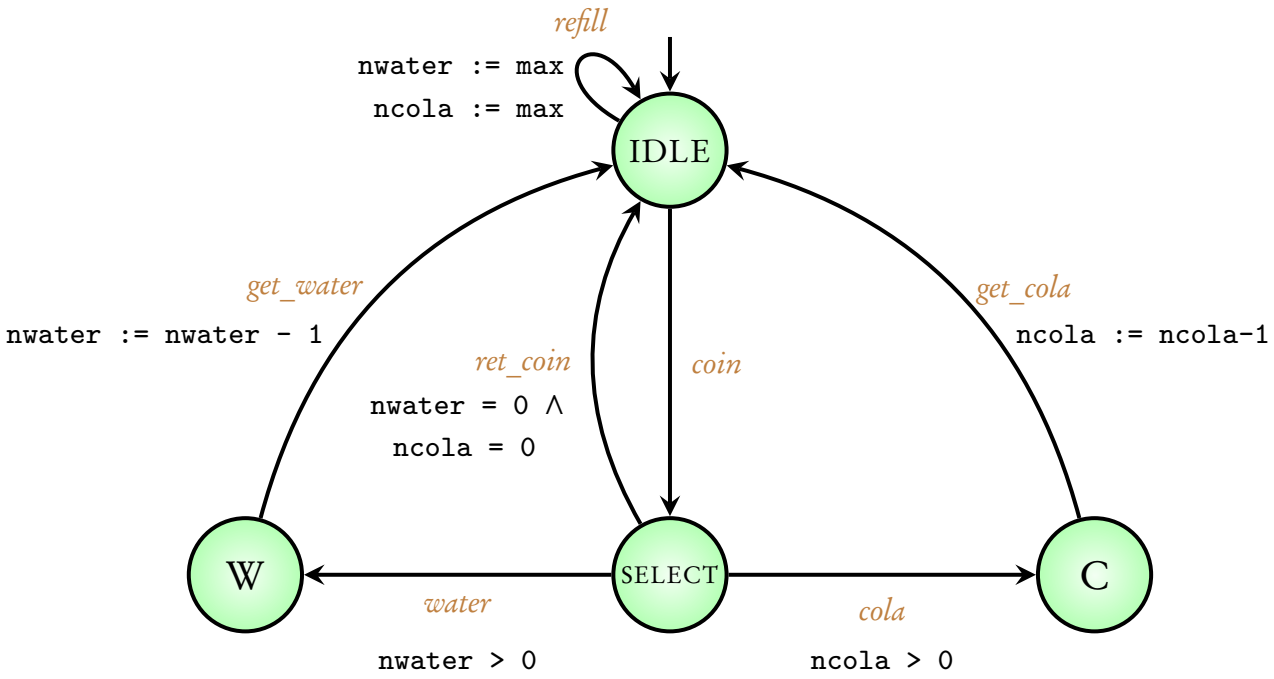
`nwater = max, ncola = max`

Variables:

`nwater, ncola, max`

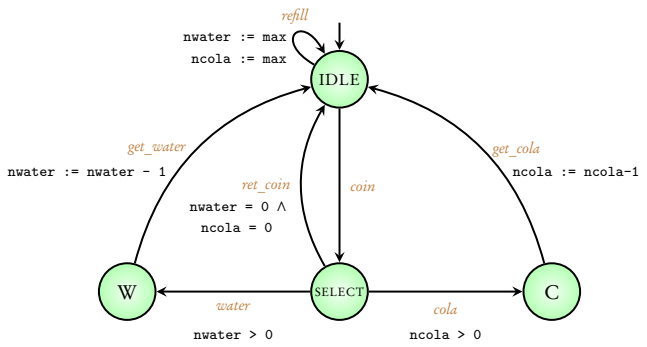


Program graph

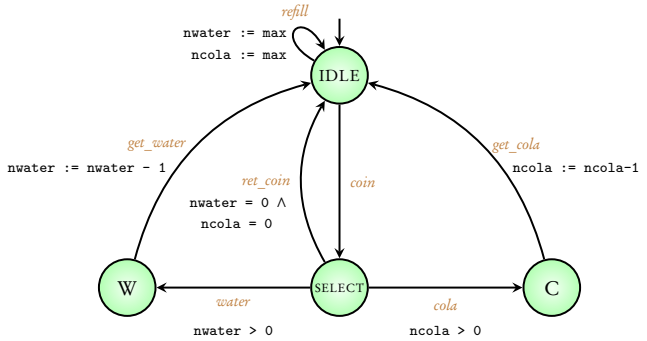


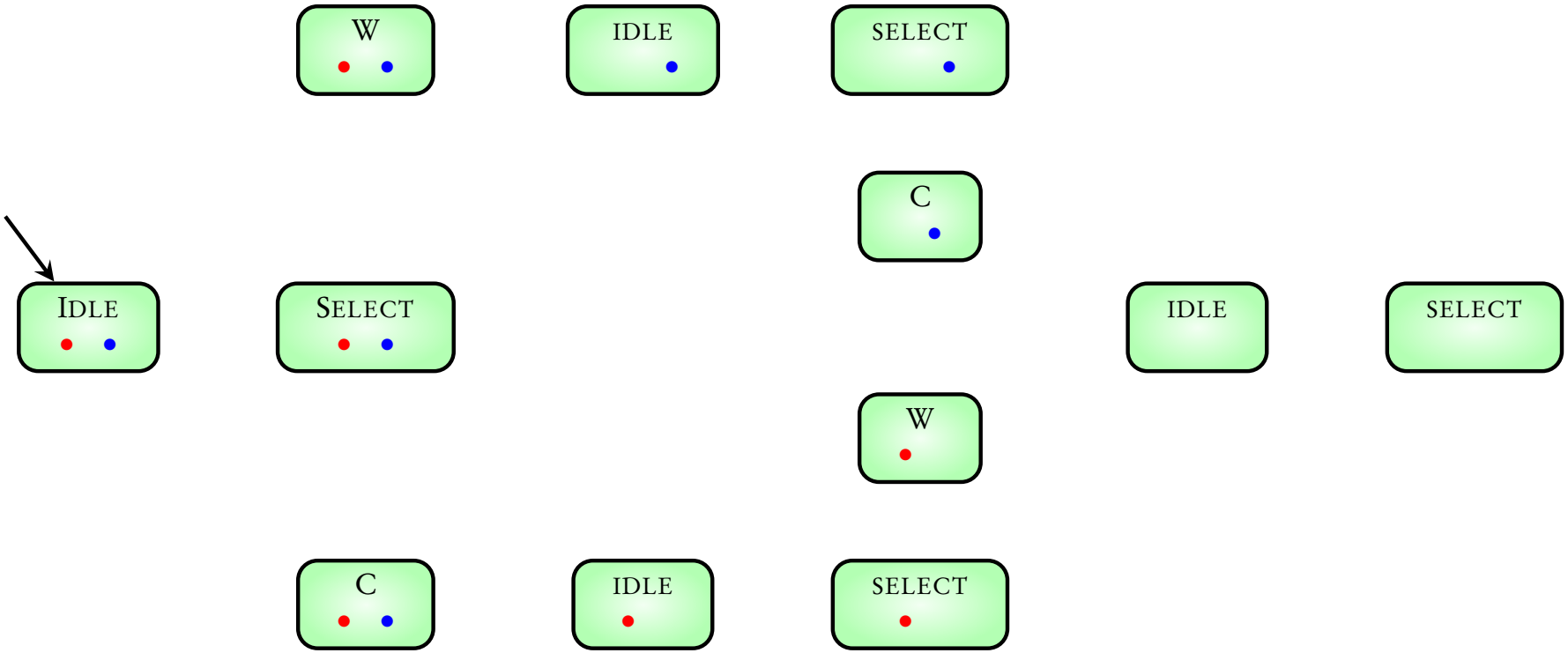
Coming next: Transition system corresponding to `max` $= 1$

$\texttt{max} = 1$

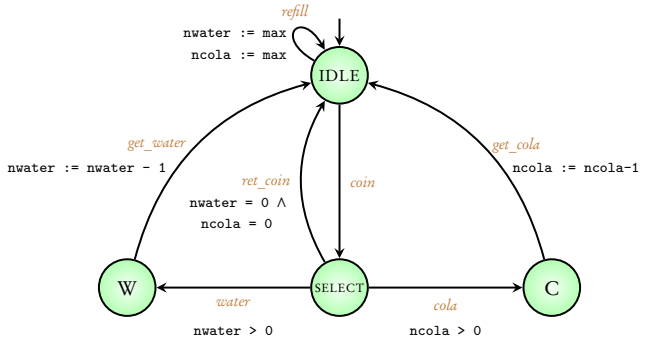

refill
nwater := max
ncola := max
IDLE
get_water
nwater := nwater - 1
ret_coin
nwater = 0 ∧
ncola = 0
coin
get_cola
ncola := ncola-1
W
SELECT
C
water
nwater > 0
cola
ncola > 0

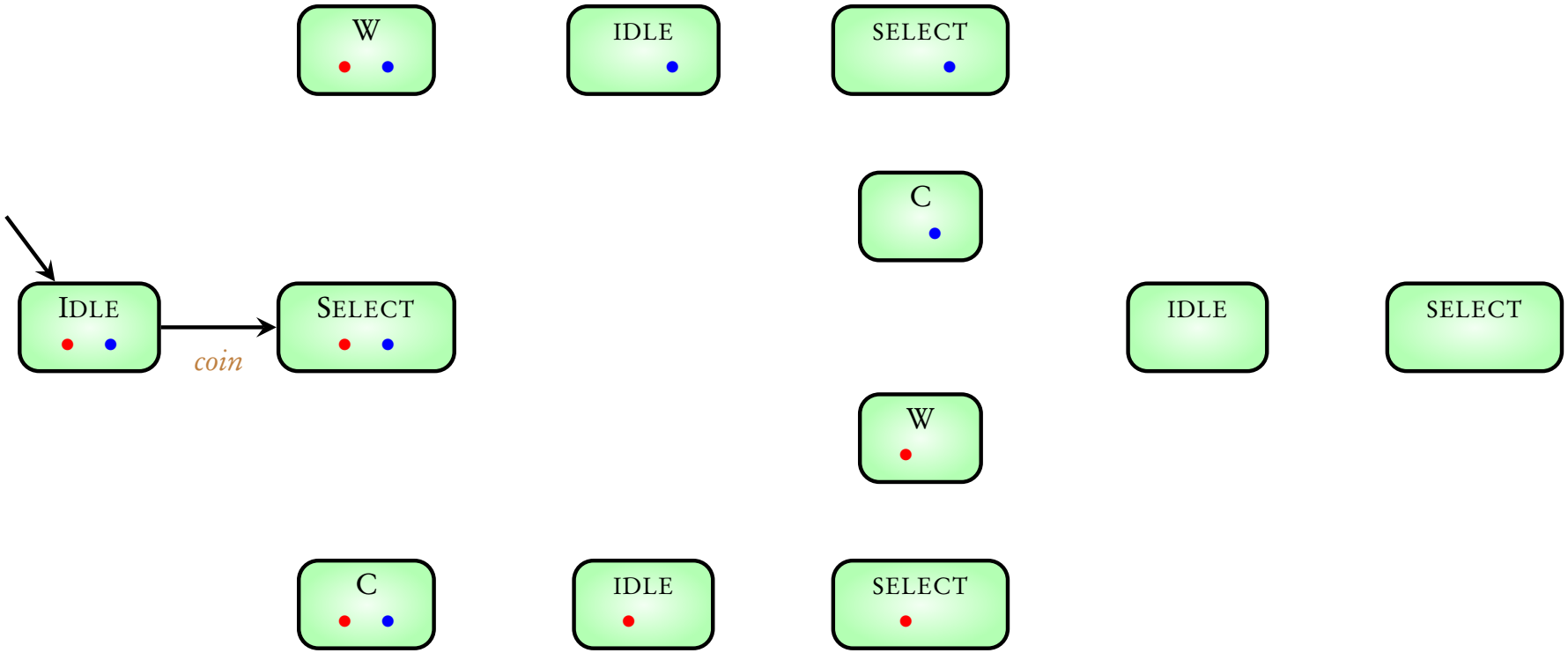
$\texttt{max} = 1$


refill
nwater := max
ncola := max
IDLE
get_water
nwater := nwater - 1
ret_coin
nwater = 0 ∧
ncola = 0
coin
get_cola
ncola := ncola-1
W
SELECT
C
water
nwater > 0
cola
ncola > 0

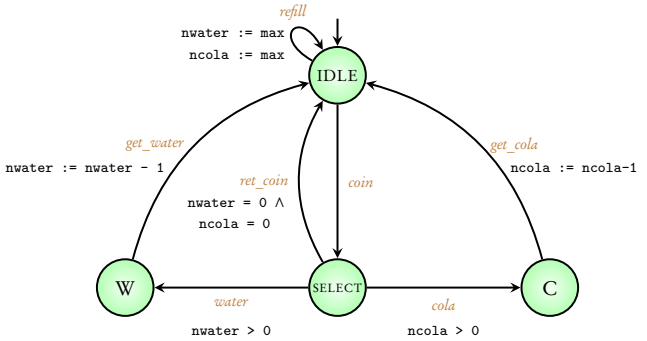

W
IDLE
SELECT
C
IDLE
SELECT
IDLE
SELECT
W
C
IDLE
SELECT

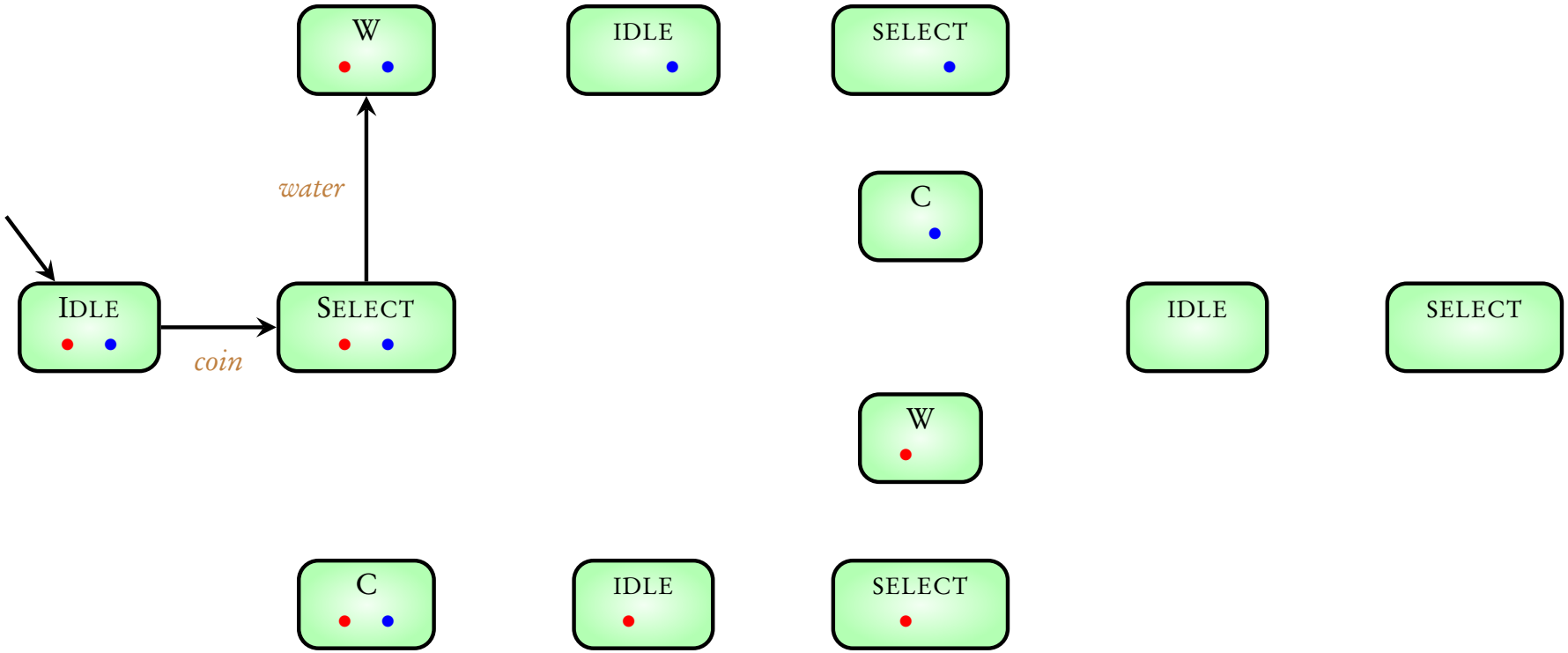
$\texttt{max} = 1$


refill
nwater := max
ncola := max
IDLE
get_water
nwater := nwater - 1
ret_coin
nwater = 0 ∧
ncola = 0
coin
get_cola
ncola := ncola-1
W
SELECT
C
water
nwater > 0
cola
ncola > 0

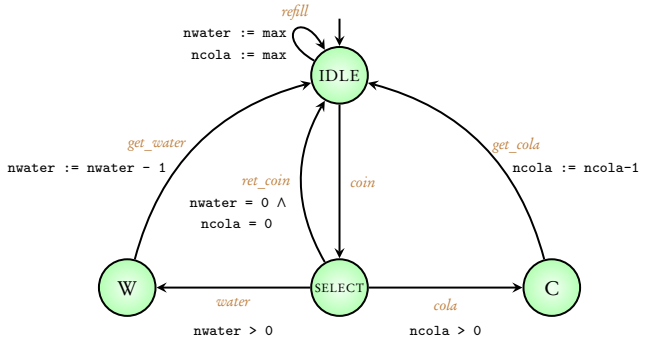

W
IDLE
SELECT
C
IDLE
SELECT
coin
IDLE
SELECT
W
C
IDLE
SELECT

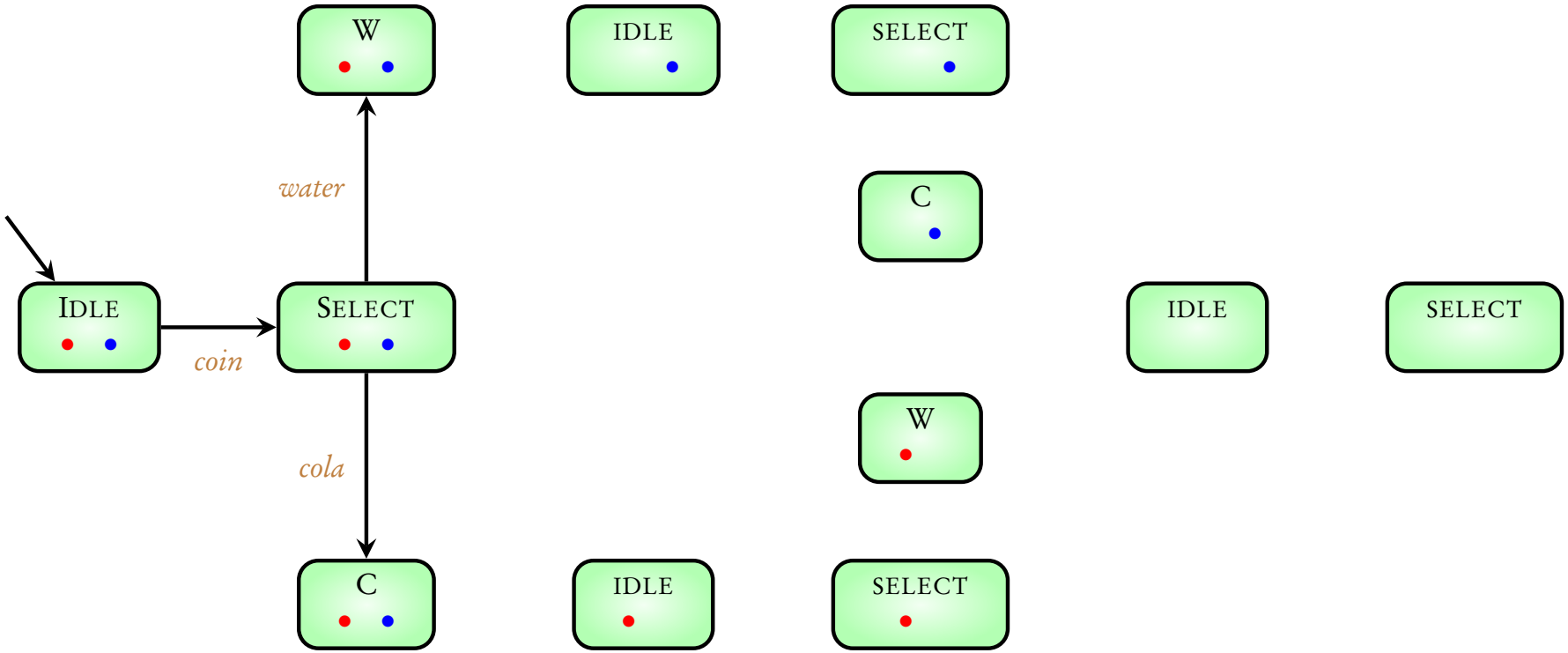
$\texttt{max} = 1$


refill
nwater := max
ncola := max
IDLE
get_water
nwater := nwater - 1
ret_coin
nwater = 0 ∧
ncola = 0
coin
get_cola
ncola := ncola-1
W
SELECT
C
water
nwater > 0
cola
ncola > 0

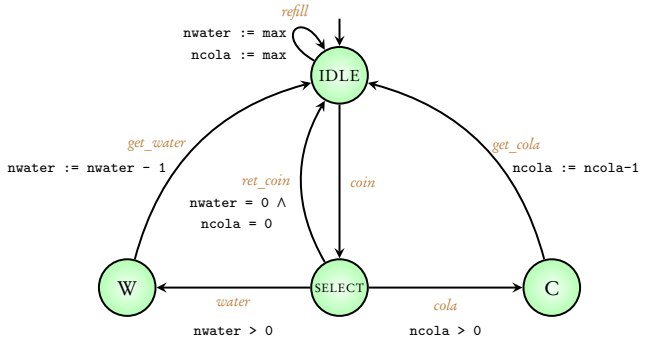

W
IDLE
SELECT
water
C
IDLE
SELECT
IDLE
SELECT
coin
W
C
IDLE
SELECT

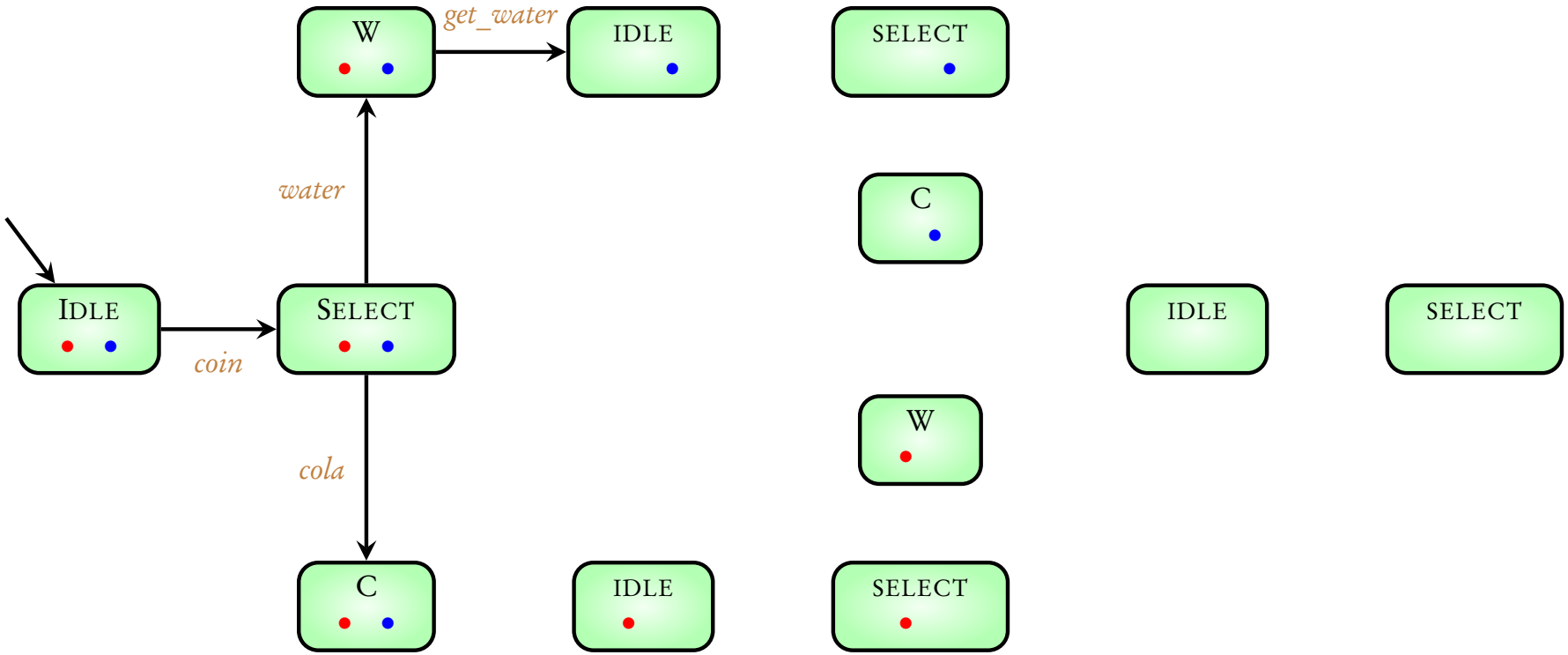
$\texttt{max} = 1$


refill
nwater := max
ncola := max
IDLE
get_water
nwater := nwater - 1
ret_coin
nwater = 0 ∧
ncola = 0
coin
get_cola
ncola := ncola-1
W
SELECT
C
water
nwater > 0
cola
ncola > 0

