Verification

Use languages to describe the implementation and the specification of a system.

Reduce the verification problem to language inclusion between implementation and specification

```
while x = 1 do
    if y = 1 then
        x ← 0
    y ← 1 − x
end
```

Configuration

Initial configuration

Execution, full execution, potential execution

```
1  while x = 1 do
2     if y = 1 then
3        x ← 0
4     y ← 1 − x
5  end
```

A configuration of the program is a triple $[\ell, n_x, n_y]$, where $\ell \in \{1, 2, 3, 4, 5\}$ is the current value of the program counter, and $n_x, n_y \in \{0, 1\}$ are the current values of x and y. So the set C of configurations contains in this case $5 \times 2 \times 2 = 20$ elements. The initial configurations are $[1, 0, 0], [1, 0, 1], [1, 1, 0], [1, 1, 1]$, i.e., all configurations in which control is at line 1. The sequence

$$[1, 1, 1]\ [2, 1, 1]\ [3, 1, 1]\ [1, 0, 1]\ [5, 0, 1]$$

is a full execution, while

$$[1, 1, 0]\ [2, 1, 0]\ [4, 1, 0]\ [1, 1, 0]$$

is also an execution, but not a full one.

Implementation: set E of executions

Specification: subset P of the potential executions that satisfy a property

or

subset V of the potential executions that violate a property

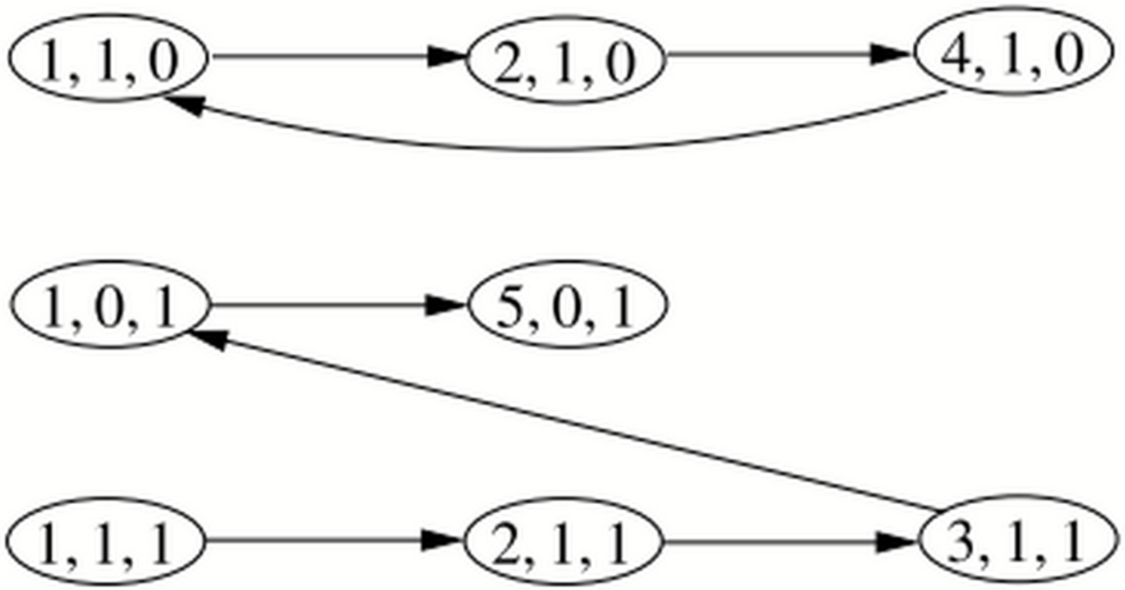
Implementation satisfies specification if :