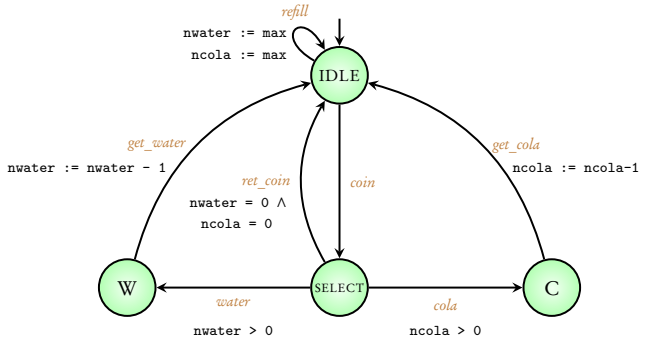

W
IDLE
SELECT
water
C
IDLE
SELECT
IDLE
coin
SELECT
W
cola
C
IDLE
SELECT

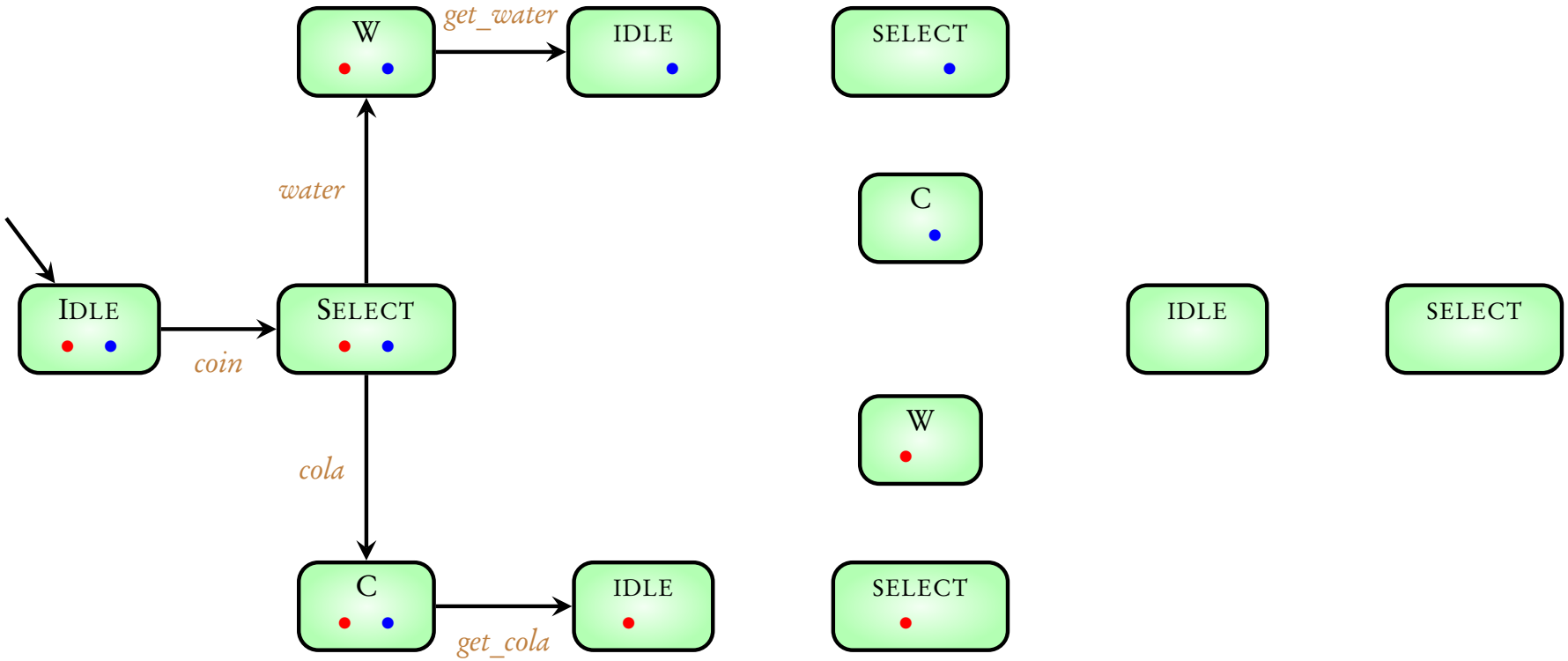
$\texttt{max} = 1$


refill
nwater := max
ncola := max
IDLE
get_water
nwater := nwater - 1
ret_coin
nwater = 0 ∧
ncola = 0
coin
get_cola
ncola := ncola-1
W
SELECT
C
water
nwater > 0
cola
ncola > 0

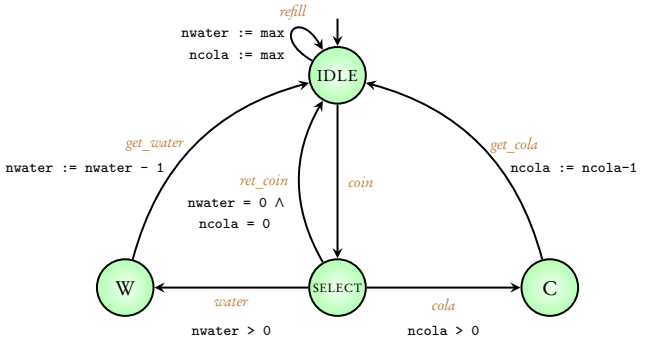

W
get_water
IDLE
SELECT
water
C
IDLE
SELECT
coin
W
cola
C
IDLE
SELECT

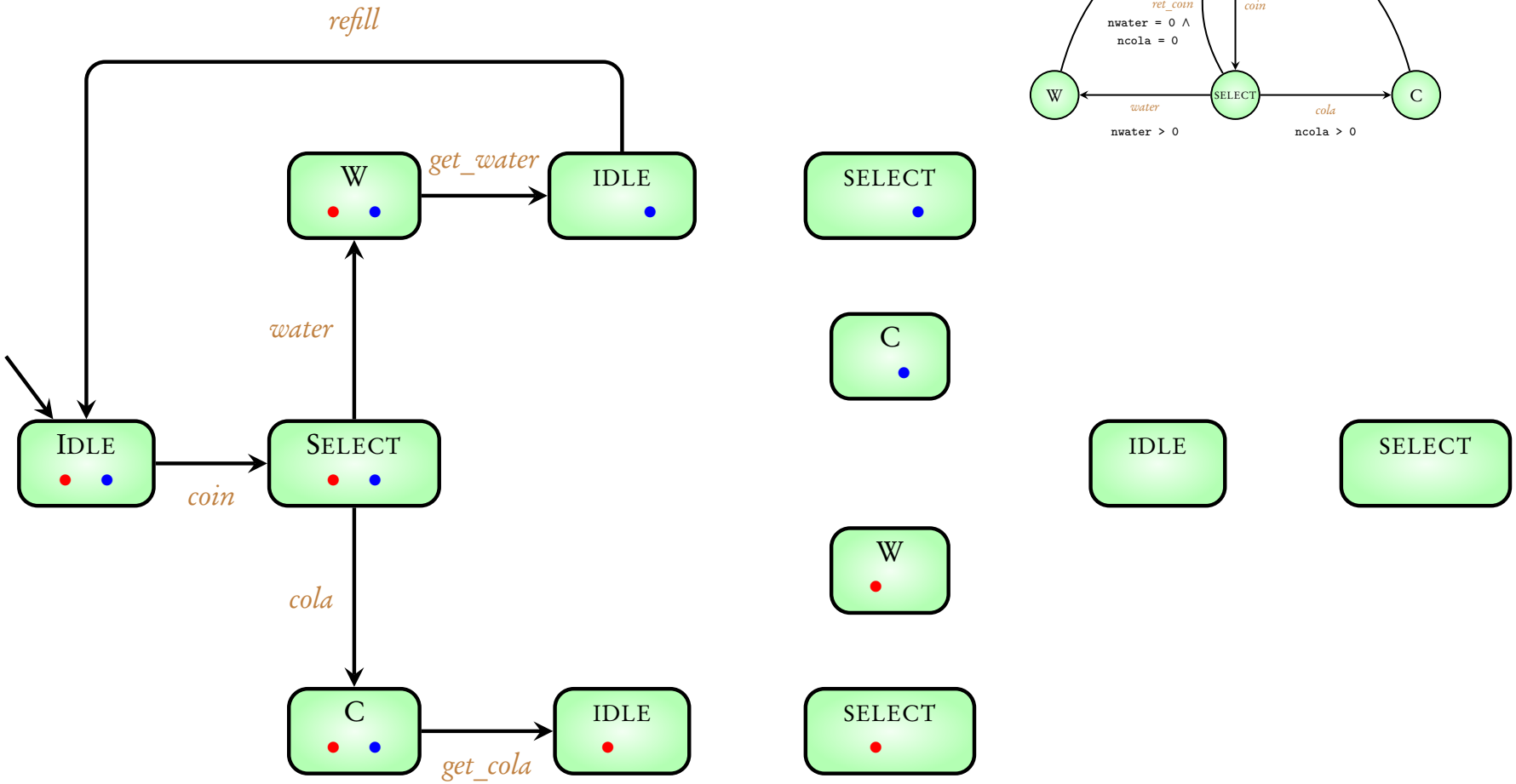
$\texttt{max} = 1$


refill
nwater := max
ncola := max
IDLE
get_water
nwater := nwater - 1
ret_coin
nwater = 0 ∧
ncola = 0
coin
get_cola
ncola := ncola-1
W
SELECT
C
water
nwater > 0
cola
ncola > 0

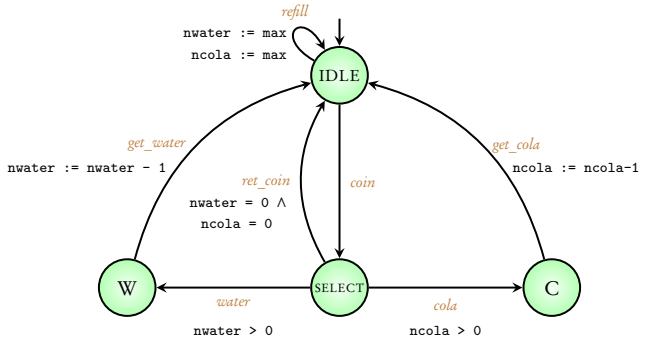

W
get_water
IDLE
SELECT
water
C
IDLE
SELECT
coin
W
cola
C
IDLE
SELECT
get_cola

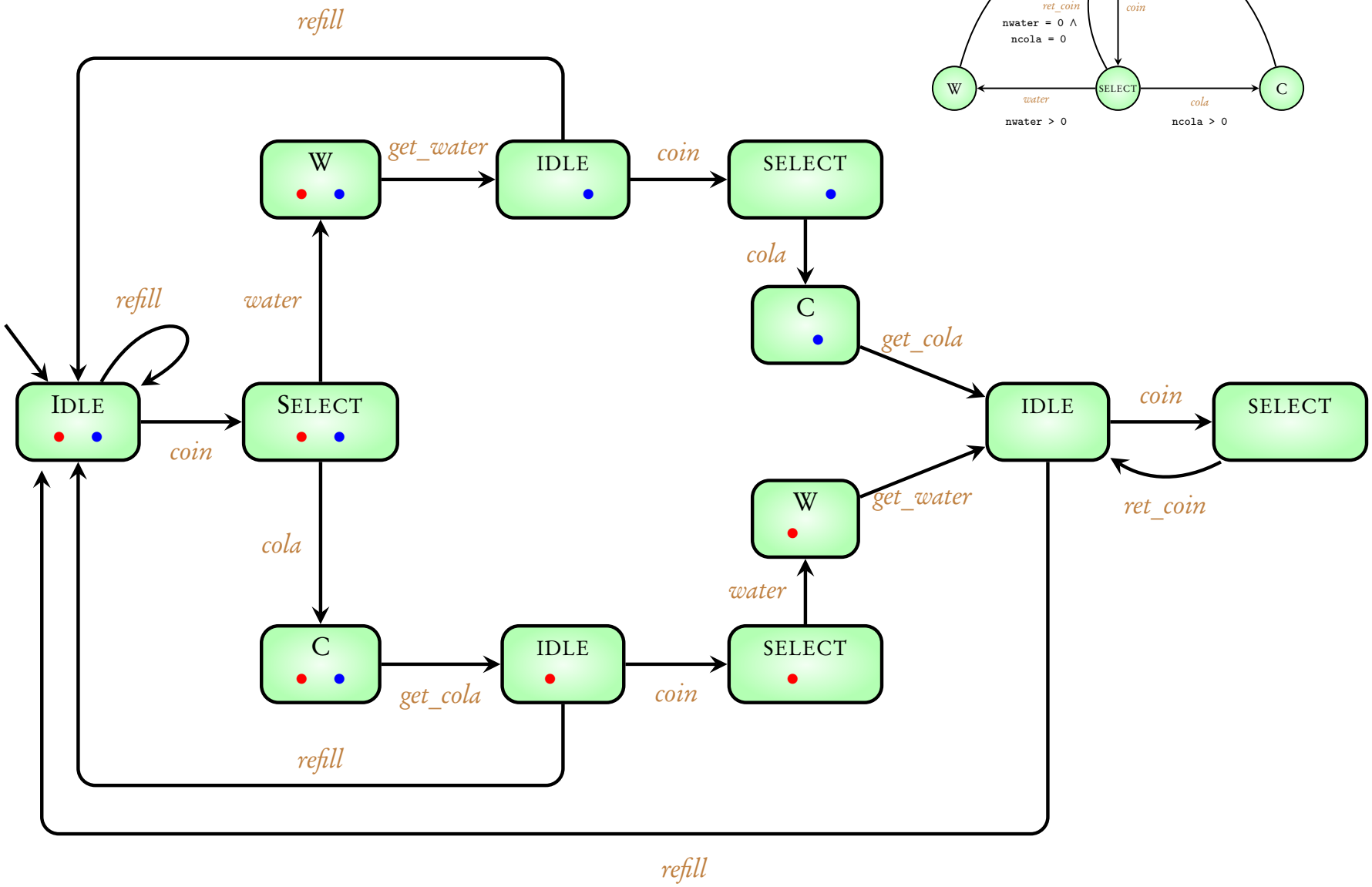
$\texttt{max} = 1$


refill
nwater := max
ncola := max
IDLE
get_water
nwater := nwater - 1
ret_coin
nwater = 0 ∧
ncola = 0
coin
get_cola
ncola := ncola-1
W
SELECT
C
water
nwater > 0
cola
ncola > 0

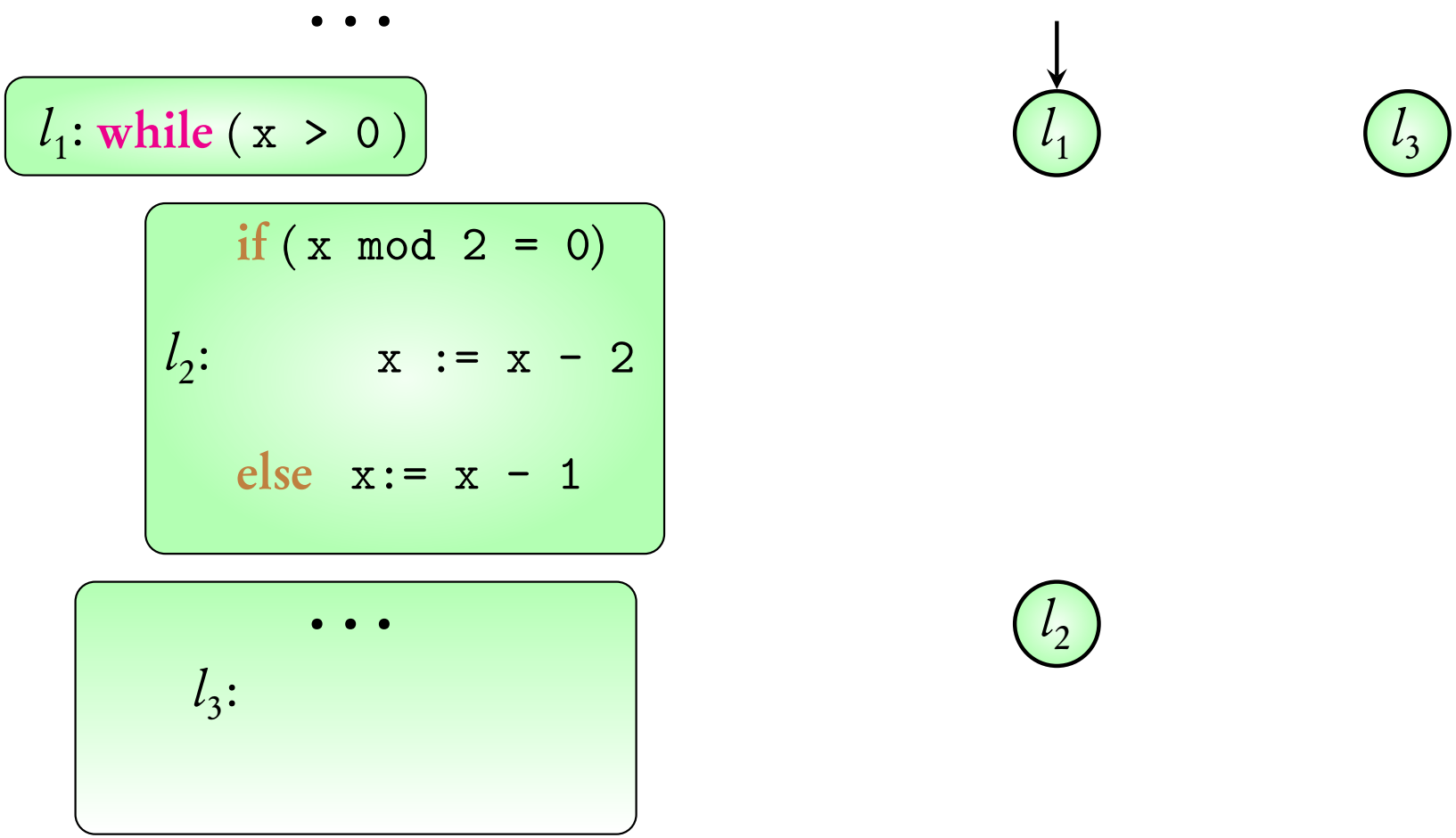

refill
W
get_water
IDLE
SELECT
water
C
IDLE
SELECT
coin
cola
W
C
IDLE
SELECT
get_cola

$\texttt{max} = 1$

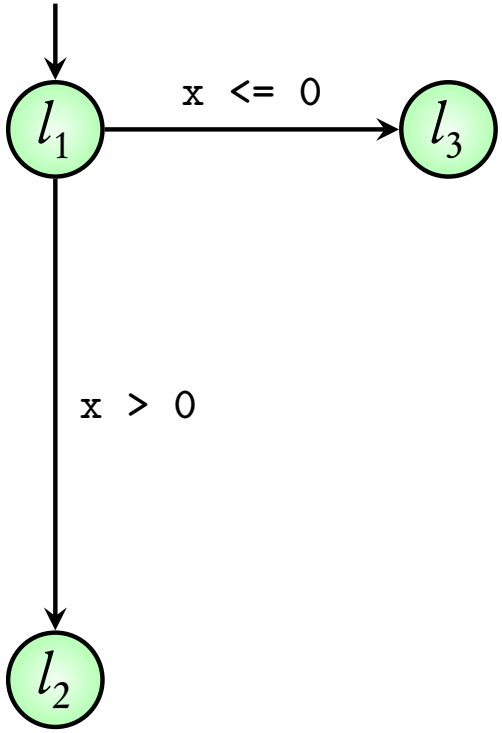

refill
nwater := max
ncola := max
IDLE
get_water
nwater := nwater - 1
ret_coin
nwater = 0 ∧
ncola = 0
coin
get_cola
ncola := ncola-1
W
SELECT
C
water
nwater > 0
cola
ncola > 0


refill
W
get_water
IDLE
coin
SELECT
cola
C
get_cola
refill
water
IDLE
SELECT
coin
coin
IDLE
SELECT
ret_coin
W
get_water
water
cola
C
IDLE
SELECT
get_cola
coin
refill
refill

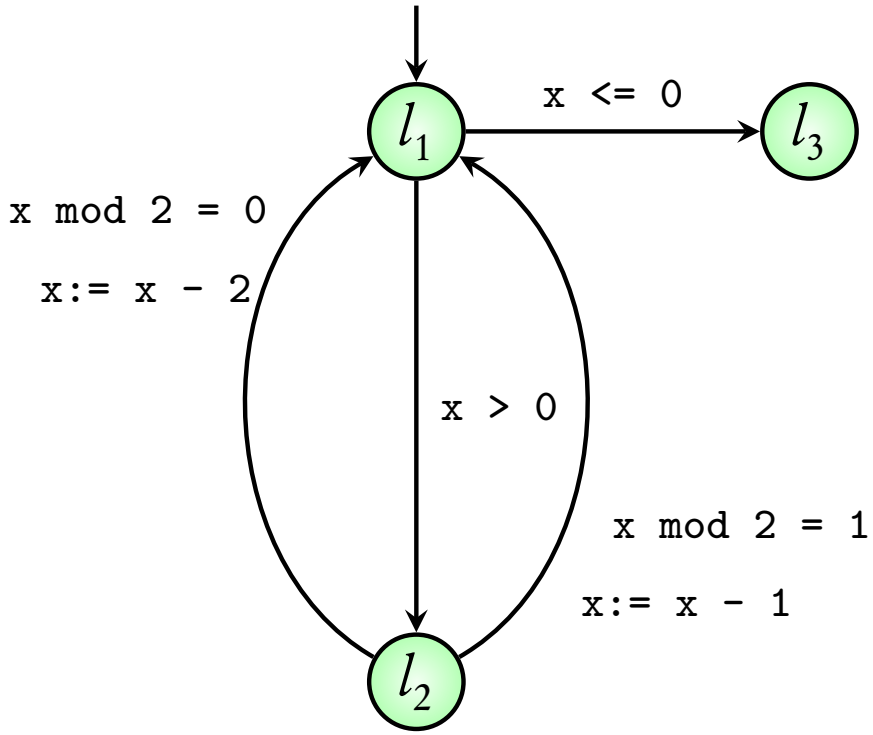
```
    ...
while (x > 0)
    if (x mod 2 = 0)
        x := x - 2
    else  x:= x - 1
    ...
```

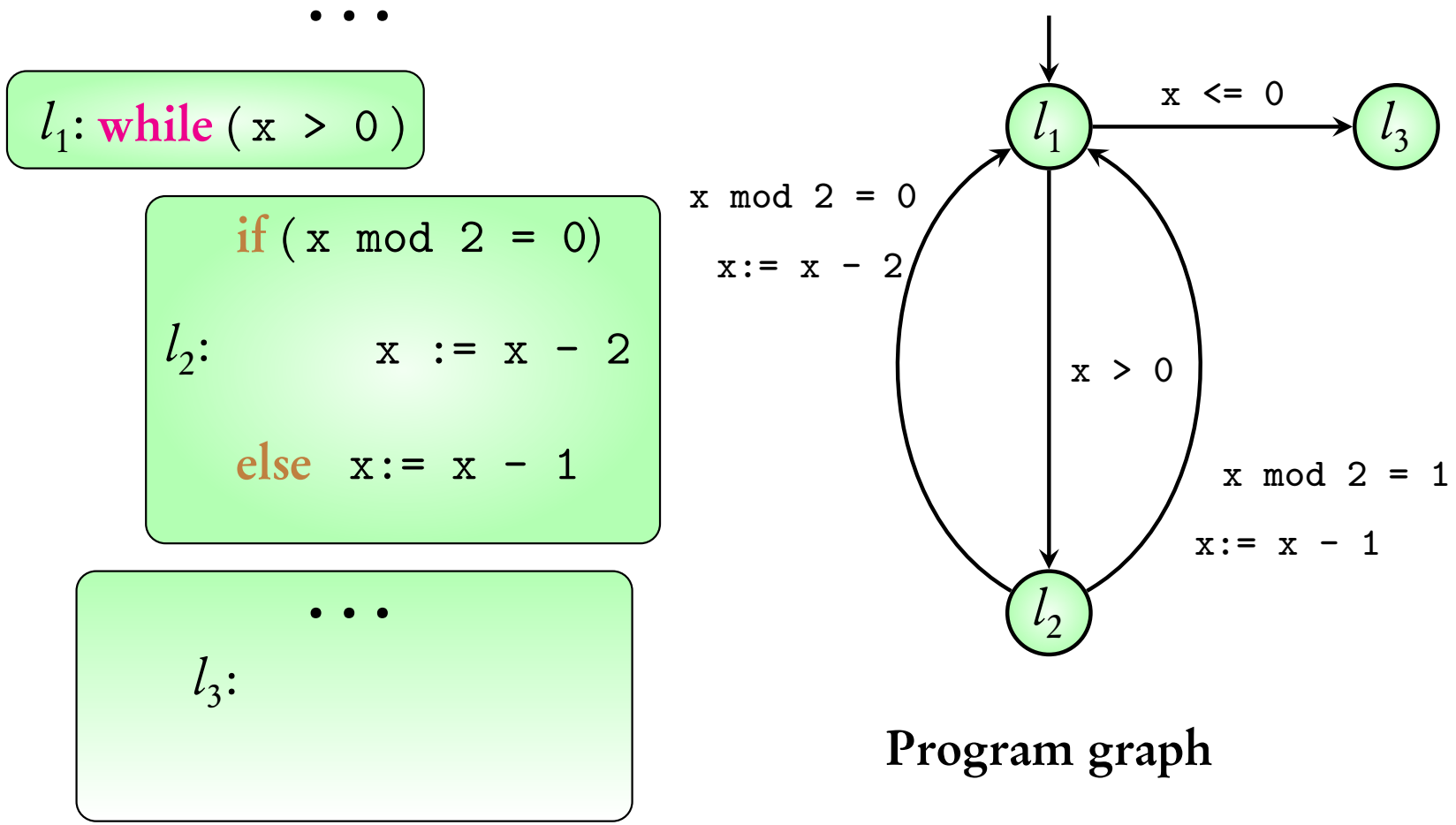

• • •
l_1: while (x > 0)
if (x mod 2 = 0)
l_2: x := x - 2
else x:= x - 1
• • •
l_3:
l_1
l_3
l_2

```
...
l1: while ( x > 0 )
      if ( x mod 2 = 0)
l2:       x := x - 2
      else  x:= x - 1
...
l3:
```



```
    . . .
l1: while ( x > 0 )
      if ( x mod 2 = 0)
l2:       x := x - 2
      else x:= x - 1
    . . .
l3:
```





Definition 2.13. Program Graph (PG)

A *program graph* PG over set Var of typed variables is a tuple $(Loc, Act, Effect, \hookrightarrow, Loc_0, g_0)$ where

- Loc is a set of locations and Act is a set of actions,
- $Effect : Act \times Eval(Var) \to Eval(Var)$ is the effect function,
- $\hookrightarrow \subseteq Loc \times Cond(Var) \times Act \times Loc$ is the conditional transition relation,
- $Loc_0 \subseteq Loc$ is a set of initial locations,
- $g_0 \in Cond(Var)$ is the initial condition.

Definition 2.15. Transition System Semantics of a Program Graph

The transition system $TS(PG)$ of program graph

$$PG = (Loc, Act, Effect, \hookrightarrow, Loc_0, g_0)$$

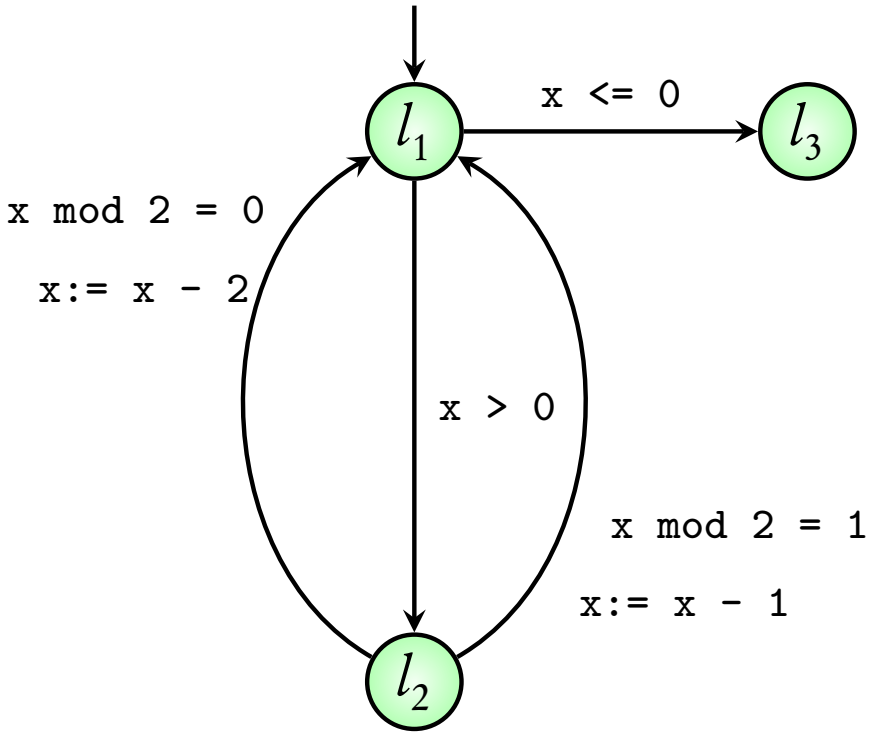
over set Var of variables is the tuple $(S, Act, \longrightarrow, I, AP, L)$ where

- $S = Loc \times Eval(Var)$
- $\longrightarrow \subseteq S \times Act \times S$ is defined by the following rule (see remark below):

$$\frac{\ell \overset{g:\alpha}{\hookrightarrow} \ell' \quad \wedge \quad \eta \models g}{\langle \ell, \eta \rangle \xrightarrow{\alpha} \langle \ell', Effect(\alpha, \eta) \rangle}$$

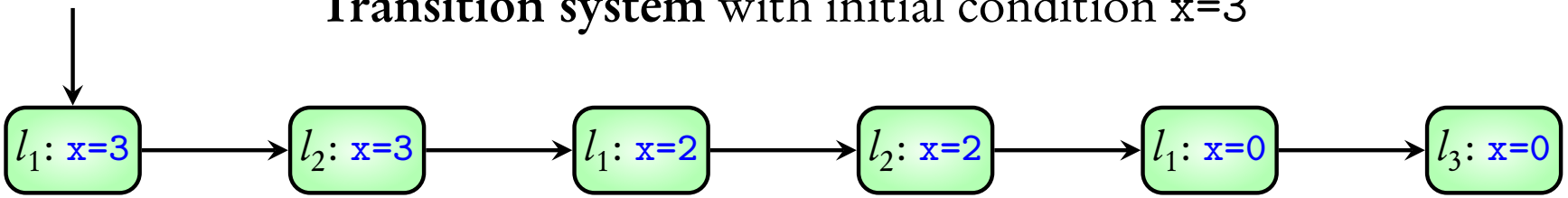
- $I = \{\langle \ell, \eta \rangle \mid \ell \in Loc_0, \eta \models g_0\}$
- $AP = Loc \cup Cond(Var)$
- $L(\langle \ell, \eta \rangle) = \{\ell\} \cup \{g \in Cond(Var) \mid \eta \models g\}$.

```
. . .
l1: while ( x > 0 )
      if ( x mod 2 = 0)
l2:       x := x - 2
      else  x:= x - 1
. . .
l3:
```



Program graph

Transition system with initial condition x=3



Module 4:

Modeling concurrent systems

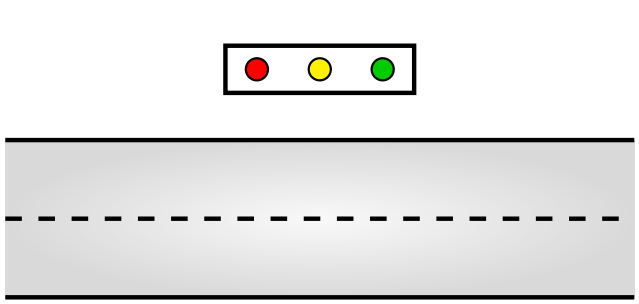
$$TS = TS_1 \parallel TS_2 \parallel \ldots \parallel TS_n$$

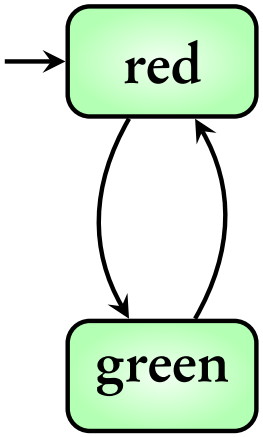
Concurrent systems

Independent

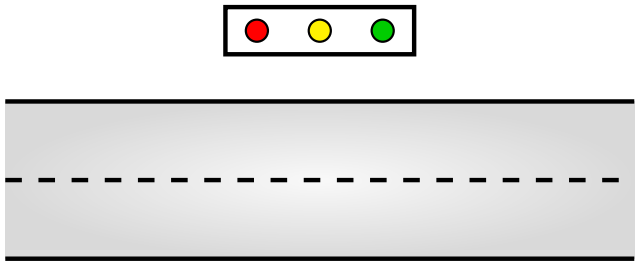
Shared variables

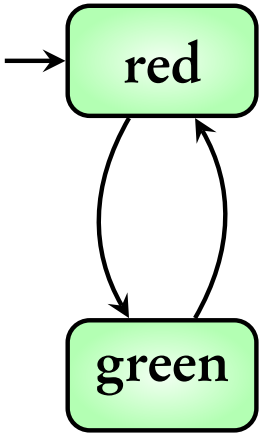
Shared actions



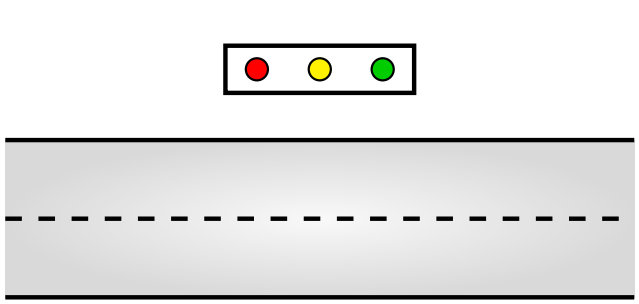


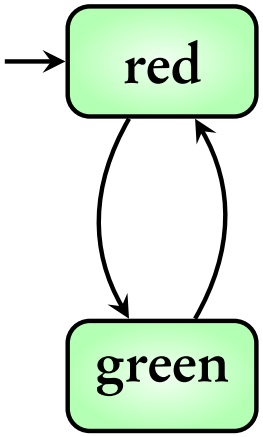
$TrLight_1$



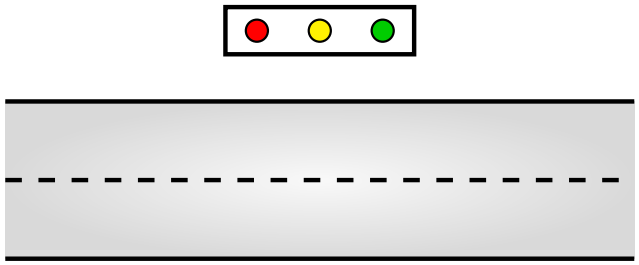


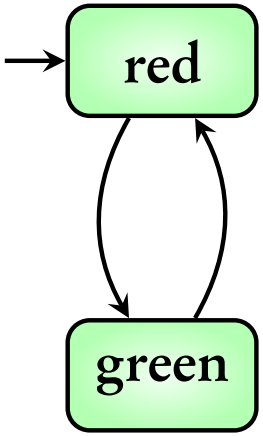
$TrLight_2$





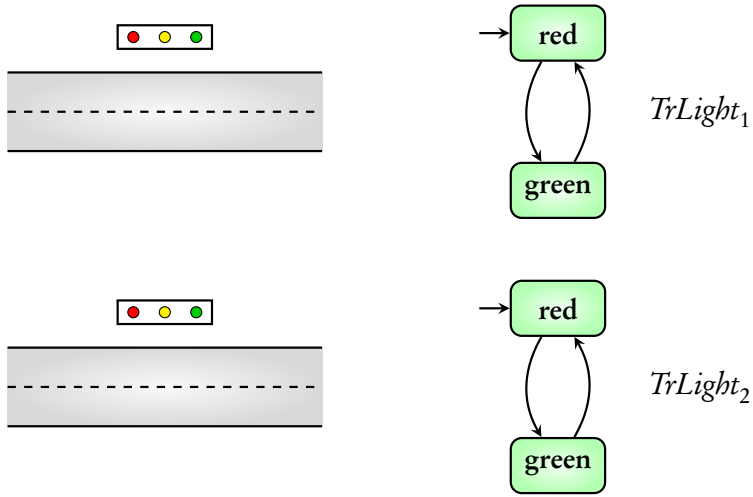
$TrLight_1$

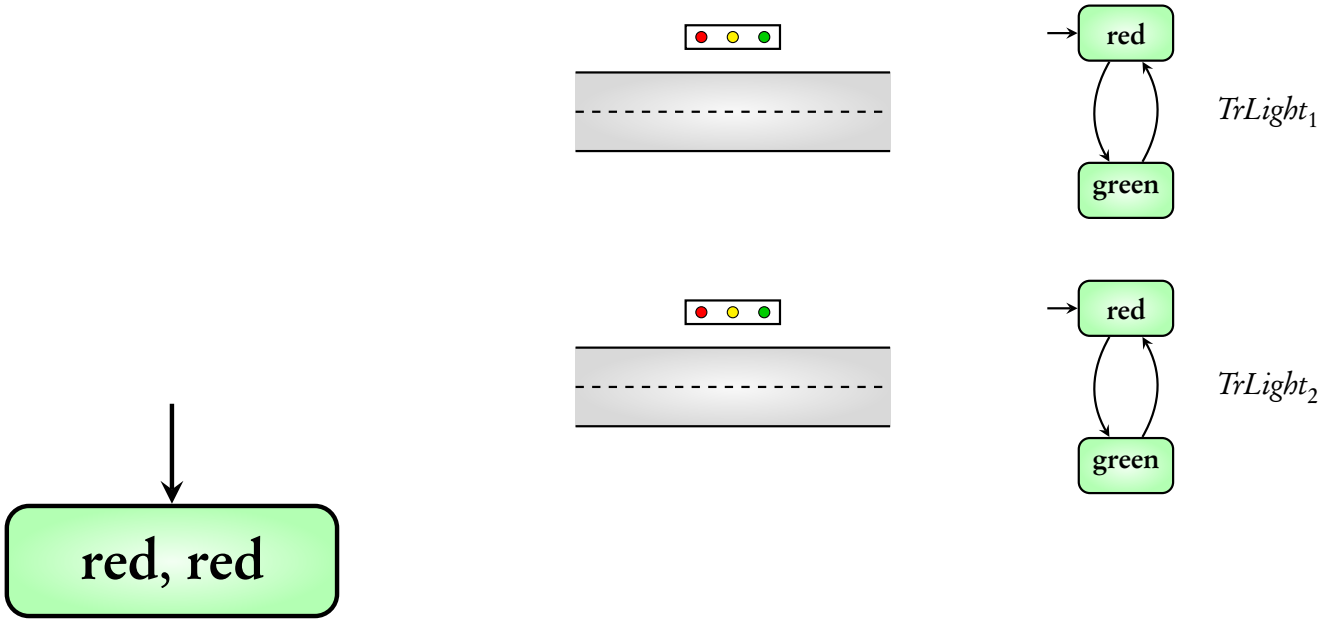


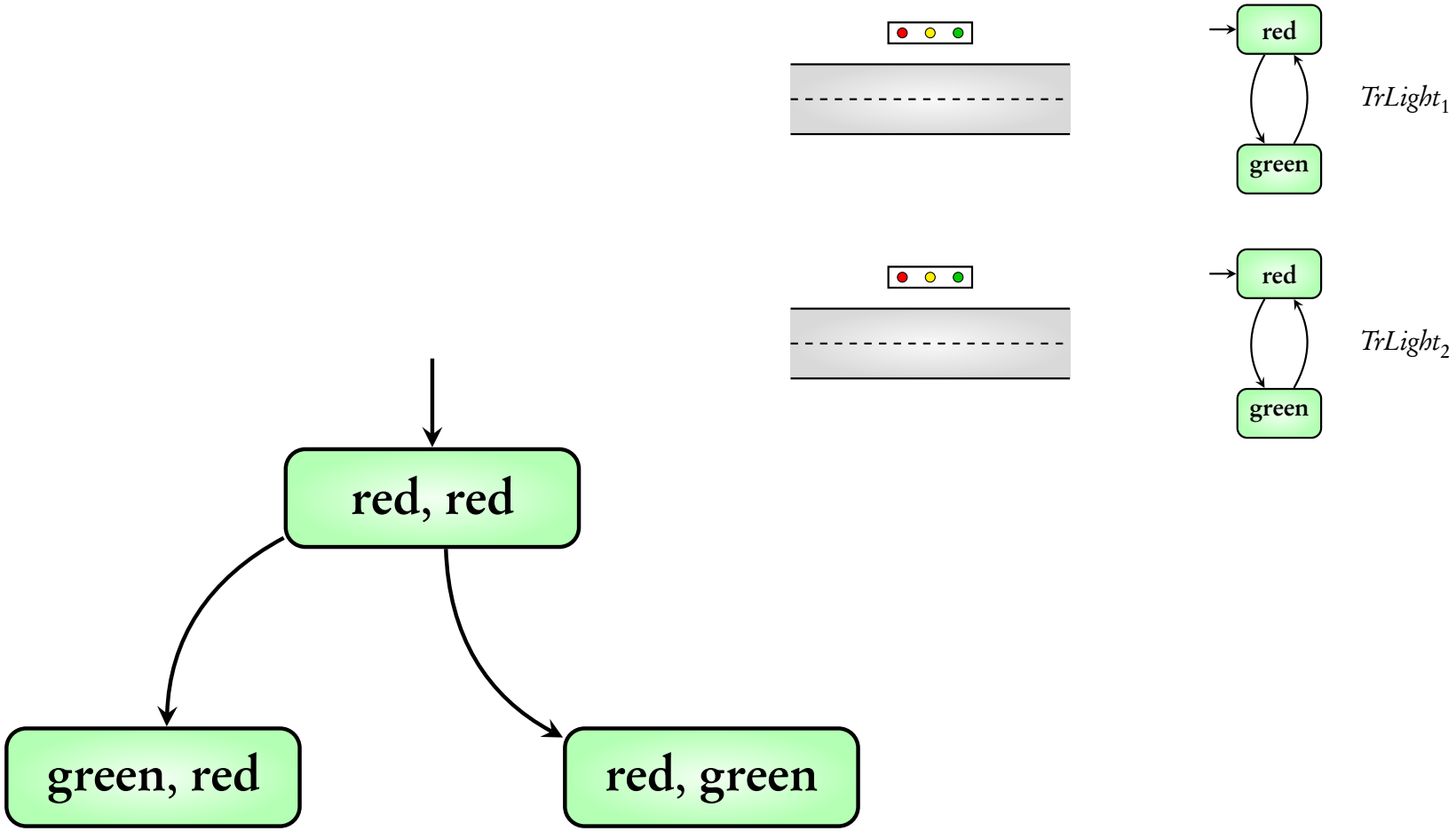


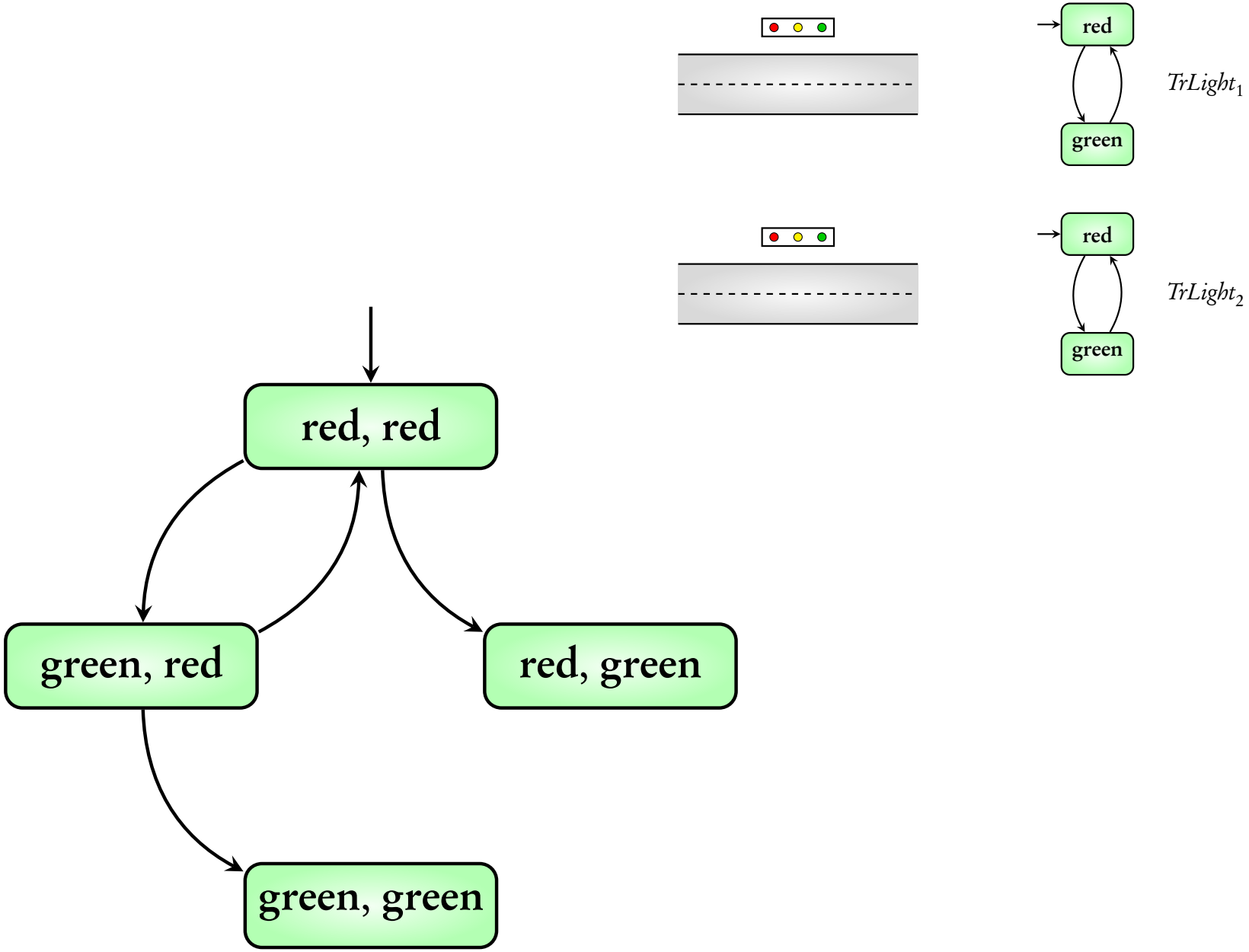
$TrLight_2$

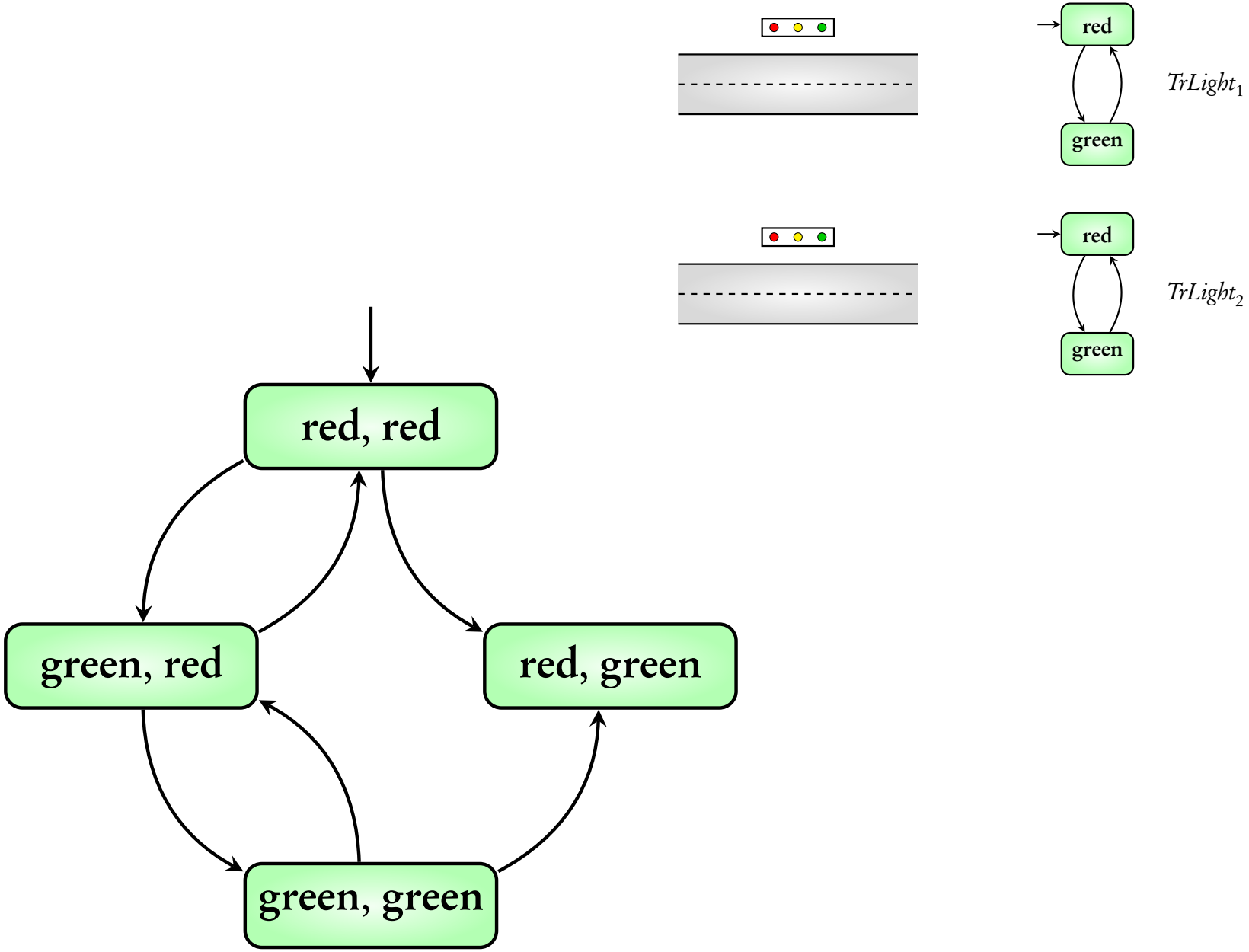
What is the transition system for the **joint behaviour**?

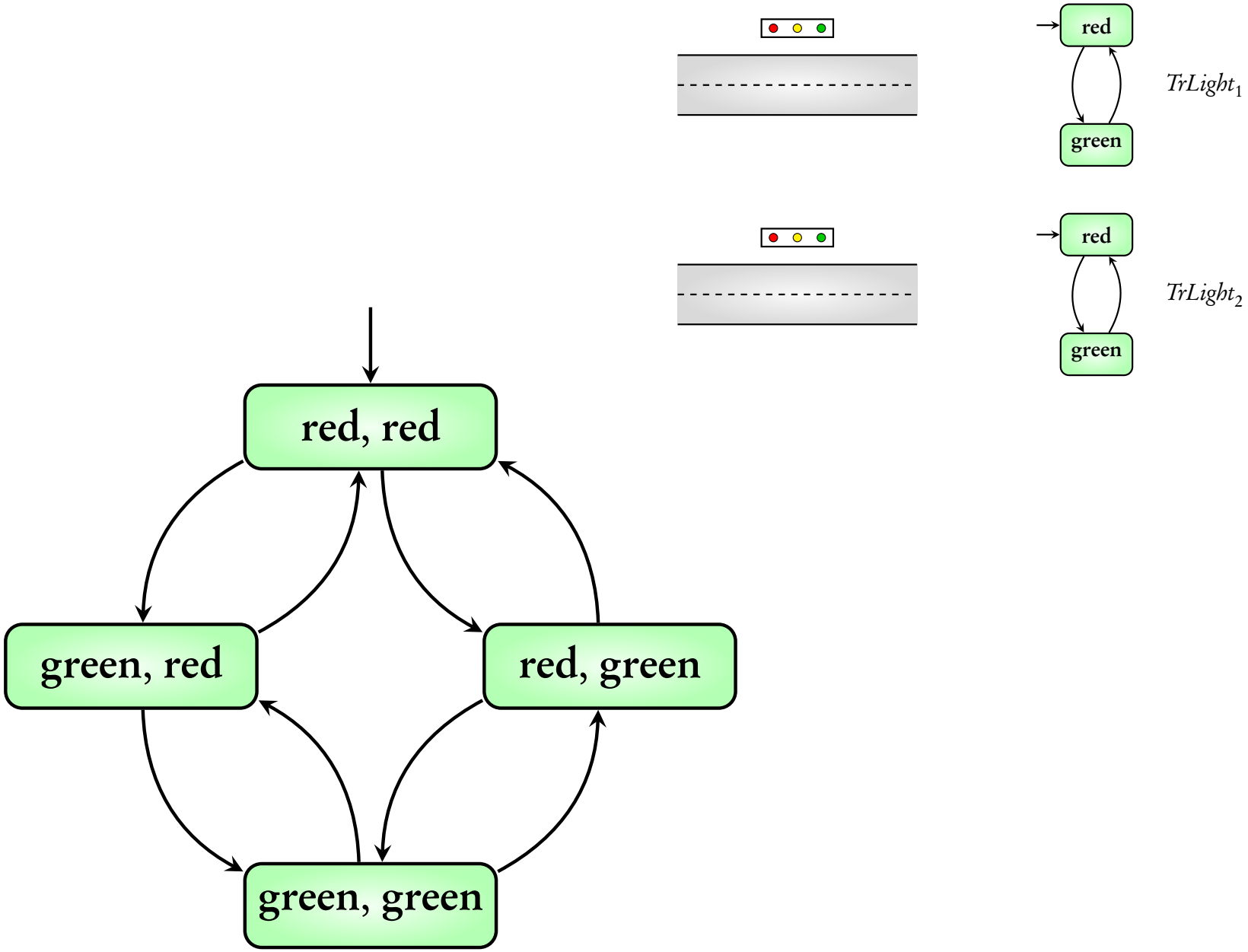

red
green
$TrLight_1$
red
green
$TrLight_2$

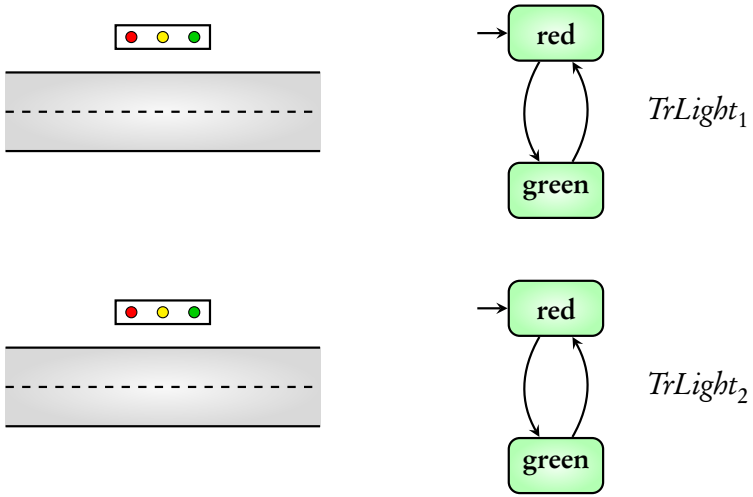

red
green
$TrLight_1$
red
green
$TrLight_2$
red, red

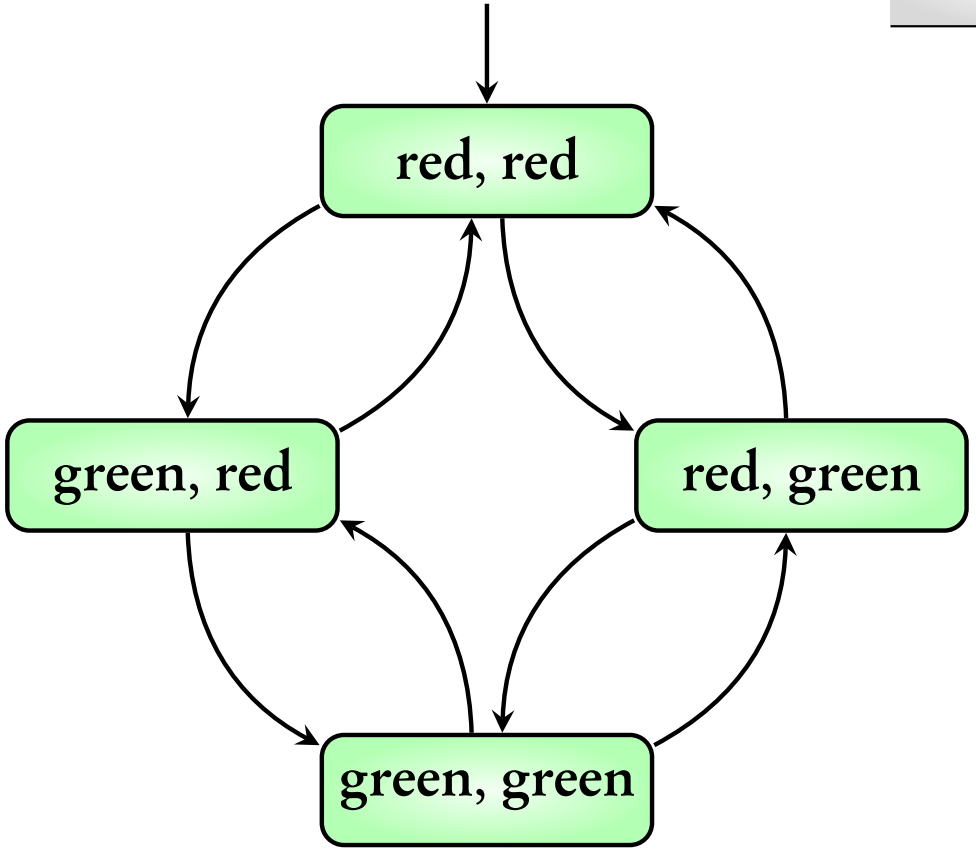

red
green
$TrLight_1$
red
green
$TrLight_2$
red, red
green, red
red, green


red
green
$TrLight_1$
red
green
$TrLight_2$
red, red
green, red
red, green
green, green