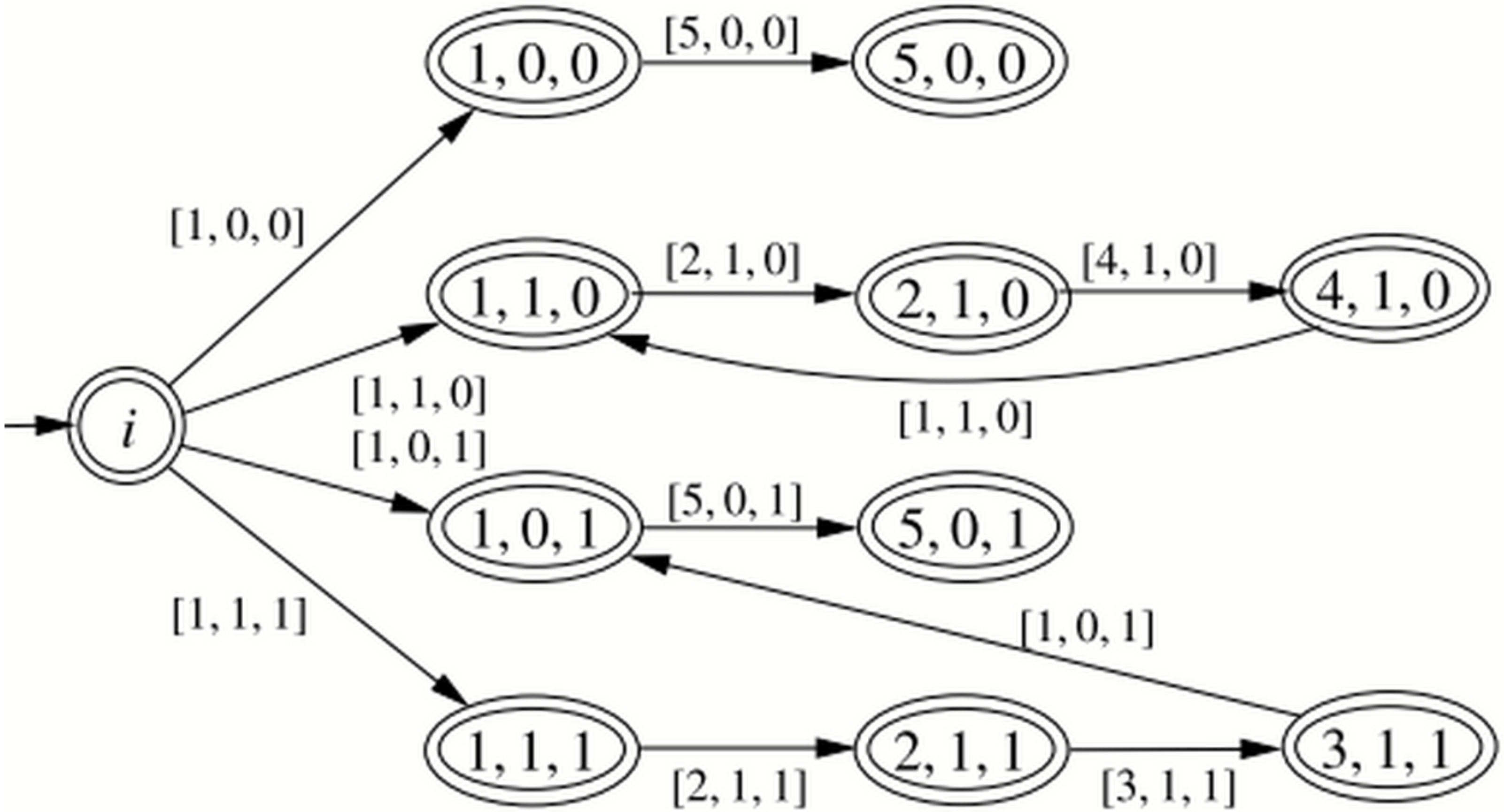
E included in P or intersection of E and V empty

If E and P regular: inclusion checkable with automata

If E and V regular: emptiness checkable with automata

How often does this happen?