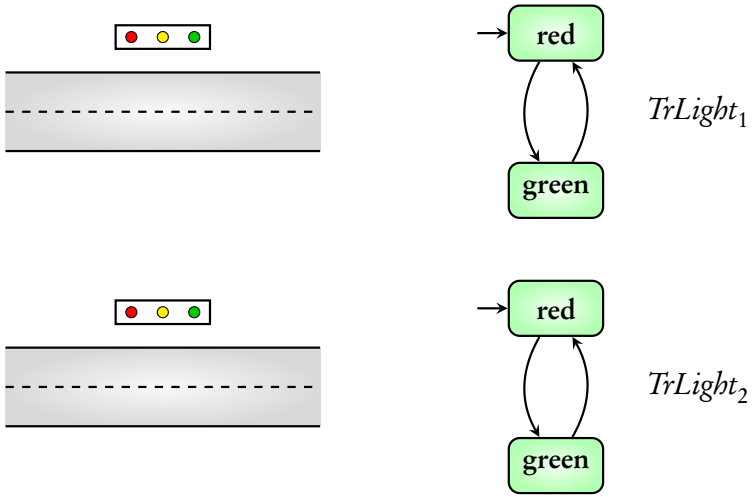

red
green
$TrLight_1$
red
green
$TrLight_2$
red, red
green, red
red, green
green, green

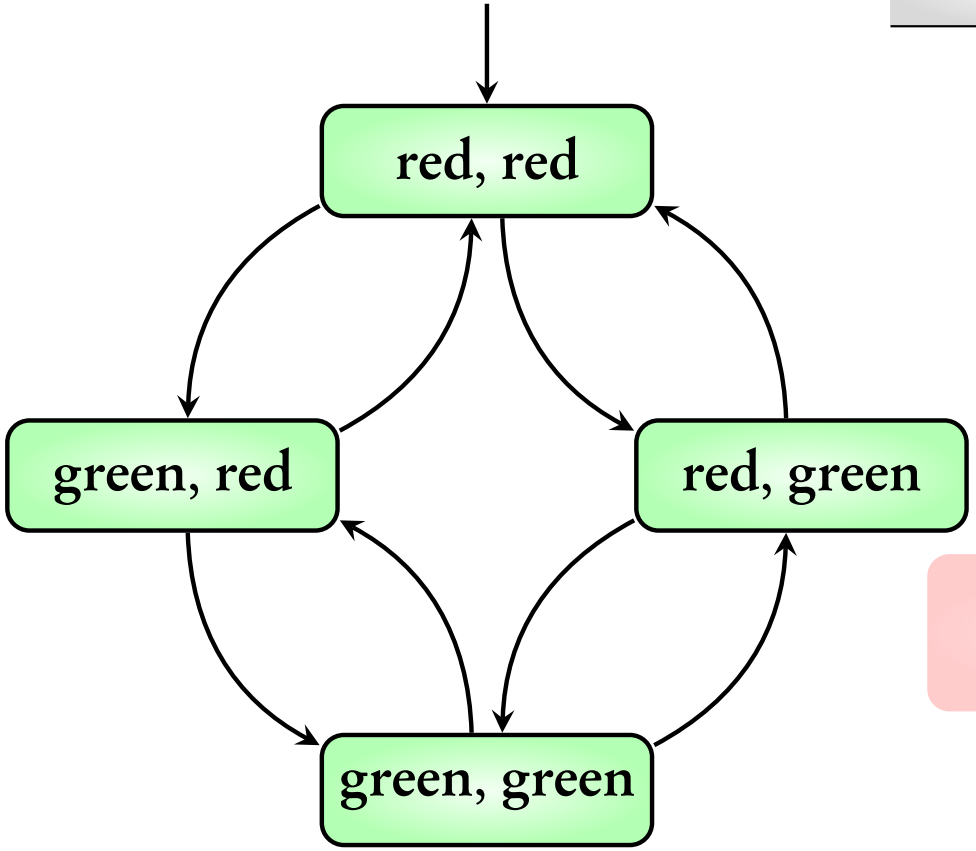

red
green
$TrLight_1$
red
green
$TrLight_2$
red, red
green, red
red, green
green, green





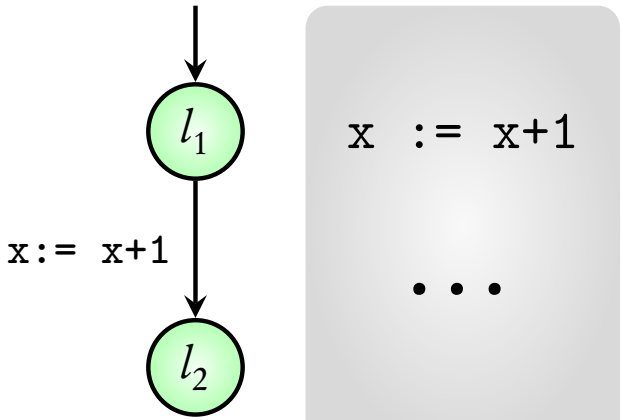
$$TrLight_1 \;|||\; TrLight_2$$



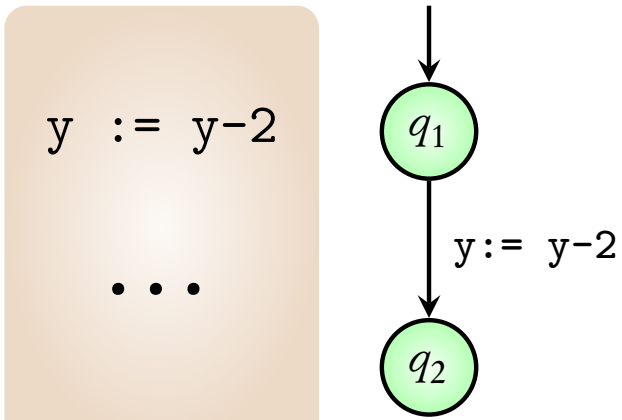


$$TrLight_1 \;|||\; TrLight_2$$

||| : Interleaving operator

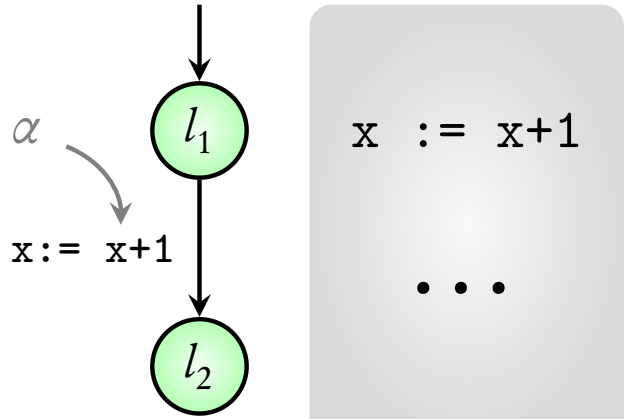


PG_1

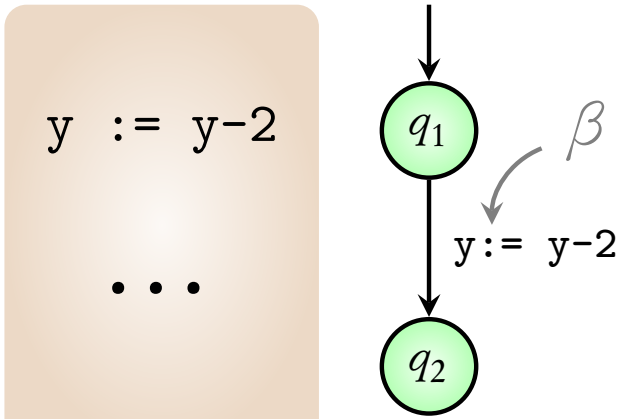


PG_2

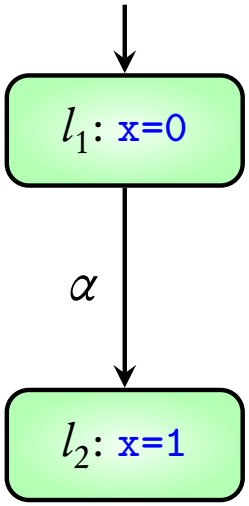
independent actions



PG_1

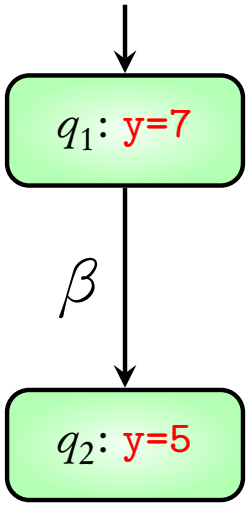


PG_2



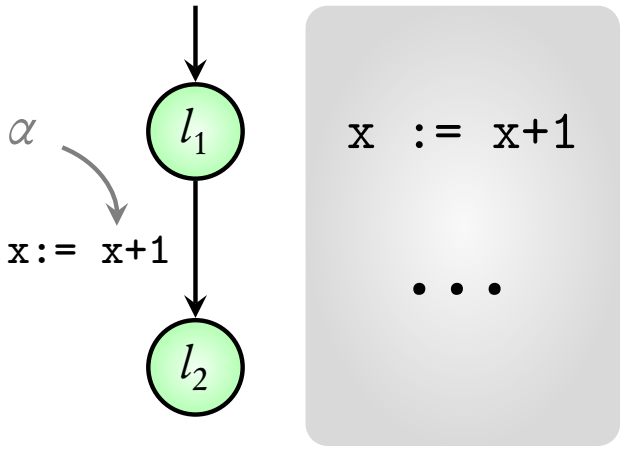
TS_1

(*initially* x=0)

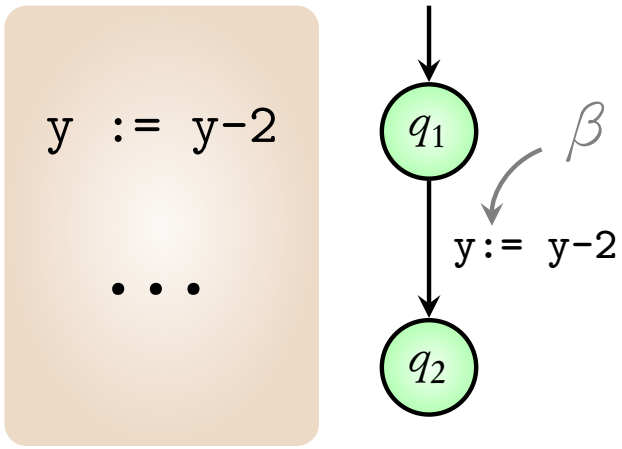


TS_2

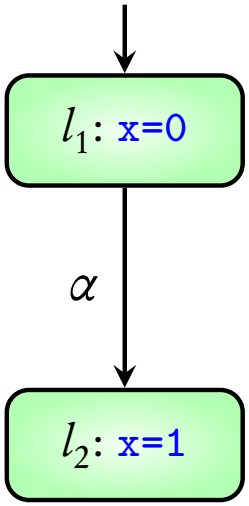
(*initially* y=7)



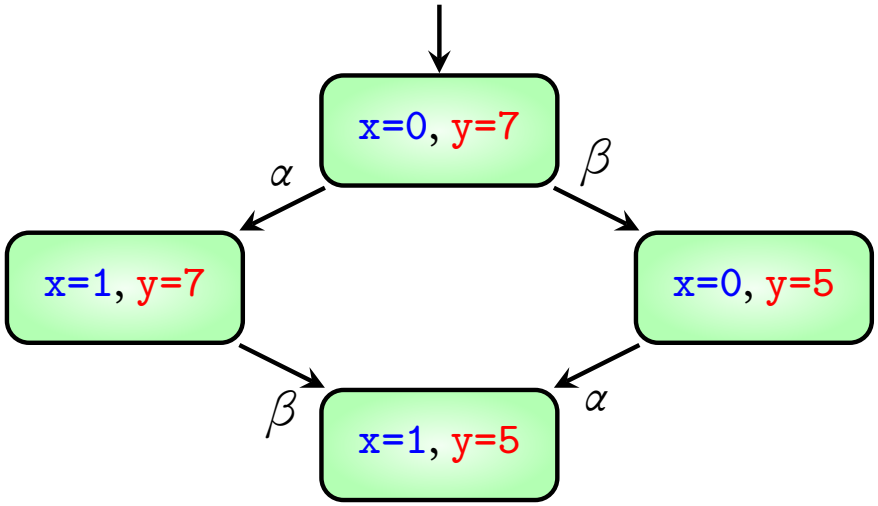
PG$_1$



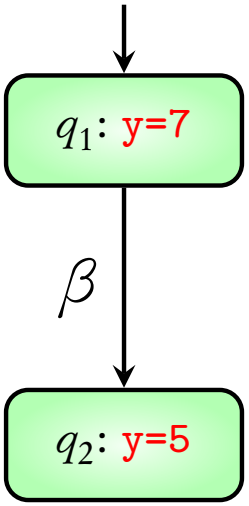
PG$_2$



TS$_1$
(*initially* x=0)

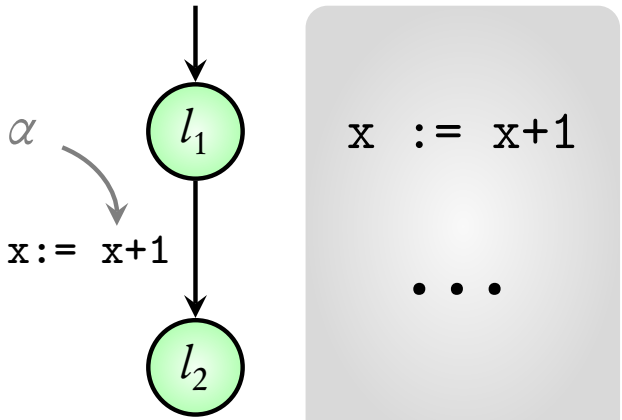


TS$_1$ ||| TS$_2$

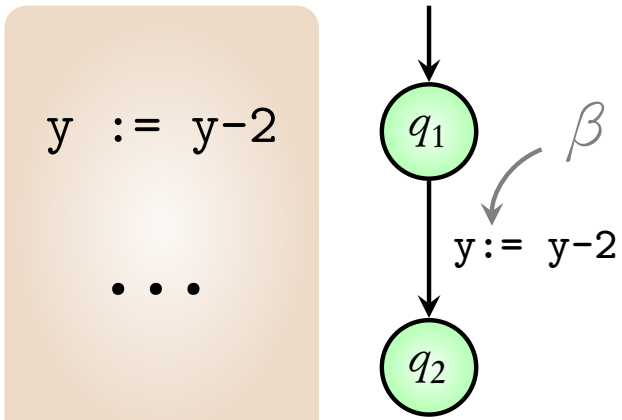


TS$_2$
(*initially* y=7)

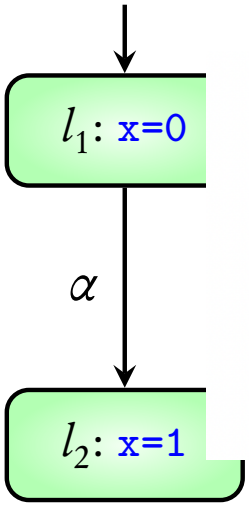
independent actions



PG_1



PG_2



TS_1

(*initially* x=0)

Definition 2.18. Interleaving of Transition Systems

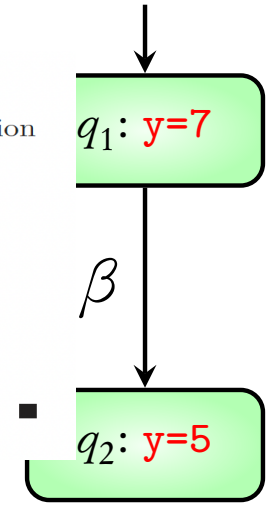
Let $TS_i = (S_i, Act_i, \rightarrow_i, I_i, AP_i, L_i)$ i=1, 2, be two transition systems. The transition system $TS_1 \,|||\, TS_2$ is defined by:

$$TS_1 \,|||\, TS_2 = (S_1 \times S_2, Act_1 \cup Act_2, \rightarrow, I_1 \times I_2, AP_1 \cup AP_2, L)$$

where the transition relation $\rightarrow$ is defined by the following rules:

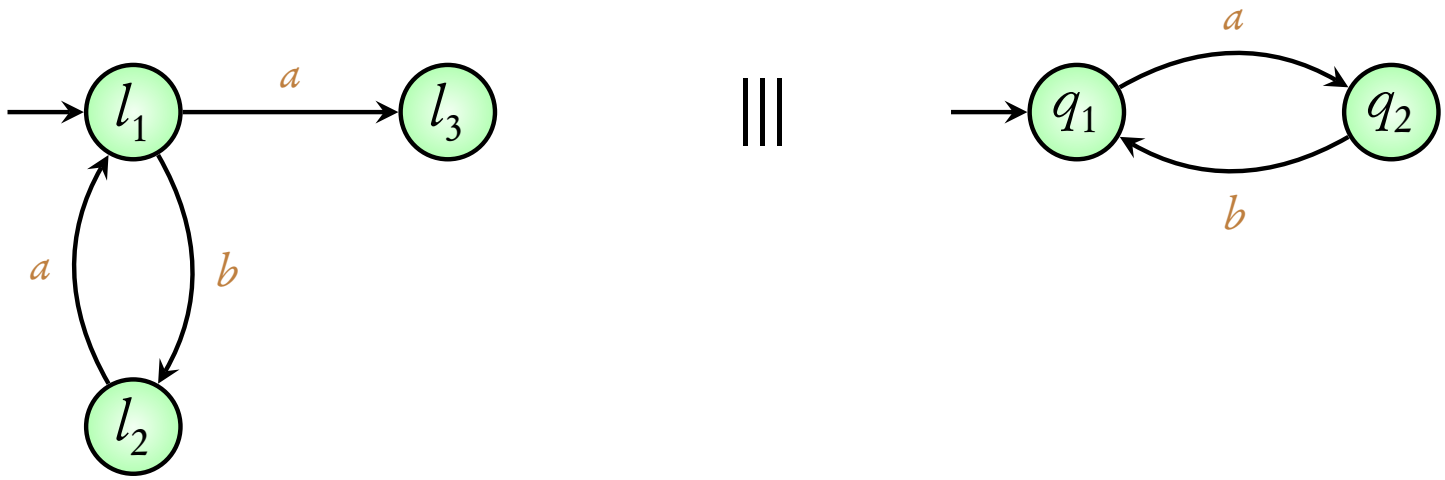
$$\frac{s_1 \xrightarrow{\alpha}_1 s_1'}{\langle s_1, s_2\rangle \xrightarrow{\alpha} \langle s_1', s_2\rangle} \quad \text{and} \quad \frac{s_2 \xrightarrow{\alpha}_2 s_2'}{\langle s_1, s_2\rangle \xrightarrow{\alpha} \langle s_1, s_2'\rangle}$$

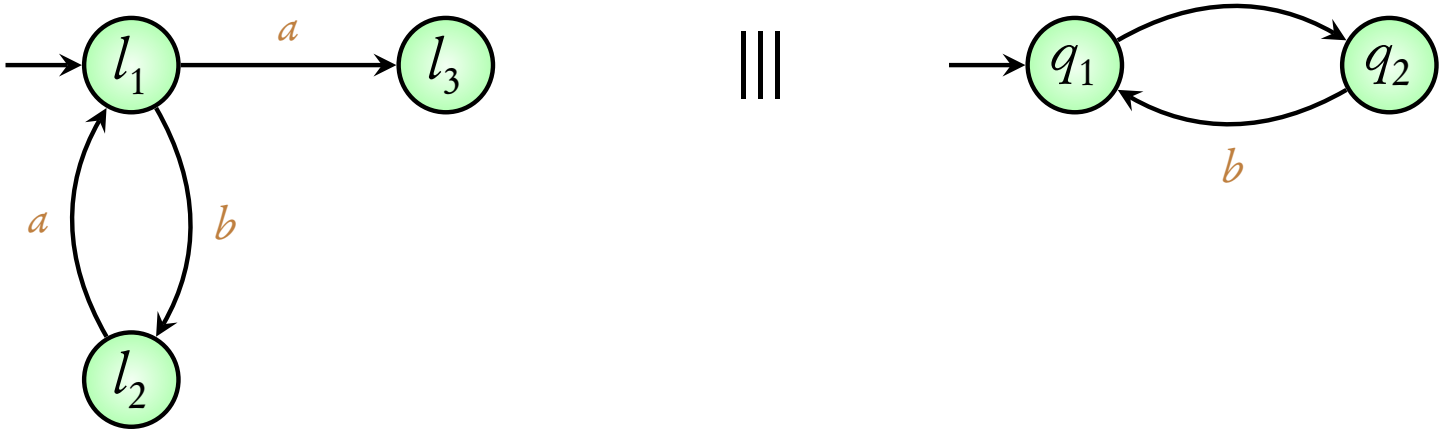
and the labeling function is defined by $L(\langle s_1, s_2\rangle) = L(s_1) \cup L(s_2)$.

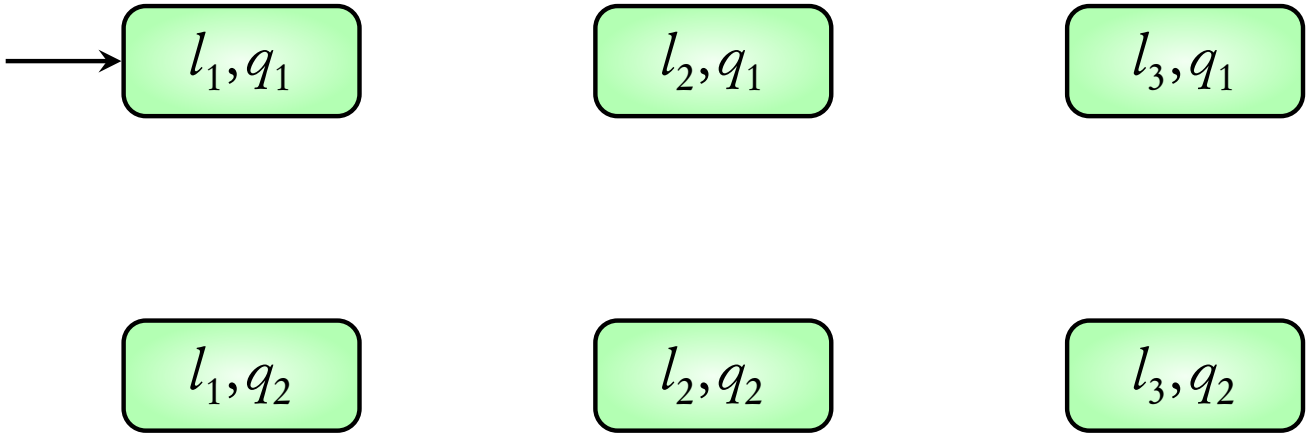


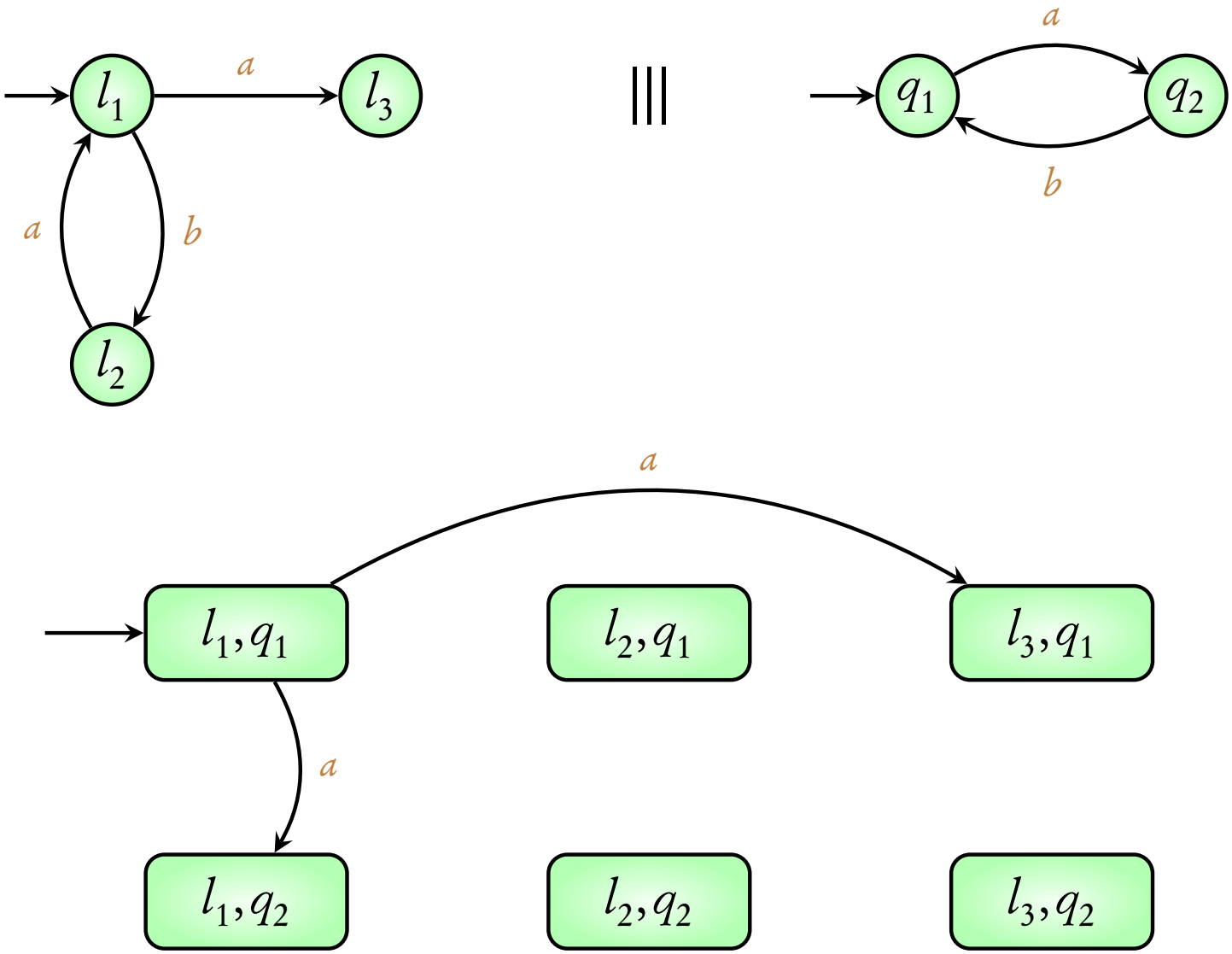
TS_2

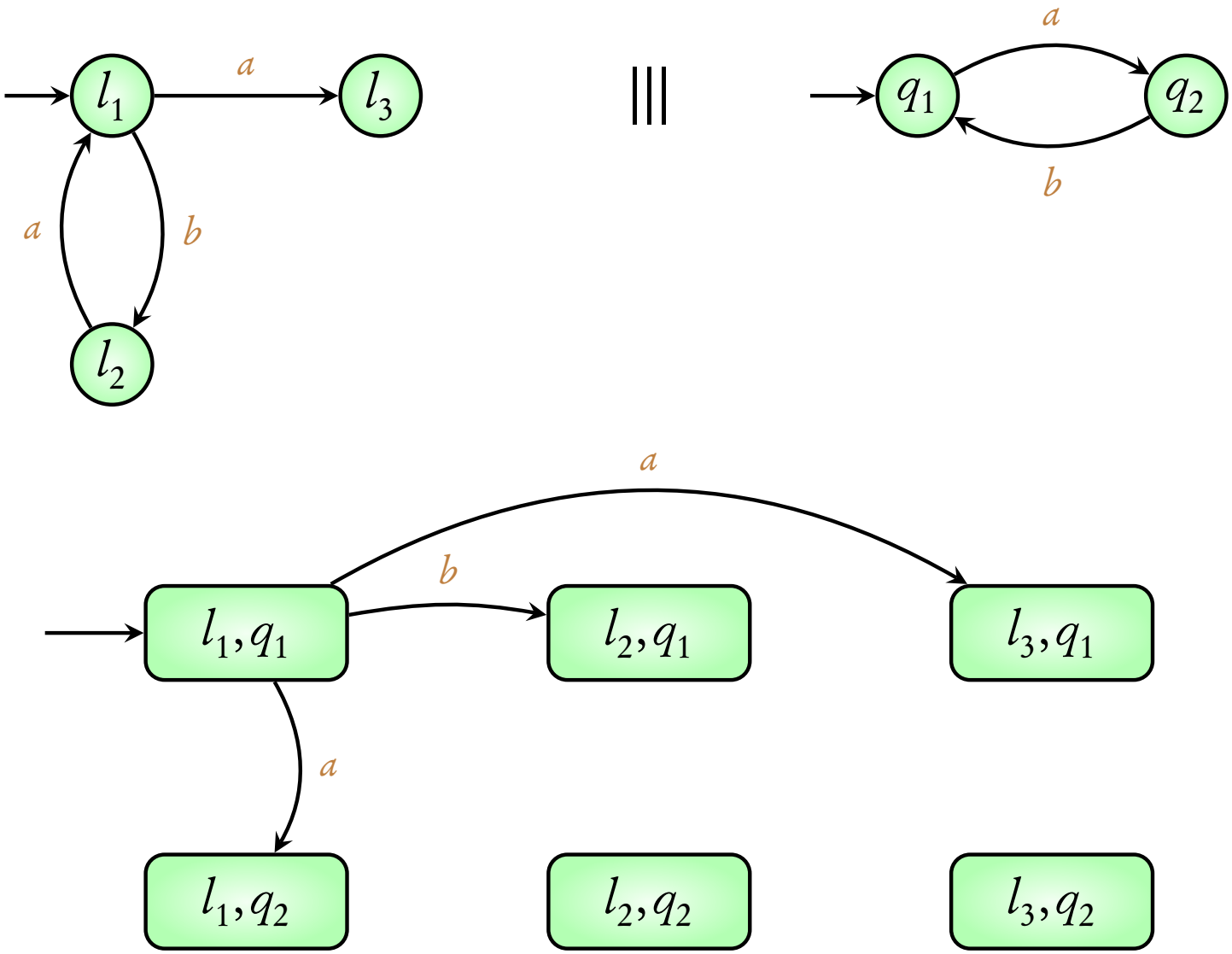
(*initially* y=7)

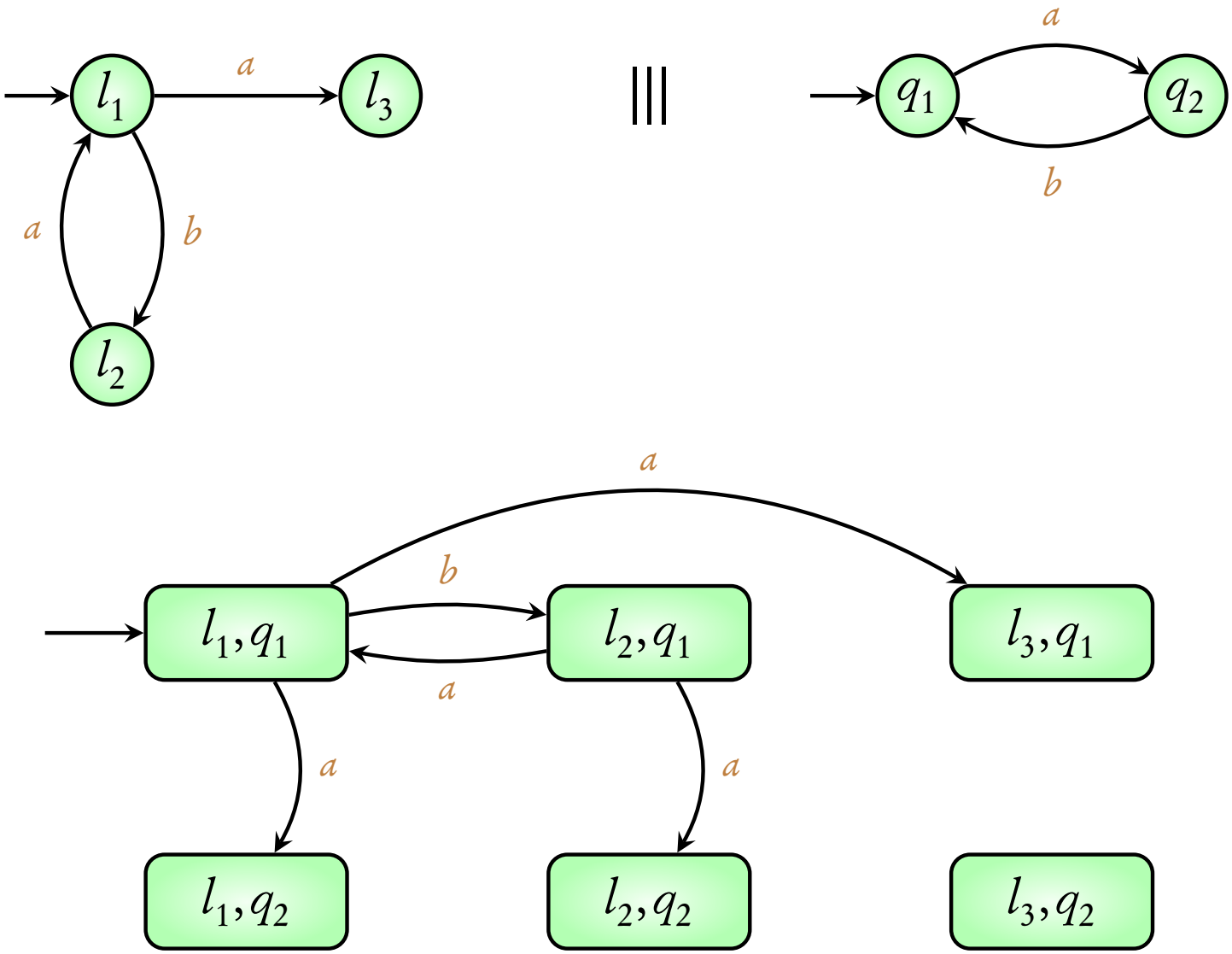

l_1
l_2
l_3
a
a
b
|||
q_1
q_2
a
b

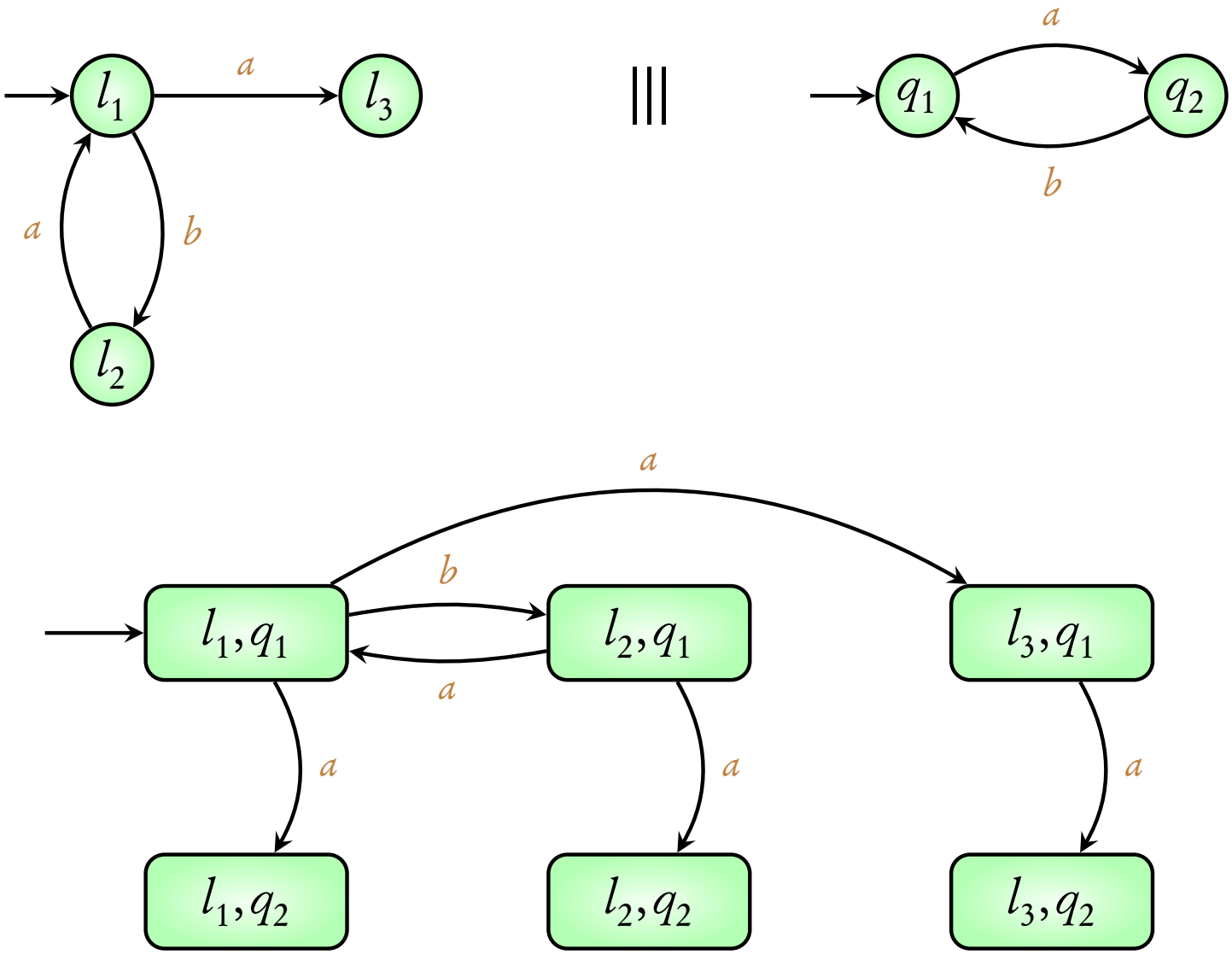

l_1
l_3
l_2
a
a
b
$\|\|$
q_1
q_2
a
b

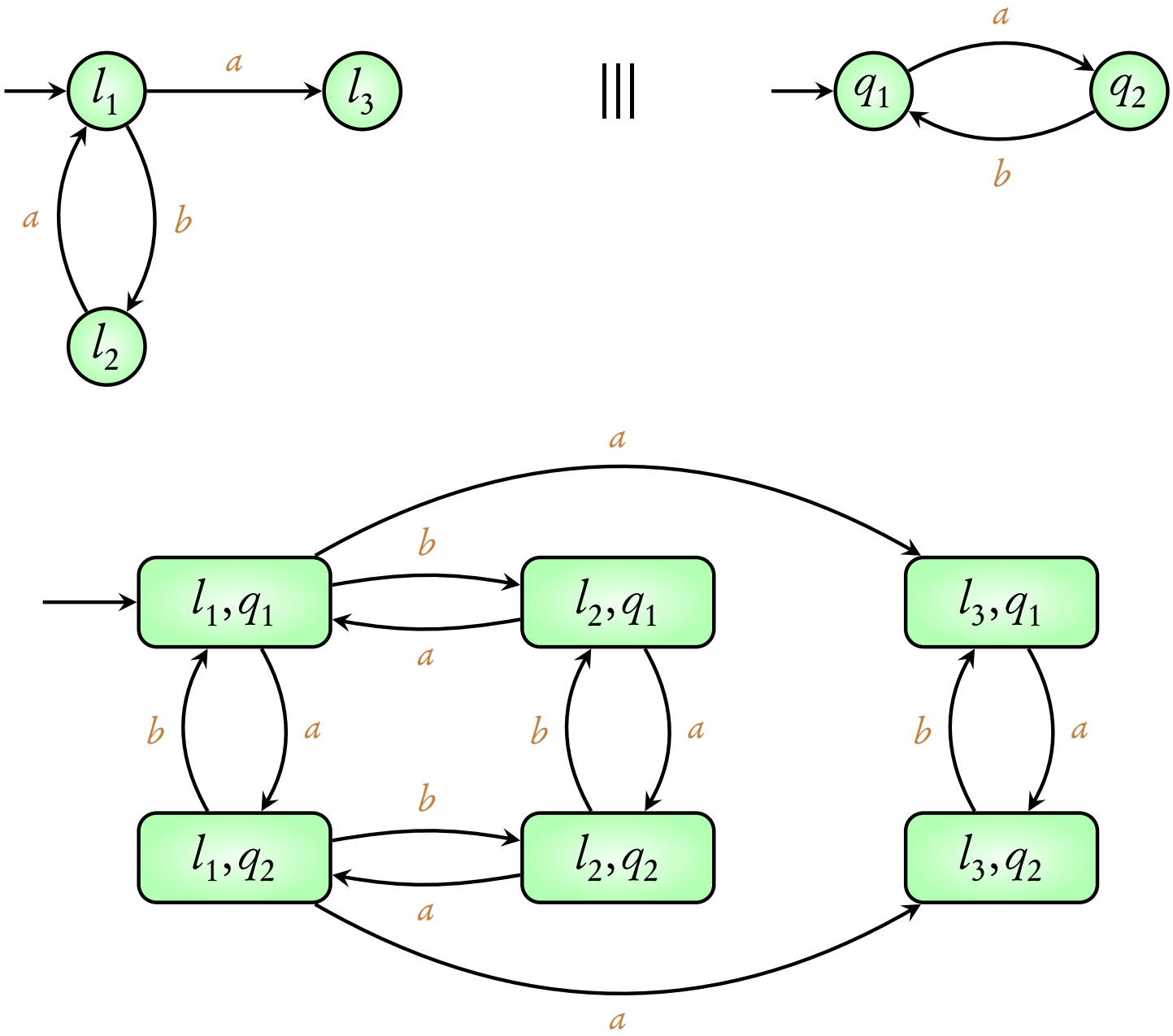

l_1, q_1
l_2, q_1
l_3, q_1
l_1, q_2
l_2, q_2
l_3, q_2


l_1
l_3
l_2
a
a
b
$\|\|$
q_1
q_2
a
b
a
l_1, q_1
l_2, q_1
l_3, q_1
a
l_1, q_2
l_2, q_2
l_3, q_2


l_1
l_3
l_2
a
a
b
|||
q_1
q_2
a
b
a
b
l_1, q_1
l_2, q_1
l_3, q_1
a
l_1, q_2
l_2, q_2
l_3, q_2


l_1
l_3
l_2
a
a
b
|||
q_1
q_2
a
b
a
b
a
l_1, q_1
l_2, q_1
l_3, q_1
a
a
l_1, q_2
l_2, q_2
l_3, q_2


l_1
l_3
l_2
a
a
b
|||
q_1
q_2
a
b
l_1, q_1
l_2, q_1
l_3, q_1
l_1, q_2
l_2, q_2
l_3, q_2
a
b
a
a
a
a


l_1
l_2
l_3
a
a
b
|||
q_1
q_2
a
b
l_1, q_1
l_2, q_1
l_3, q_1
l_1, q_2
l_2, q_2
l_3, q_2
a
b
a
b
a
b
a
b
a
b
a

Multiple systems

$$\mathrm{TS}_1 \;|||\; \mathrm{TS}_2 \;|||\; \ldots \;|||\; \mathrm{TS}_n$$

Multiple systems

$$TS_1 \;|||\; TS_2 \;|||\; \ldots \;|||\; TS_n$$

Exercise: Try out an example of interleaving **three** systems

Concurrent systems

Independent

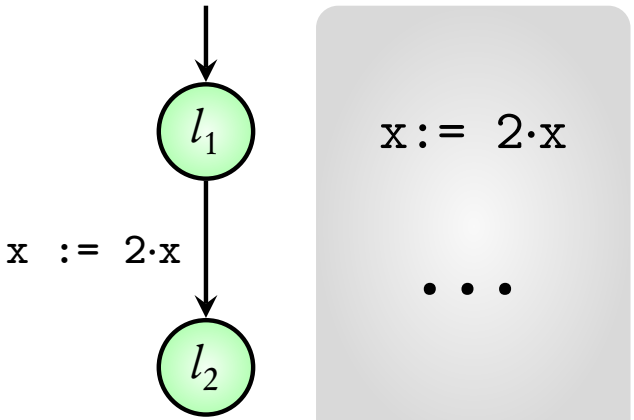
Interleaving

$TS_1 \;|||\; TS_2 \;|||\; \ldots \;|||\; TS_n$

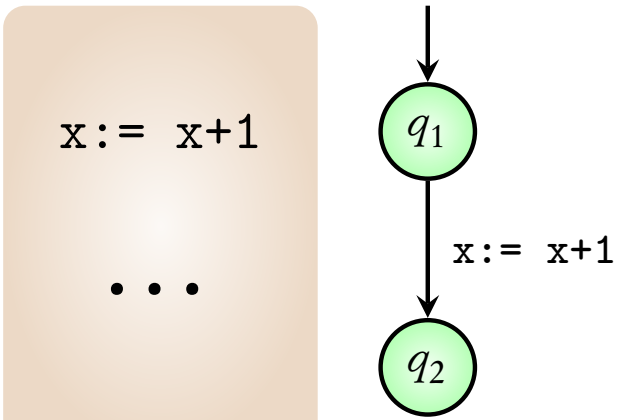
Modeling Asynchronous Concurrency,
No communocation

Shared variables

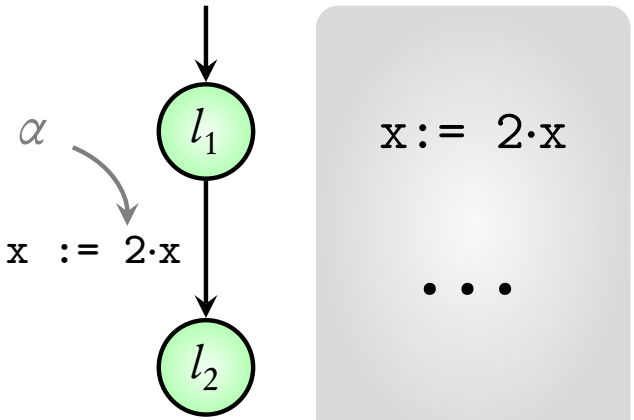
Shared actions



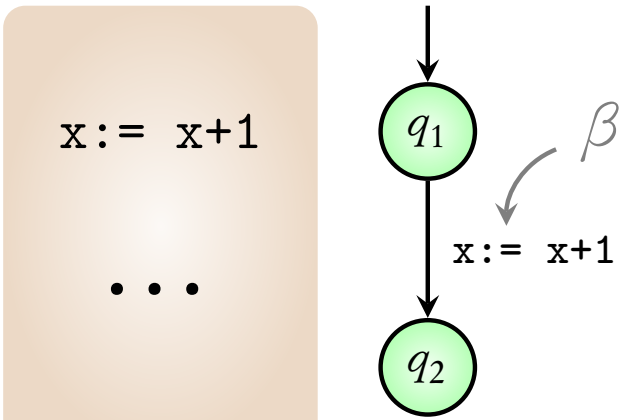
PG_1



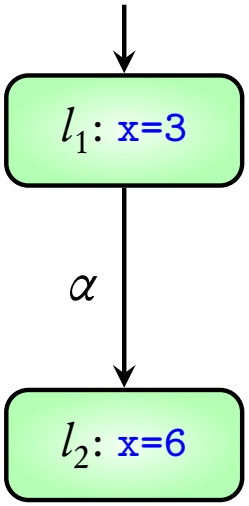
PG_2



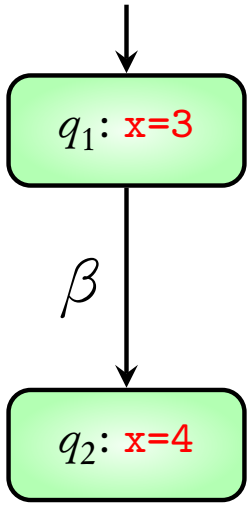
PG_1



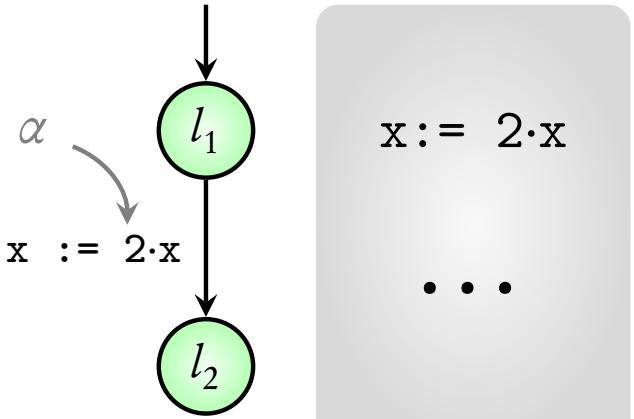
PG_2



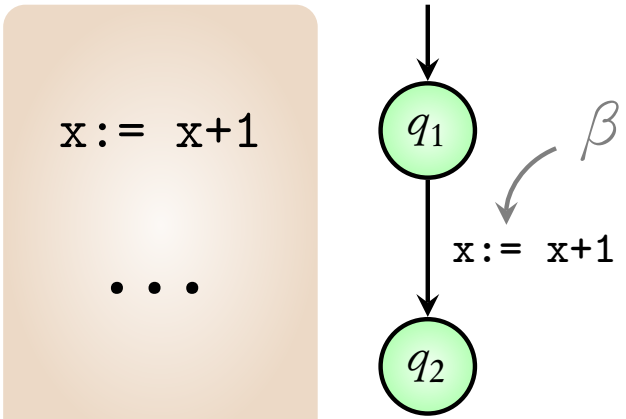
TS_1
(*initially* x=3)



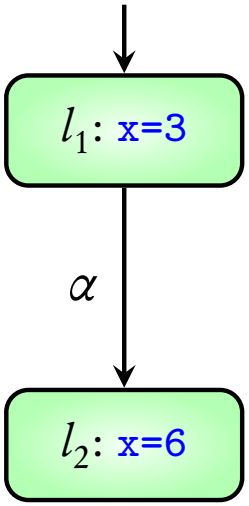
TS_2
(*initially* x=3)



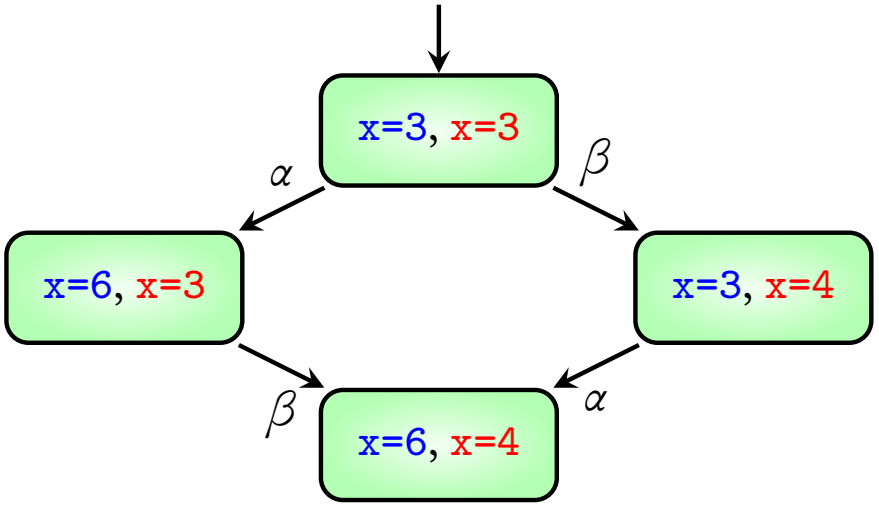
PG_1



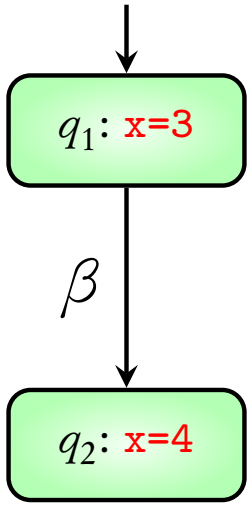
PG_2



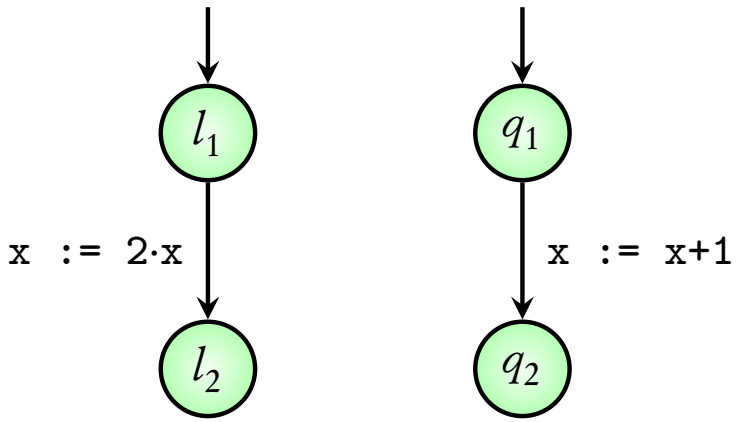
TS_1
(*initially* x=3)

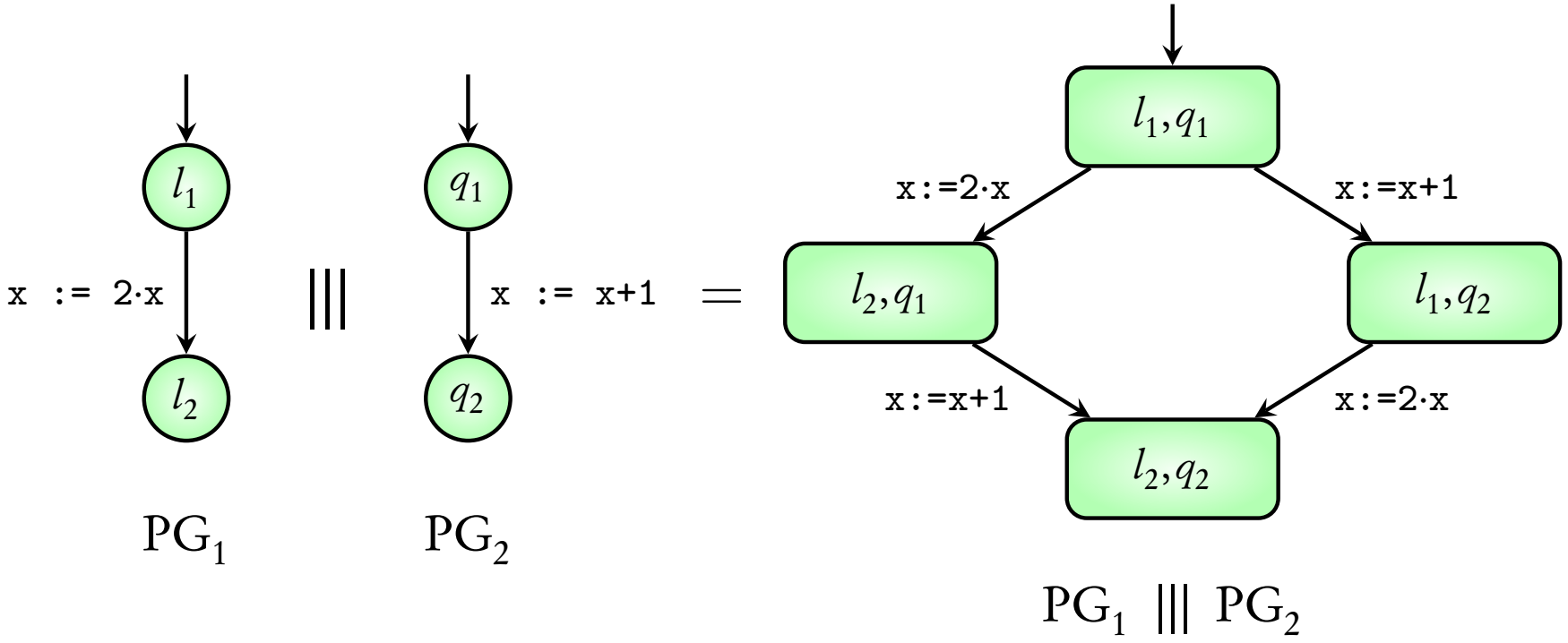


$TS_1 \;|||\; TS_2$



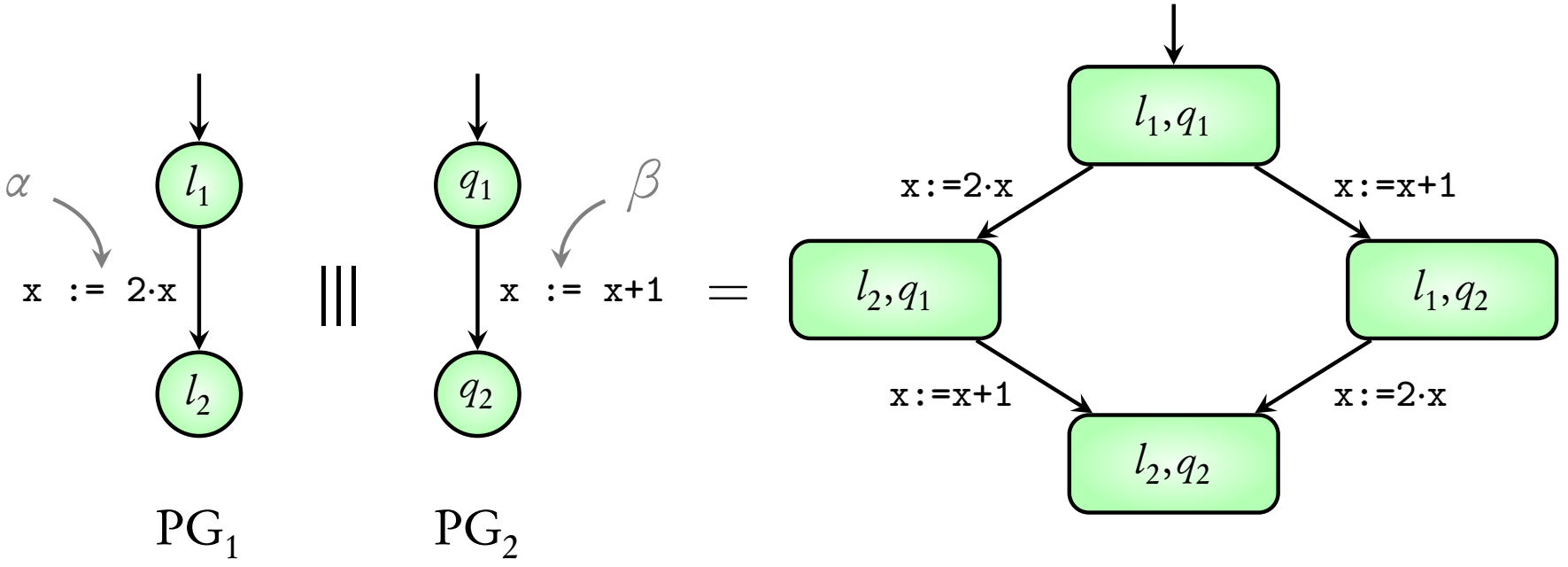
TS_2
(*initially* x=3)


l_1
x := 2·x
l_2
q_1
x := x+1
q_2
PG_1
PG_2



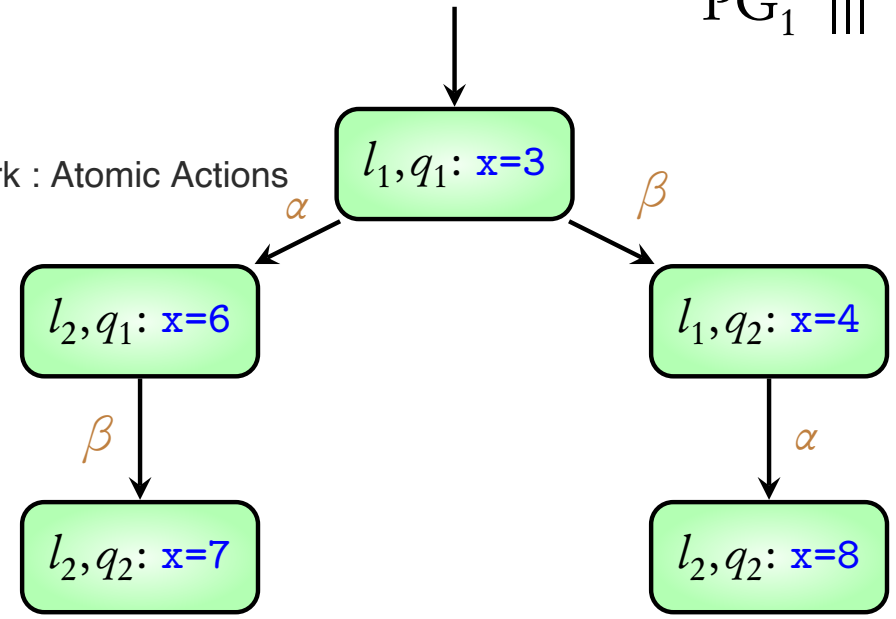
PG_1 PG_2

$\mathrm{PG}_1 \;|||\; \mathrm{PG}_2$



Remark : Atomic Actions

TS(PG1 ||| PG2):



Concurrent systems

Independent

Interleaving

$TS_1 \;|||\; TS_2 \;|||\; \ldots \;|||\; TS_n$

Shared variables

$TS(\, PG_1 \;|||\; PG_2 \;|||\; \ldots \;|||\; PG_n \,)$

Shared actions

```
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
```

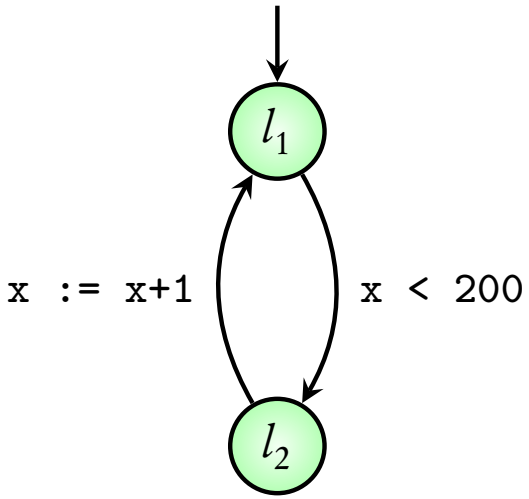
```
while x=200
    x := 0
```

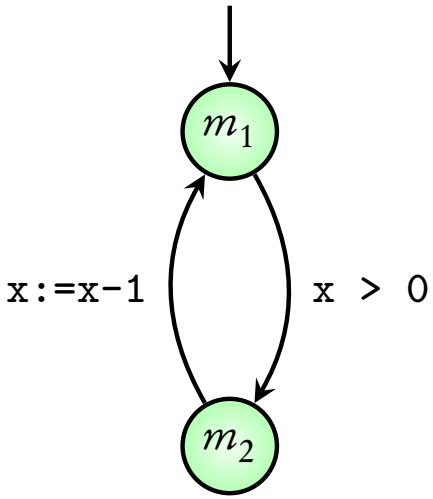
Is the value of x **always between** 0 and 200?

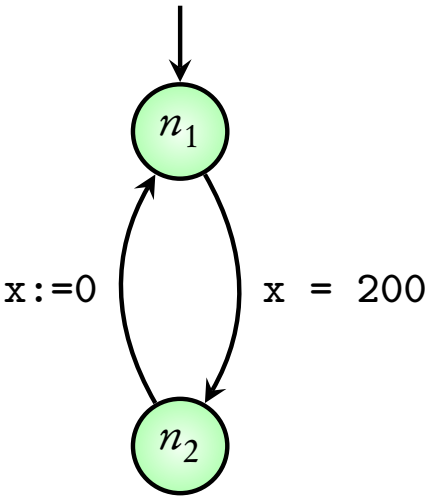
```
while x < 200
    x := x+1
```

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

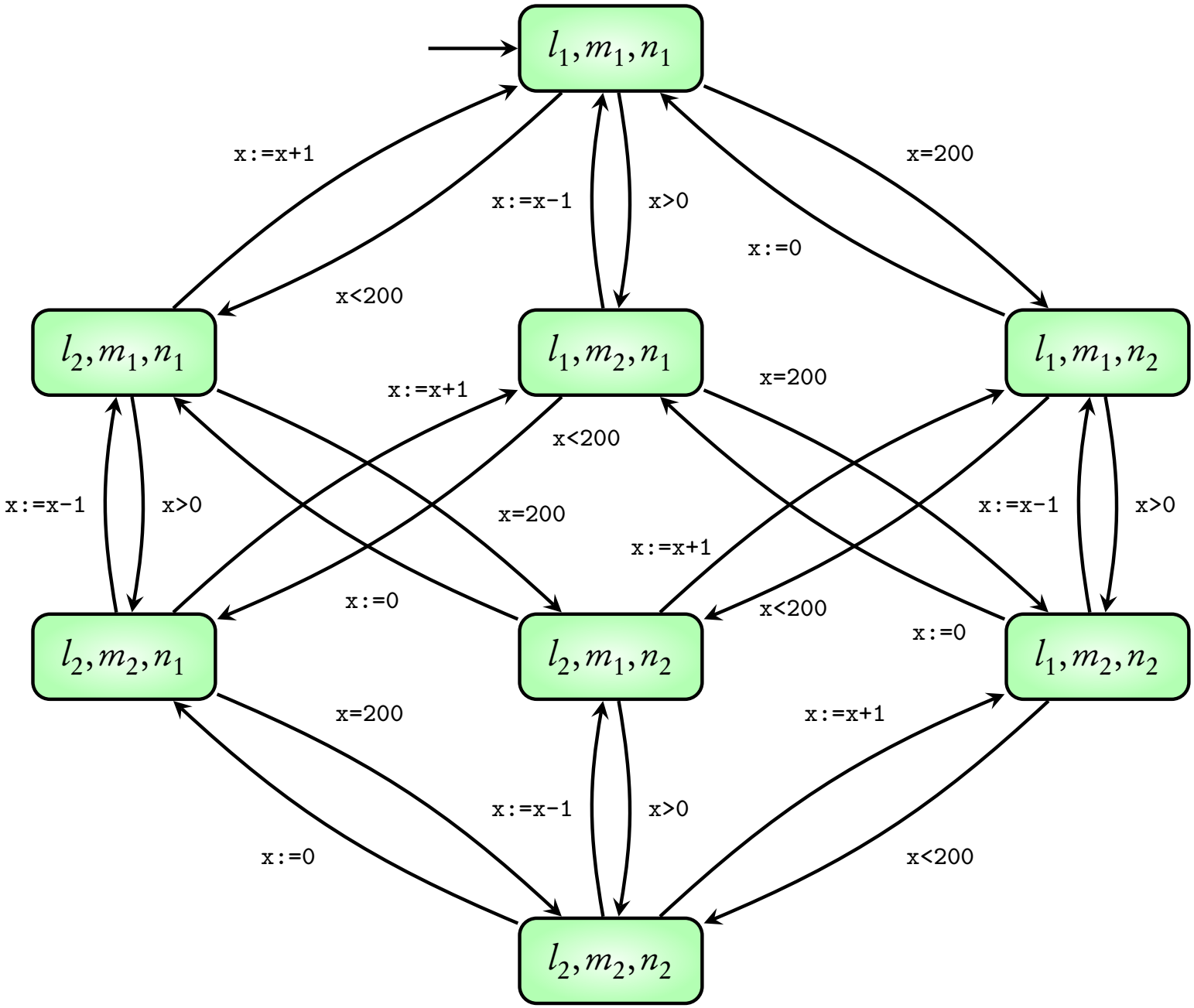
```
while x=200
    x := 0
```

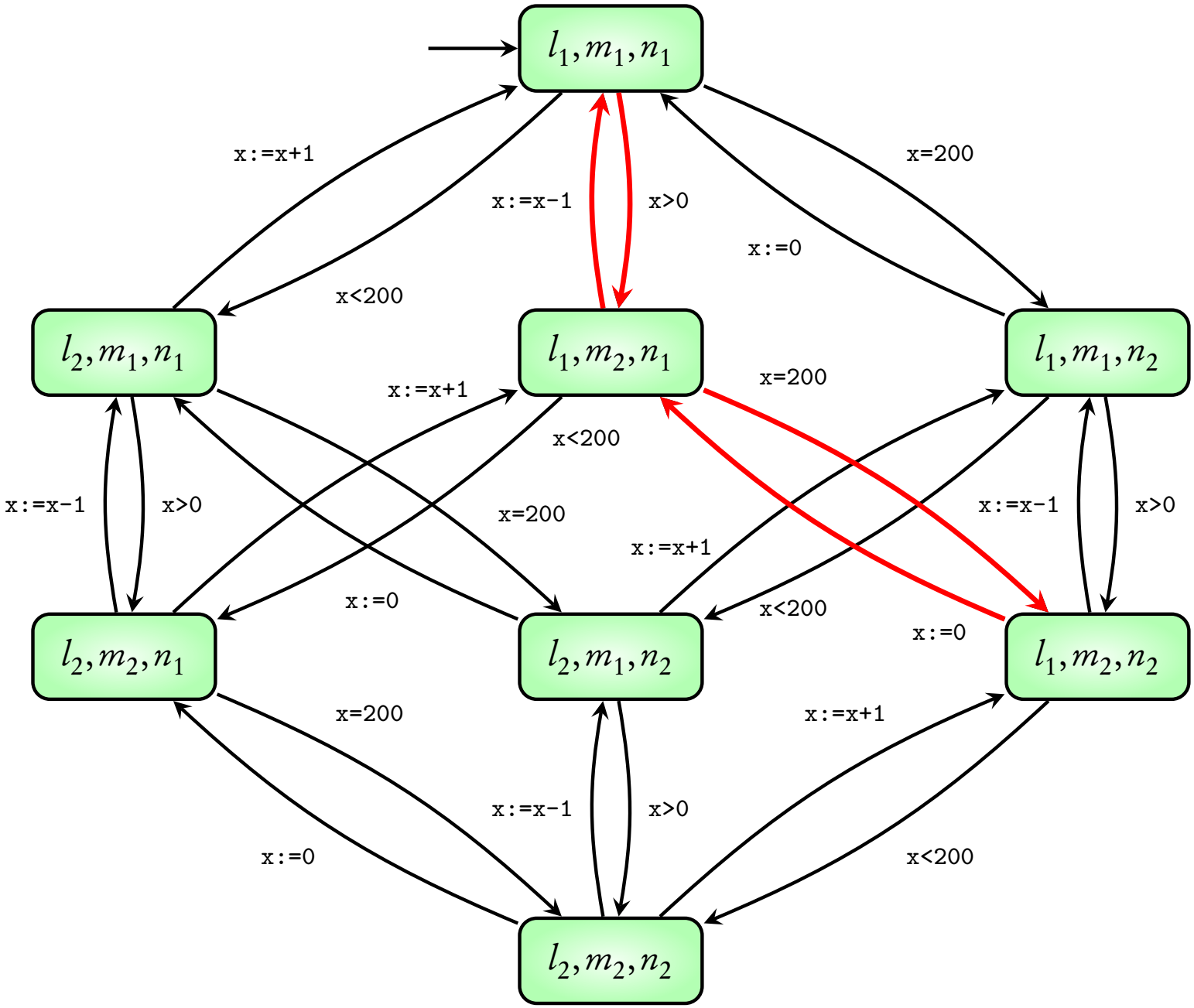


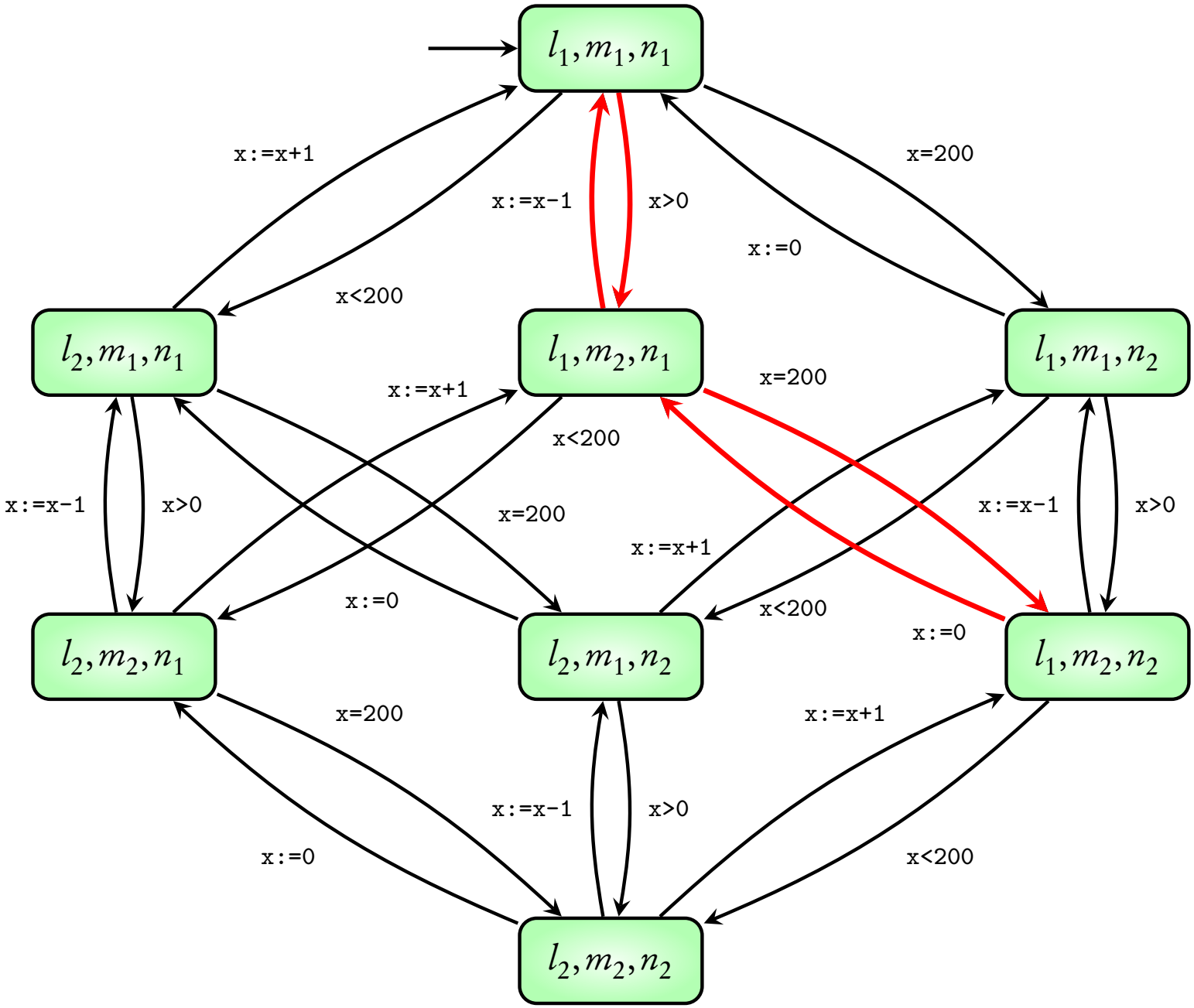




Is the value of x **always between** 0 and 200?

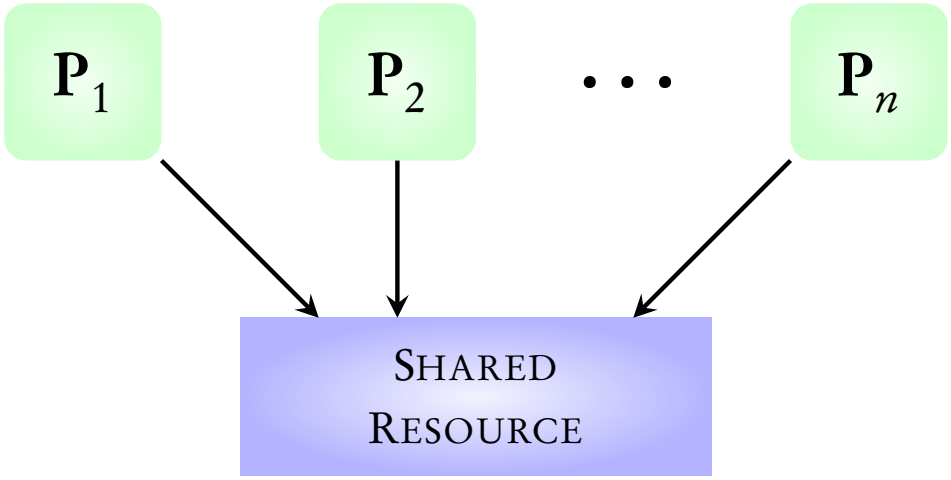

l_1, m_1, n_1
x:=x+1
x:=x-1
x>0
x=200
x:=0
x<200
l_2, m_1, n_1
l_1, m_2, n_1
l_1, m_1, n_2
x:=x+1
x<200
x=200
x:=x-1
x>0
x=200
x:=x+1
x:=0
x<200
x:=0
l_2, m_2, n_1
l_2, m_1, n_2
l_1, m_2, n_2
x=200
x:=x+1
x:=x-1
x>0
x:=0
x<200
l_2, m_2, n_2


l_1, m_1, n_1
l_2, m_1, n_1
l_1, m_2, n_1
l_1, m_1, n_2
l_2, m_2, n_1
l_2, m_1, n_2
l_1, m_2, n_2
l_2, m_2, n_2
x:=x+1
x:=x-1
x>0
x<200
x=200
x:=0



Is the value of x **always between** 0 and 200? **No**

Coming next: Mutual exclusion



(variable, printer, ...)

Mutual Exclusion: **No two processes** can access the resource **simultaneously**

Goal: Modeling the **protocols** used for mutual exclusion

P_1

loop forever

⋮ *non-critical actions*

request

critical section

release

⋮ *non-critical actions*

end loop

P_2

loop forever

⋮ *non-critical actions*

request

critical section

release

⋮ *non-critical actions*

end loop

P_1

loop forever

⋮ *non-critical actions*

request

critical section

release

⋮ *non-critical actions*

end loop

P_2

loop forever

⋮ *non-critical actions*

request

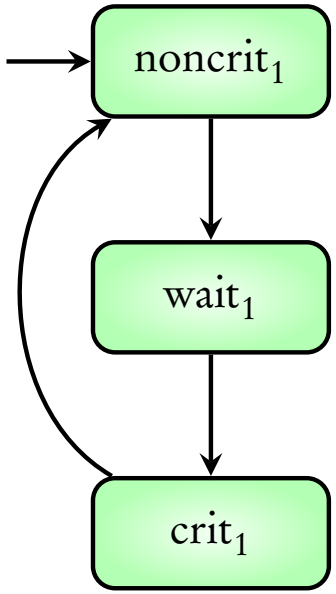
critical section

release

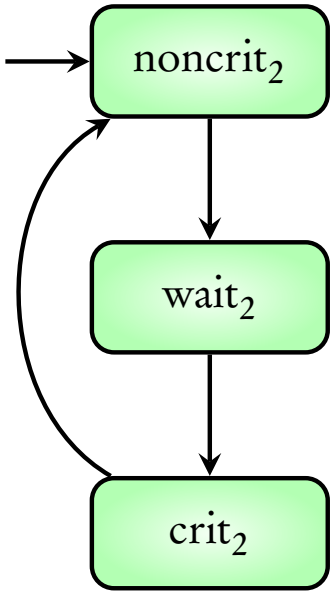
⋮ *non-critical actions*

end loop

PG_1



PG_2



P_1

loop forever

⋮ *non-critical actions*

⟨ if y>0: y:=y−1 ⟩ *request*

critical section

y:=y+1 *release*

⋮ *non-critical actions*

end loop

P_2

loop forever

⋮ *non-critical actions*

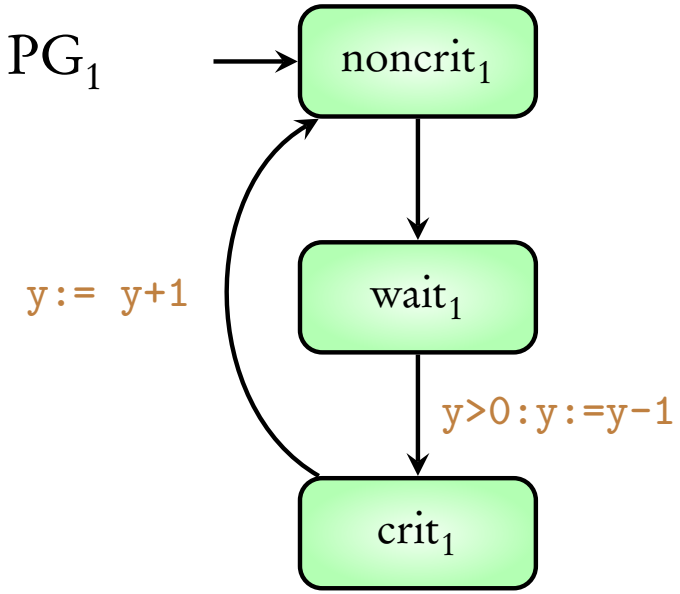
⟨ if y>0: y:=y−1 ⟩ *request*

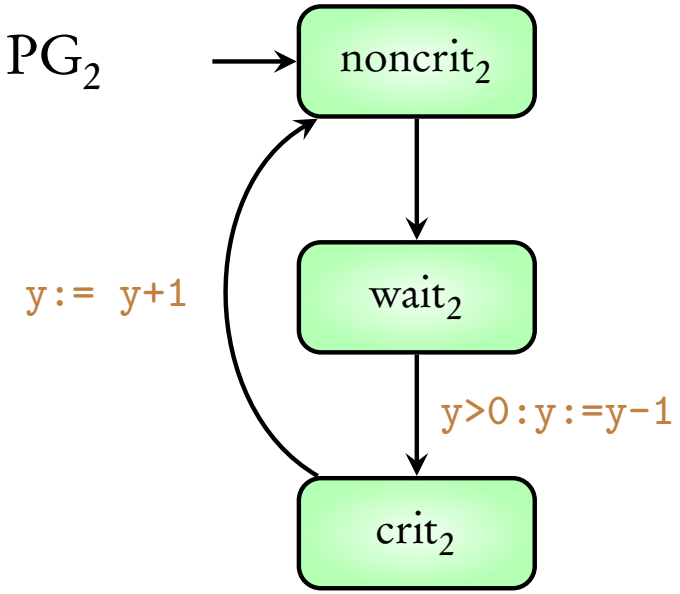
critical section

y:=y+1 *release*

⋮ *non-critical actions*

end loop





P_1

loop forever

⋮ *non-critical actions*

⟨ if y>0: y:=y−1 ⟩ *request*

critical section

y:=y+1 *release*

⋮ *non-critical actions*

end loop

P_2

loop forever

⋮ *non-critical actions*

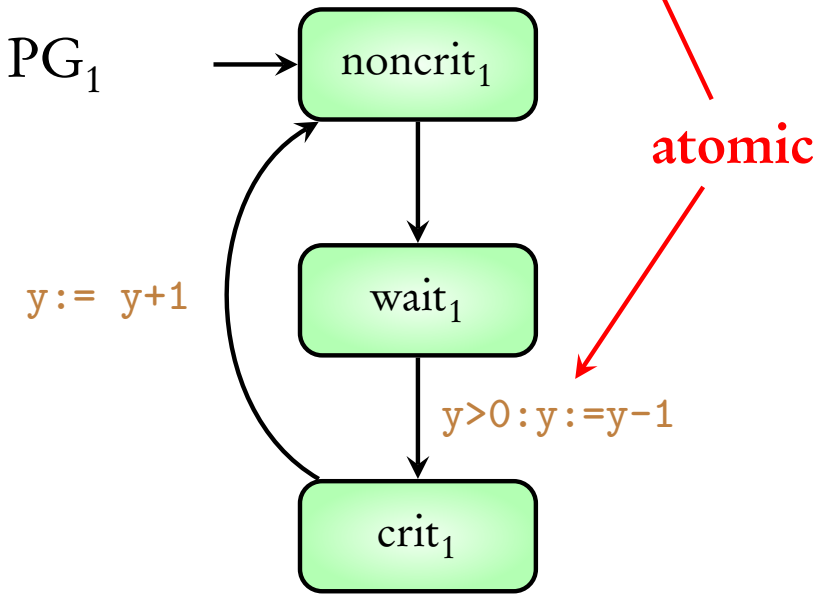
⟨ if y>0: y:=y−1 ⟩ *request*

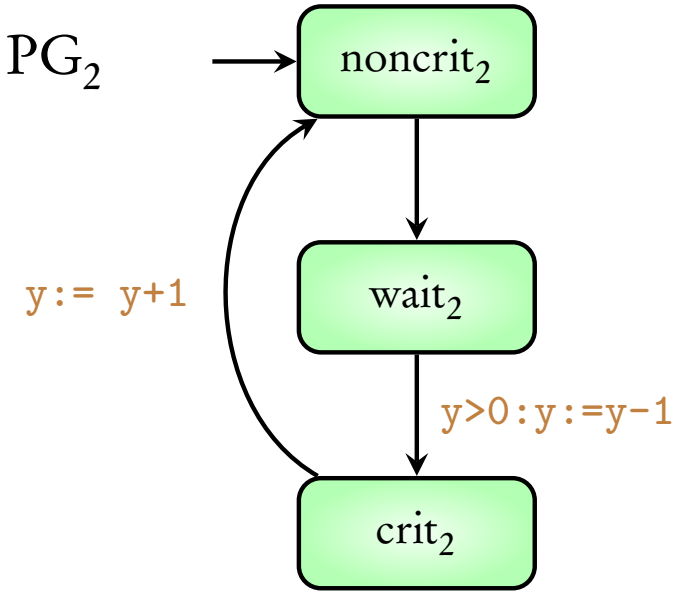
critical section

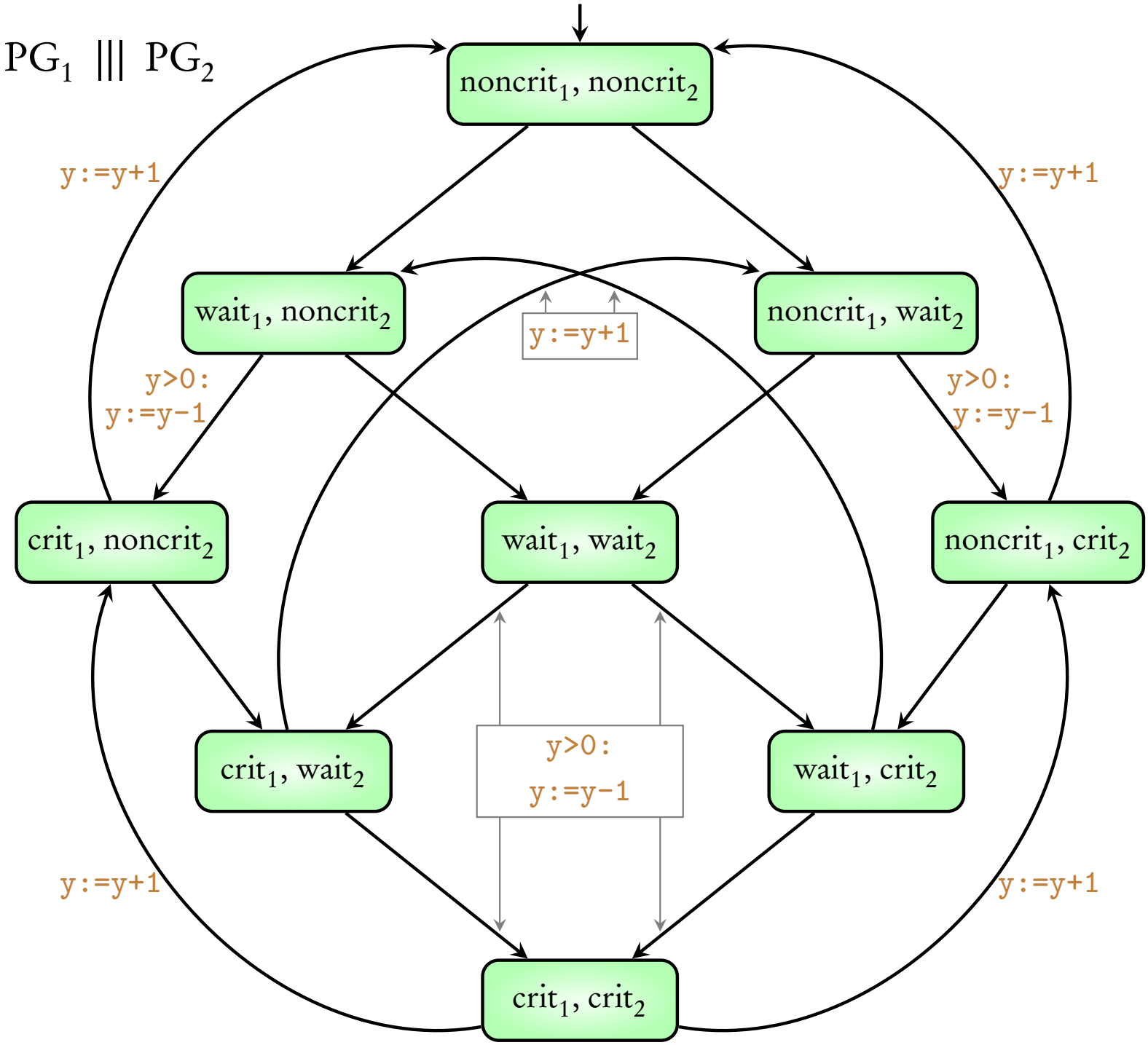
y:=y+1 *release*

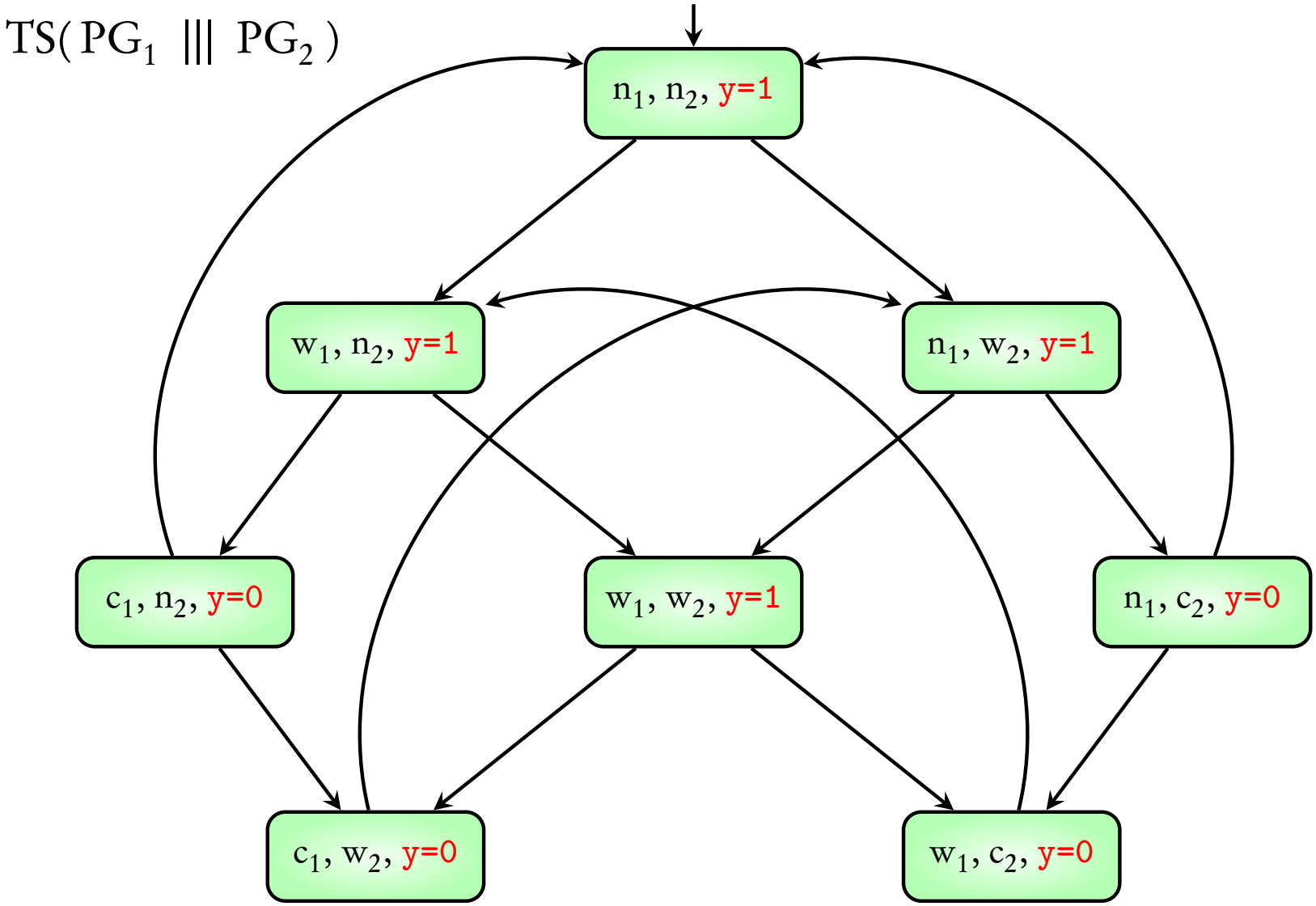
⋮ *non-critical actions*

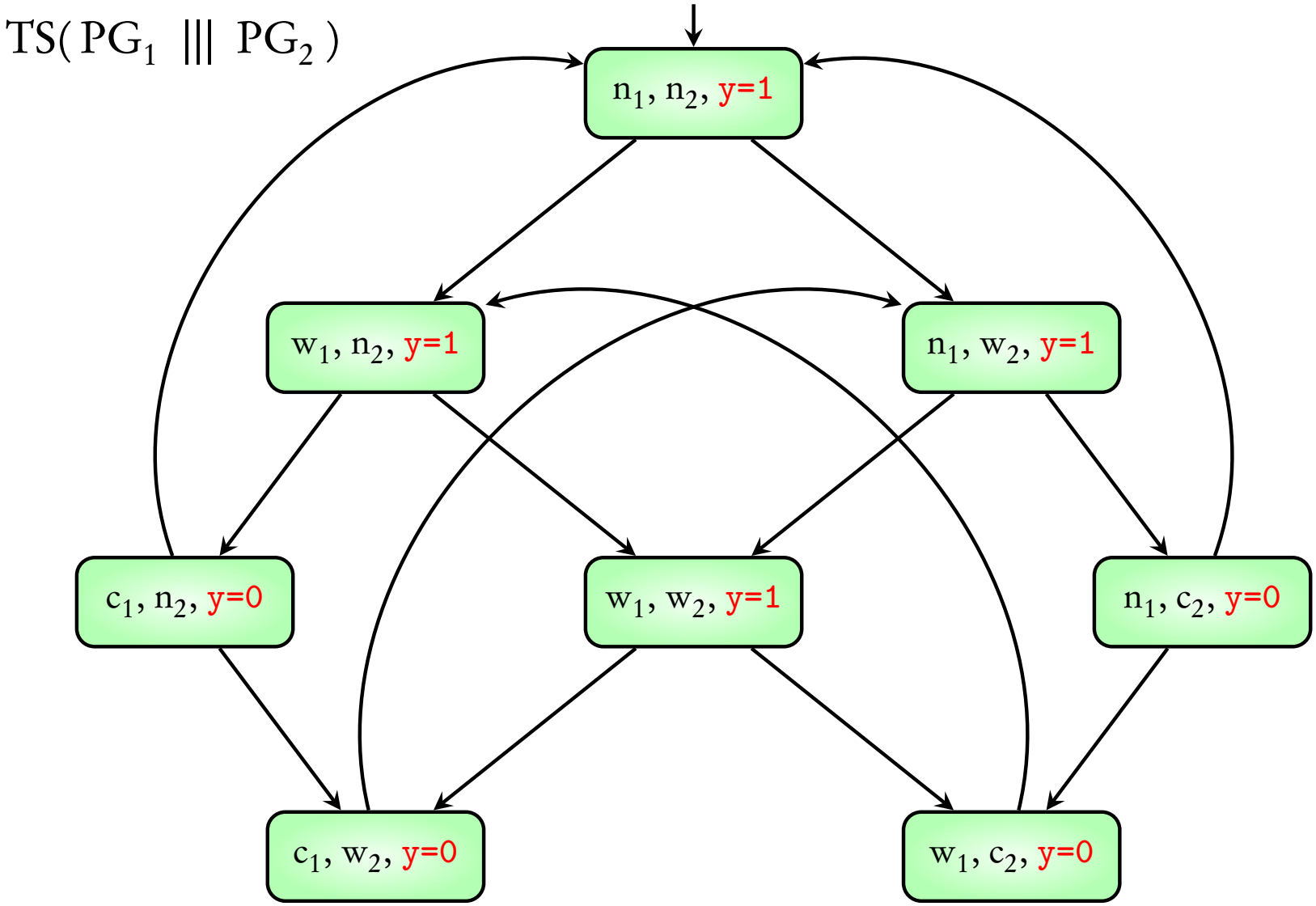
end loop






PG1 ||| PG2
noncrit1, noncrit2
y:=y+1
y:=y+1
wait1, noncrit2
noncrit1, wait2
y:=y+1
y>0:
y:=y-1
y>0:
y:=y-1
crit1, noncrit2
wait1, wait2
noncrit1, crit2
crit1, wait2
y>0:
y:=y-1
wait1, crit2
y:=y+1
y:=y+1
crit1, crit2


TS(PG1 ||| PG2)
n1, n2, y=1
w1, n2, y=1
n1, w2, y=1
c1, n2, y=0
w1, w2, y=1
n1, c2, y=0
c1, w2, y=0
w1, c2, y=0



Both processes **cannot be** in critical section **simultaneously**

Concurrent systems

Independent

Interleaving

$TS_1 \;|||\; TS_2 \;|||\; \ldots \;|||\; TS_n$

Shared variables

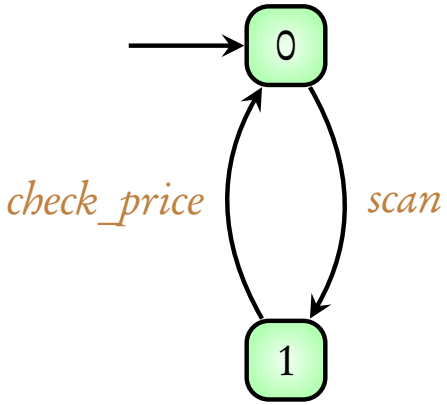
$TS(\, PG_1 \;|||\; PG_2 \;|||\; \ldots \;|||\; PG_n \,)$

Mutual Exclusion

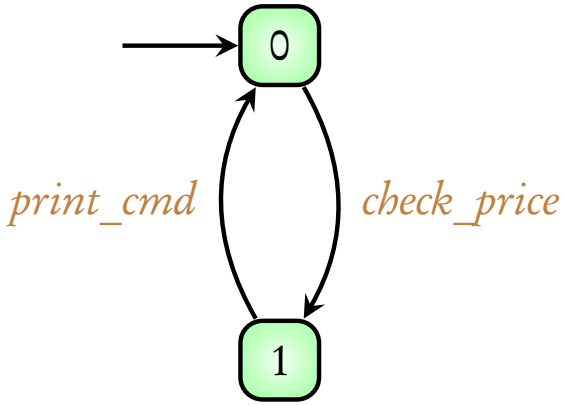
Shared actions

Book-keeping system in a supermarket

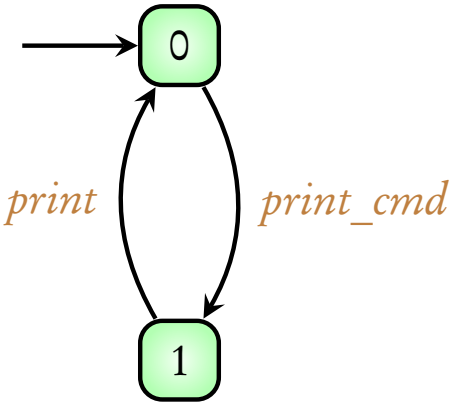
Bar-Code Reader (BCR)



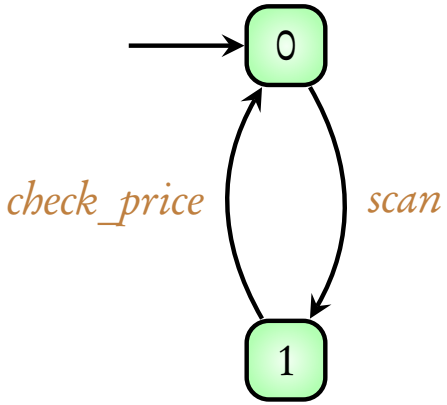
Booking Program (BP)



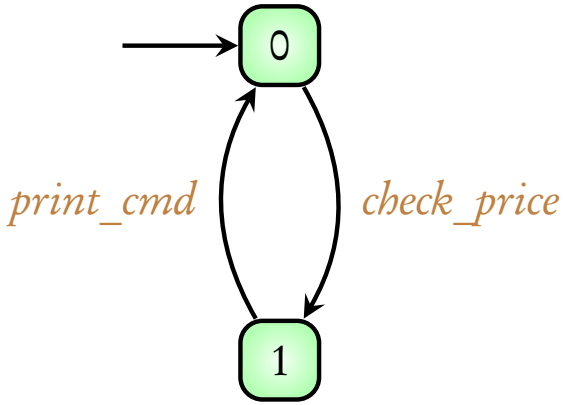
Printer (P)



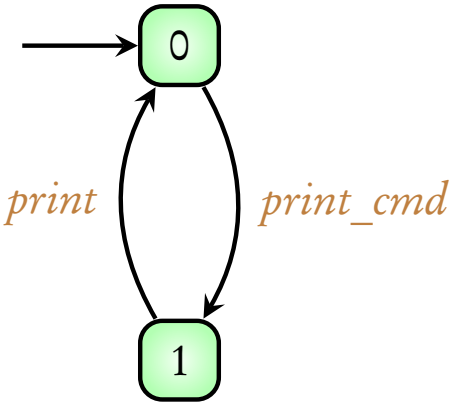
Bar-Code Reader (BCR)

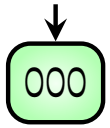


Booking Program (BP)

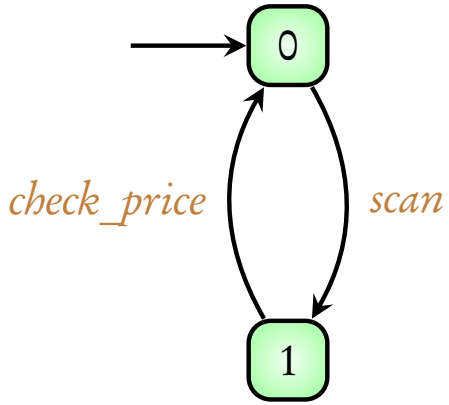


Printer (P)

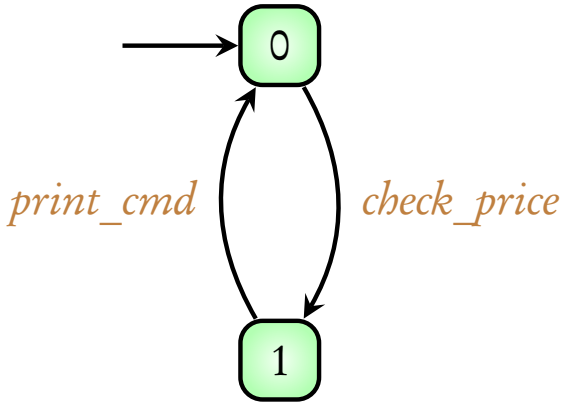




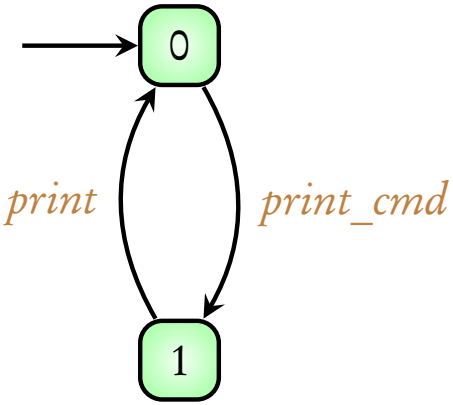
Bar-Code Reader (BCR)



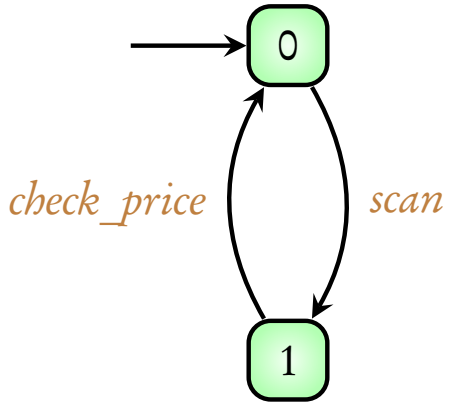
Booking Program (BP)



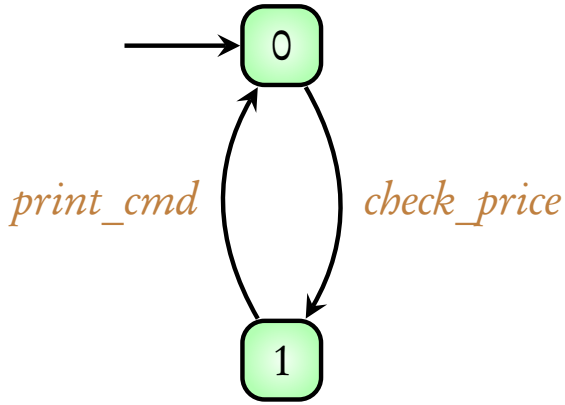
Printer (P)



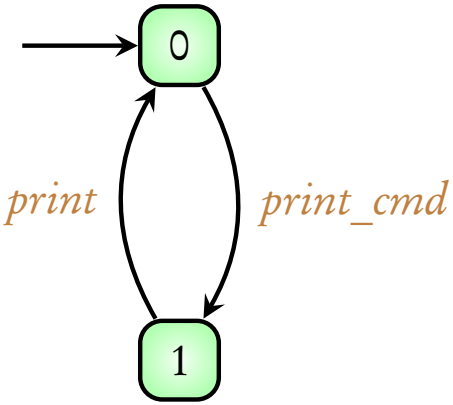
Bar-Code Reader (BCR)

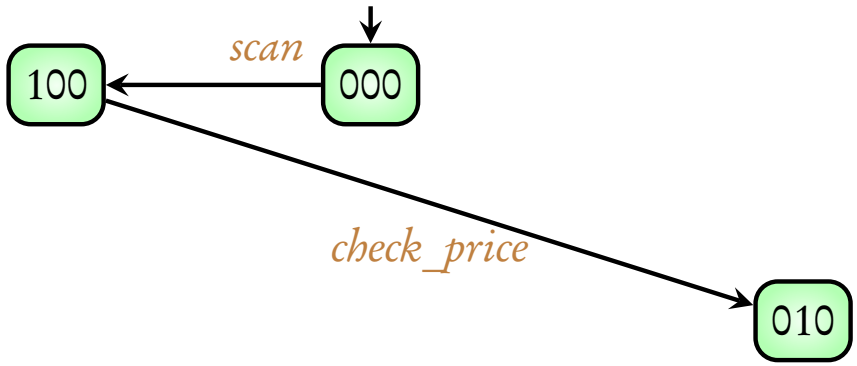


Booking Program (BP)

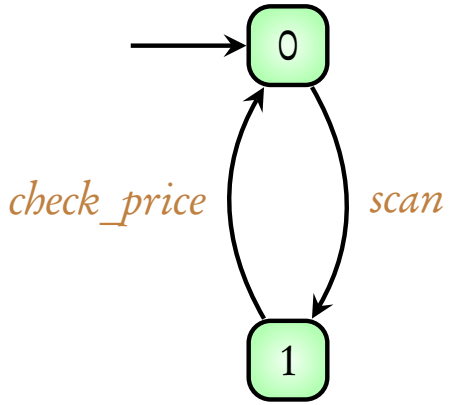


Printer (P)

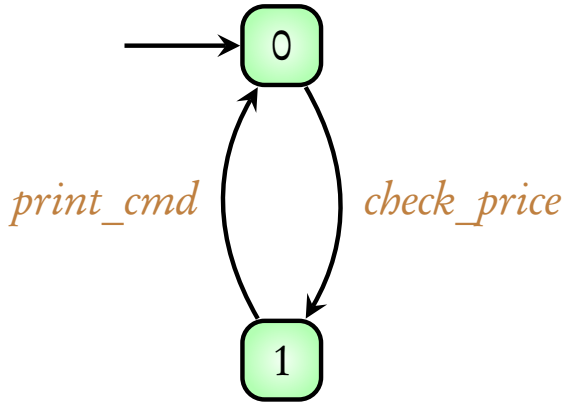




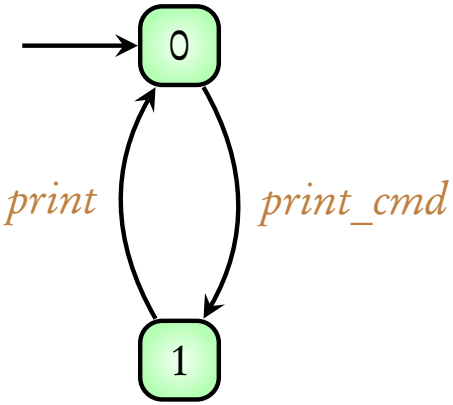
Bar-Code Reader (BCR)

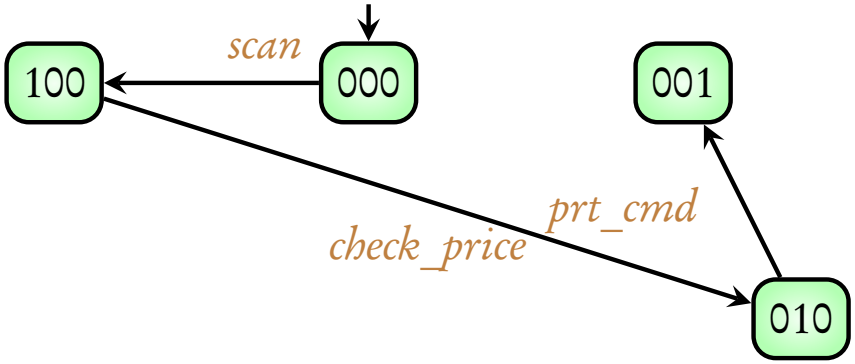


Booking Program (BP)

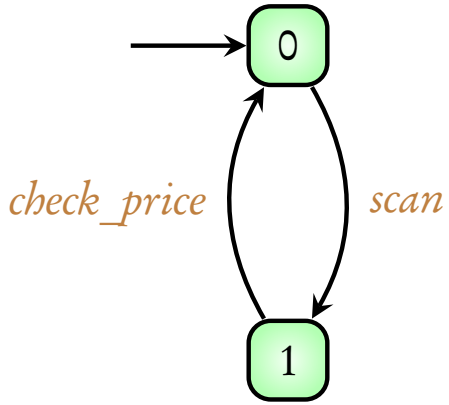


Printer (P)

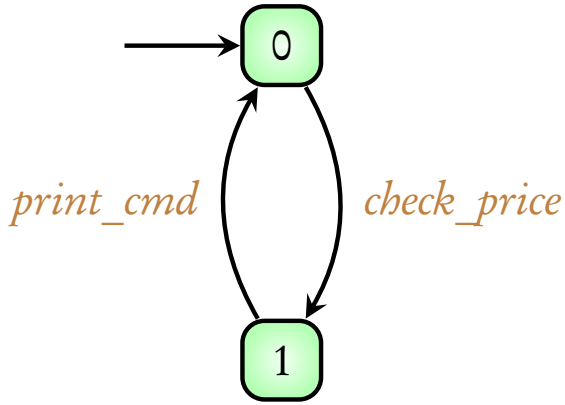




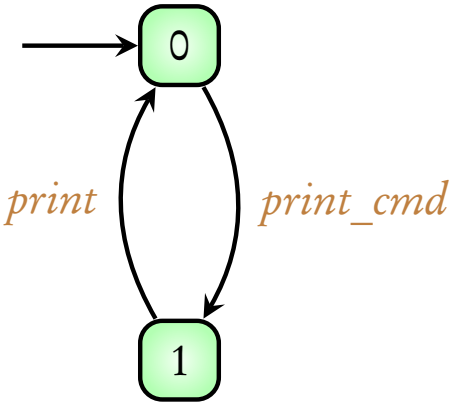
Bar-Code Reader (BCR)

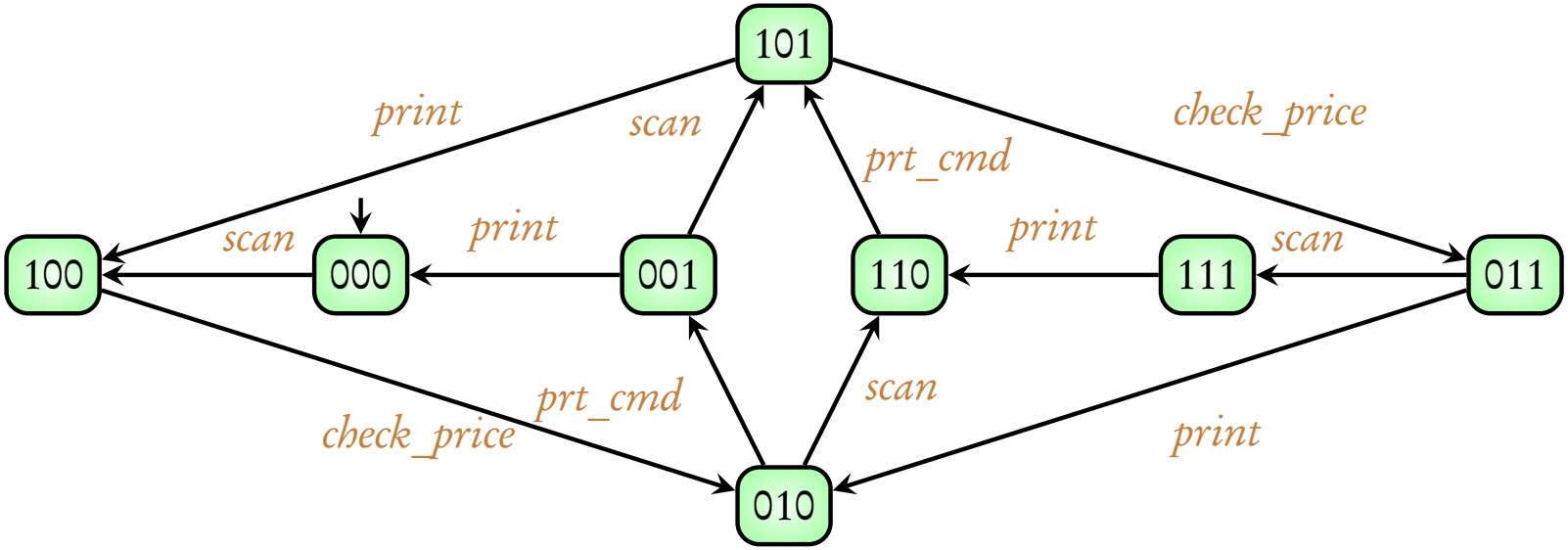


Booking Program (BP)

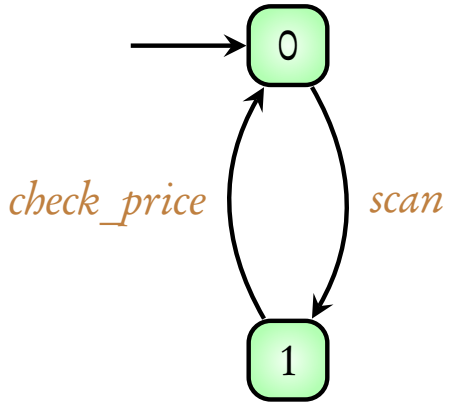


Printer (P)

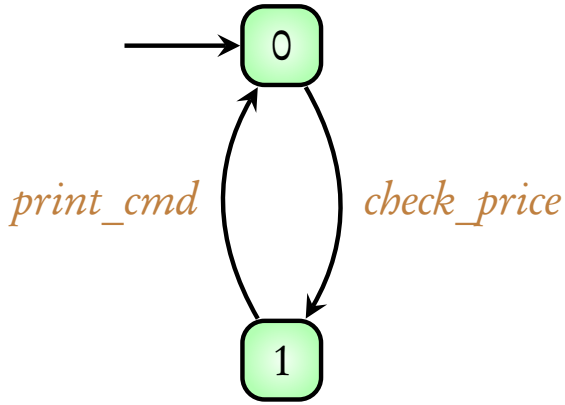




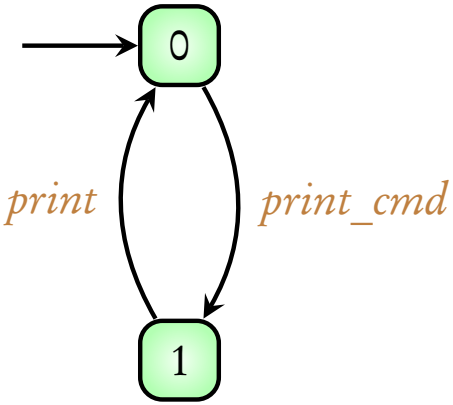
Bar-Code Reader (BCR)



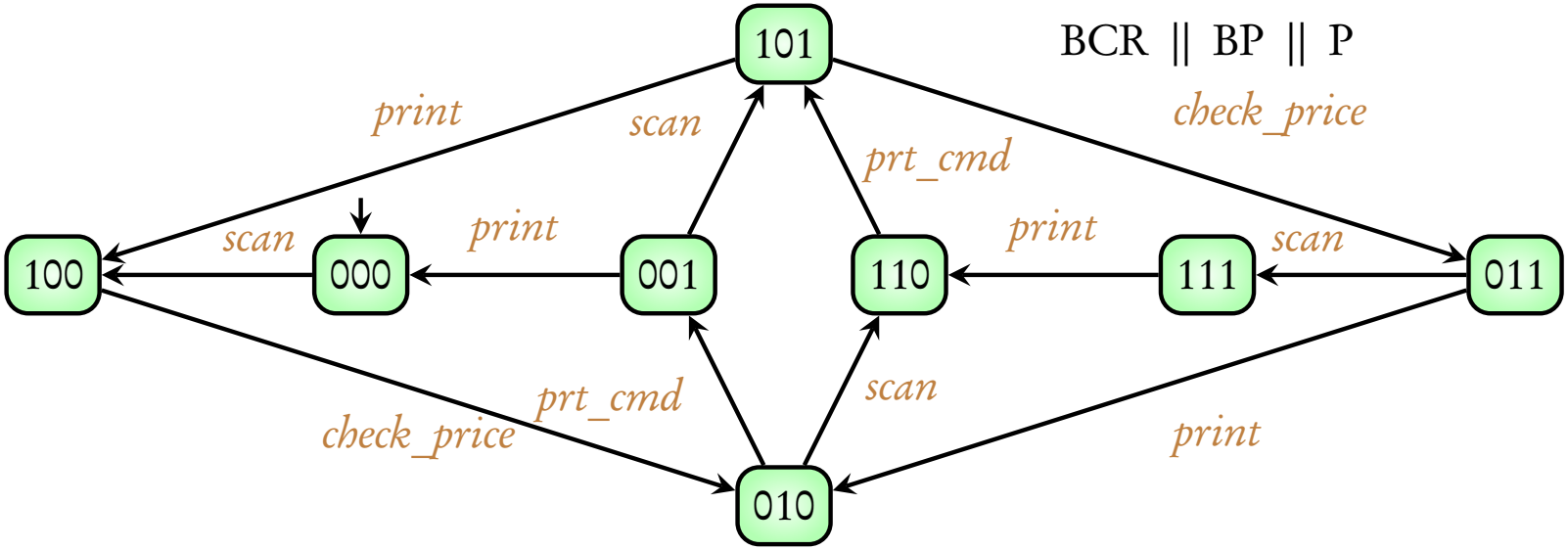
Booking Program (BP)



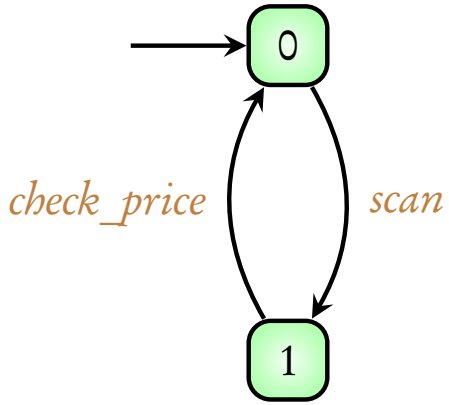
Printer (P)



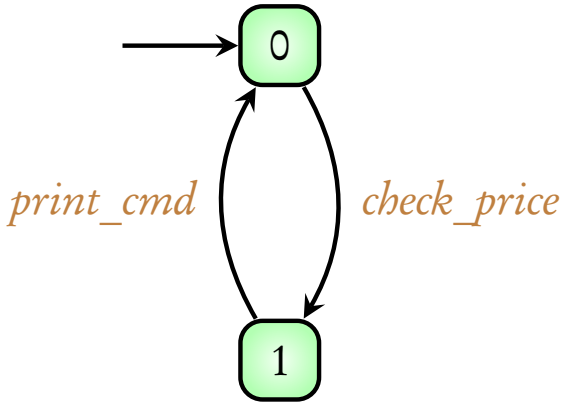
BCR || BP || P



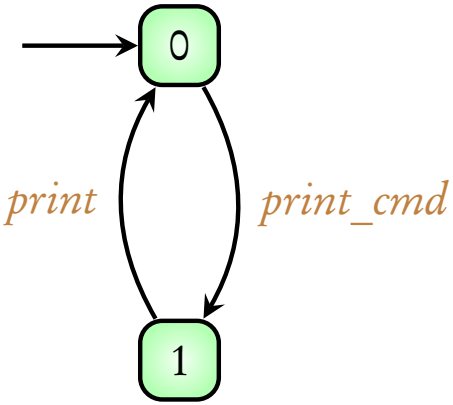
Bar-Code Reader (BCR)



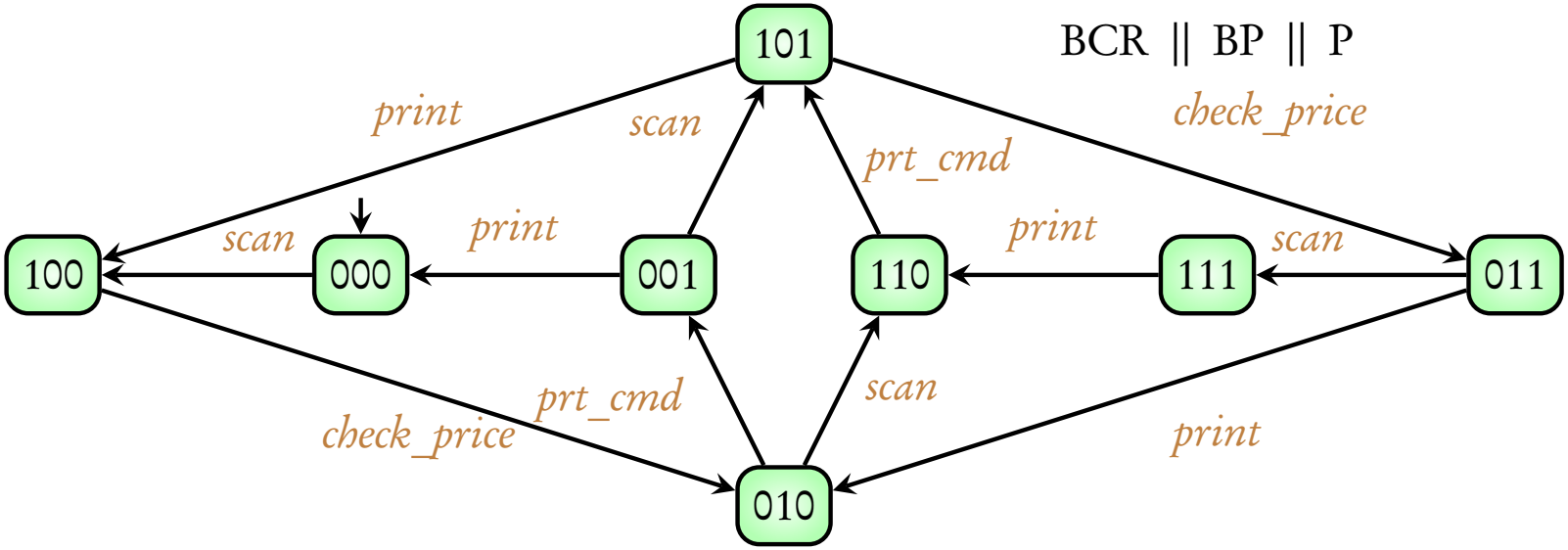
Booking Program (BP)



Printer (P)

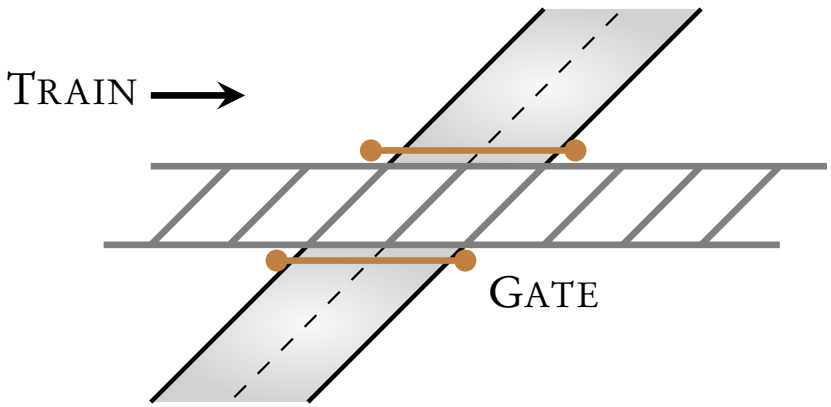


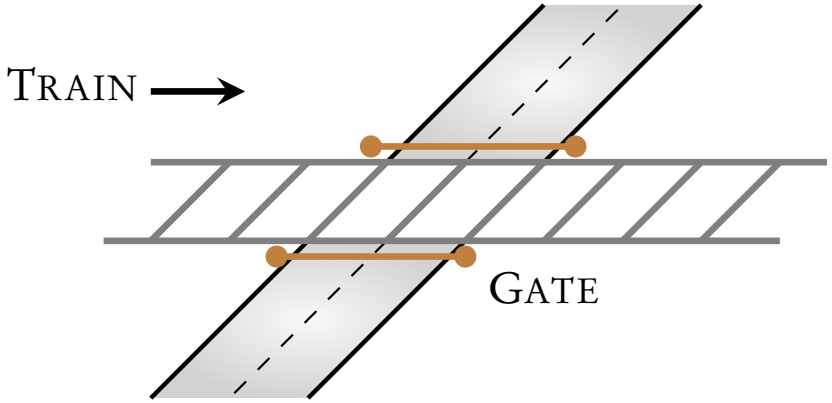
BCR || BP || P

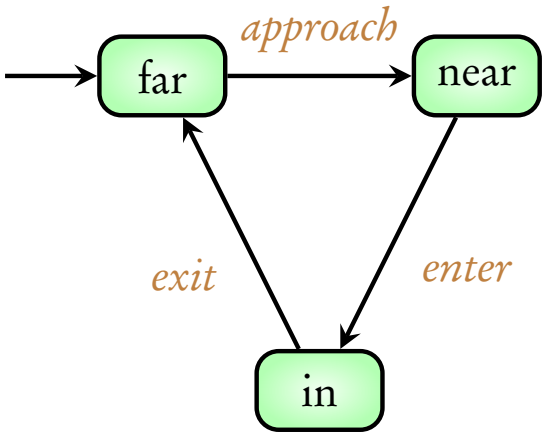


check_price, *print_cmd*: **Shared** actions (*also called* **handshaking** actions)

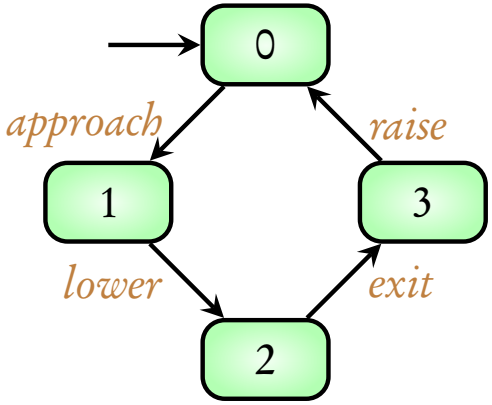
Next example: Train-Gate-Controller


TRAIN
GATE

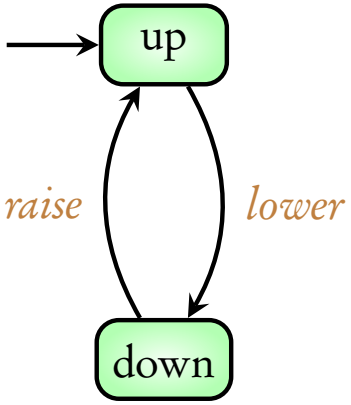




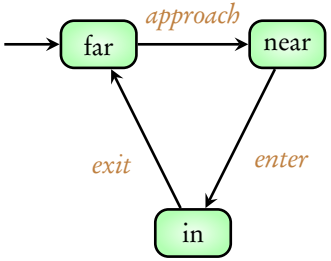
Train



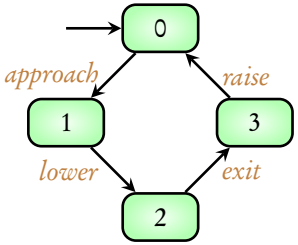
Controller



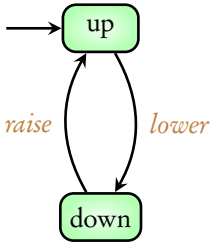
Gate



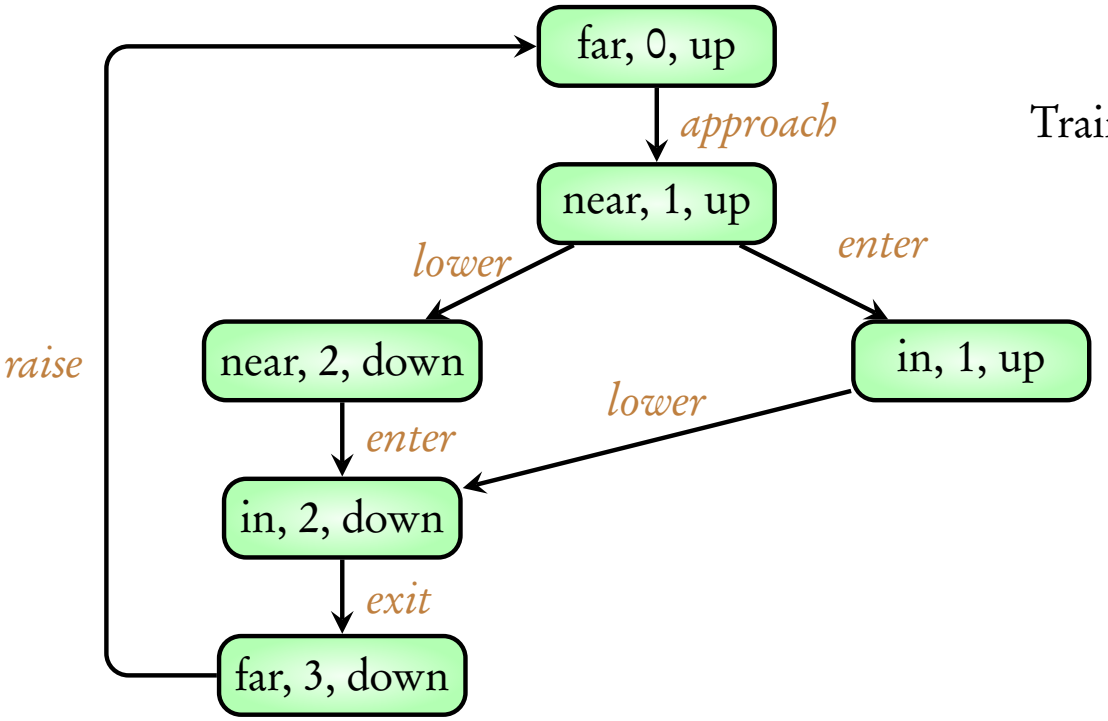
Train



Controller



Gate



Train || Controller || Gate

|| : **Handshake** operator

Independent

Interleaving

$TS_1 \;|||\; TS_2 \;|||\; \ldots \;|||\; TS_n$

Shared variables

$TS(\,PG_1 \;|||\; PG_2 \;|||\; \ldots \;|||\; PG_n\,)$

Mutual Exclusion

Shared actions

$TS_1 \;||\; TS_2$

Reference: Principles of Model Checking, *Baier and Katoen*, MIT Press (2008)
Pages 35 - 53

Summary

- **Module 1:** Transition systems, Modeling simple sequential programs
- **Module 2:** Modeling sequential hardware circuits
- **Module 3:** Modeling data-dependent programs
- **Module 4:** Modeling concurrent systems

Important concepts: Non-determinism, program graphs, interleaving and handshake operators