1, 0, 0
5, 0, 0
1, 1, 0
2, 1, 0
4, 1, 0
1, 0, 1
5, 0, 1
1, 1, 1
2, 1, 1
3, 1, 1

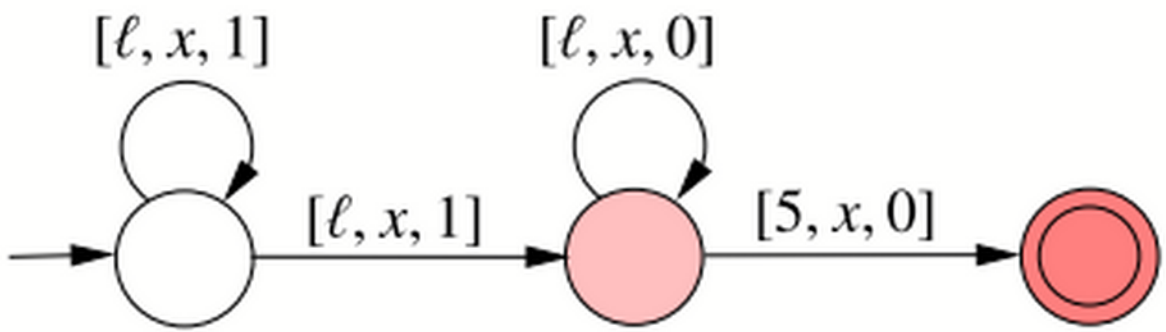


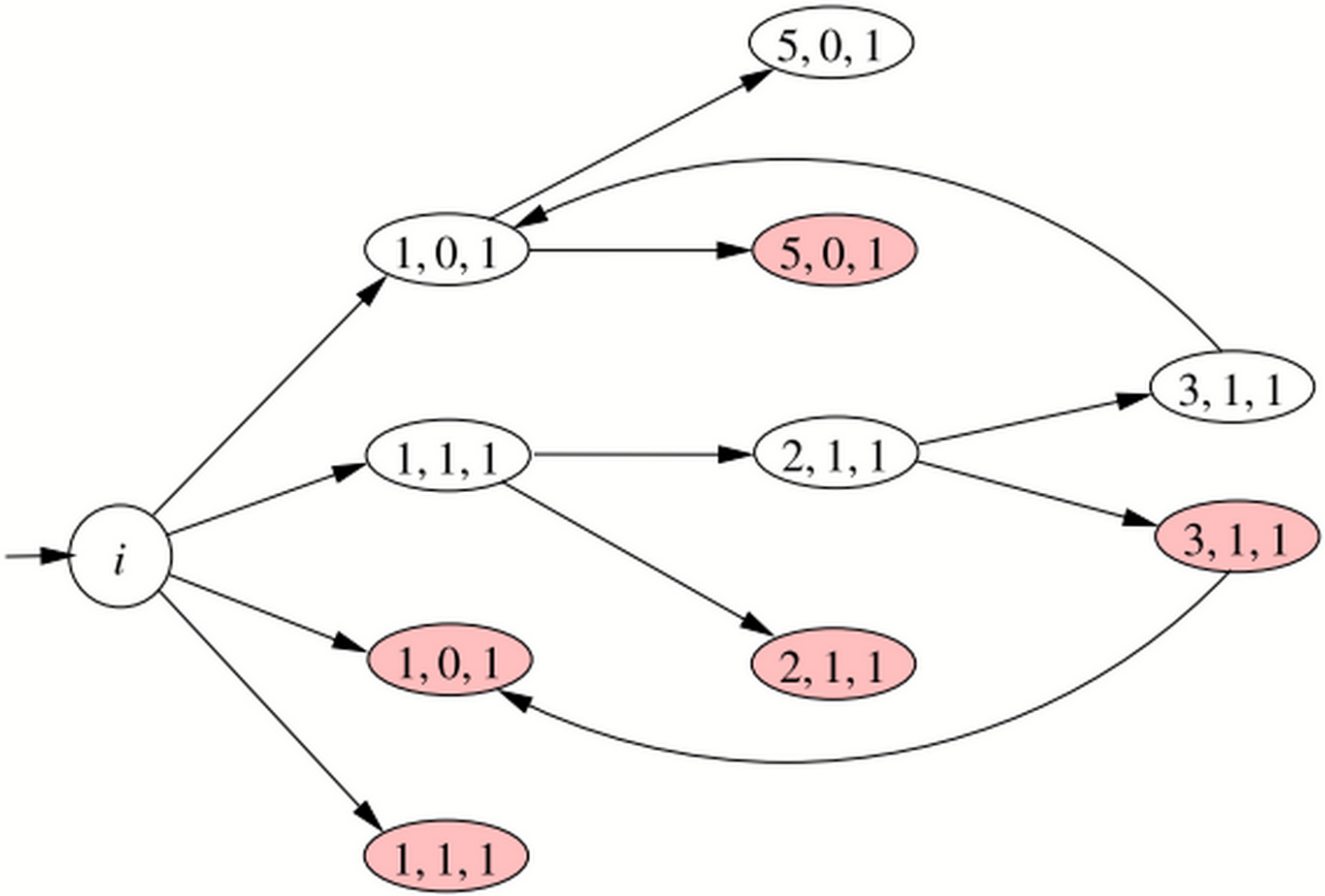
i
1, 0, 0
5, 0, 0
[5, 0, 0]
[1, 0, 0]
1, 1, 0
2, 1, 0
4, 1, 0
[2, 1, 0]
[4, 1, 0]
[1, 1, 0]
[1, 1, 0]
[1, 0, 1]
1, 0, 1
5, 0, 1
[5, 0, 1]
[1, 1, 1]
[1, 0, 1]
1, 1, 1
2, 1, 1
3, 1, 1
[2, 1, 1]
[3, 1, 1]

Is there a full execution such that
- initially y=1,
- finally y=0, and
- y never increases ?

Potential executions satisfying the property:

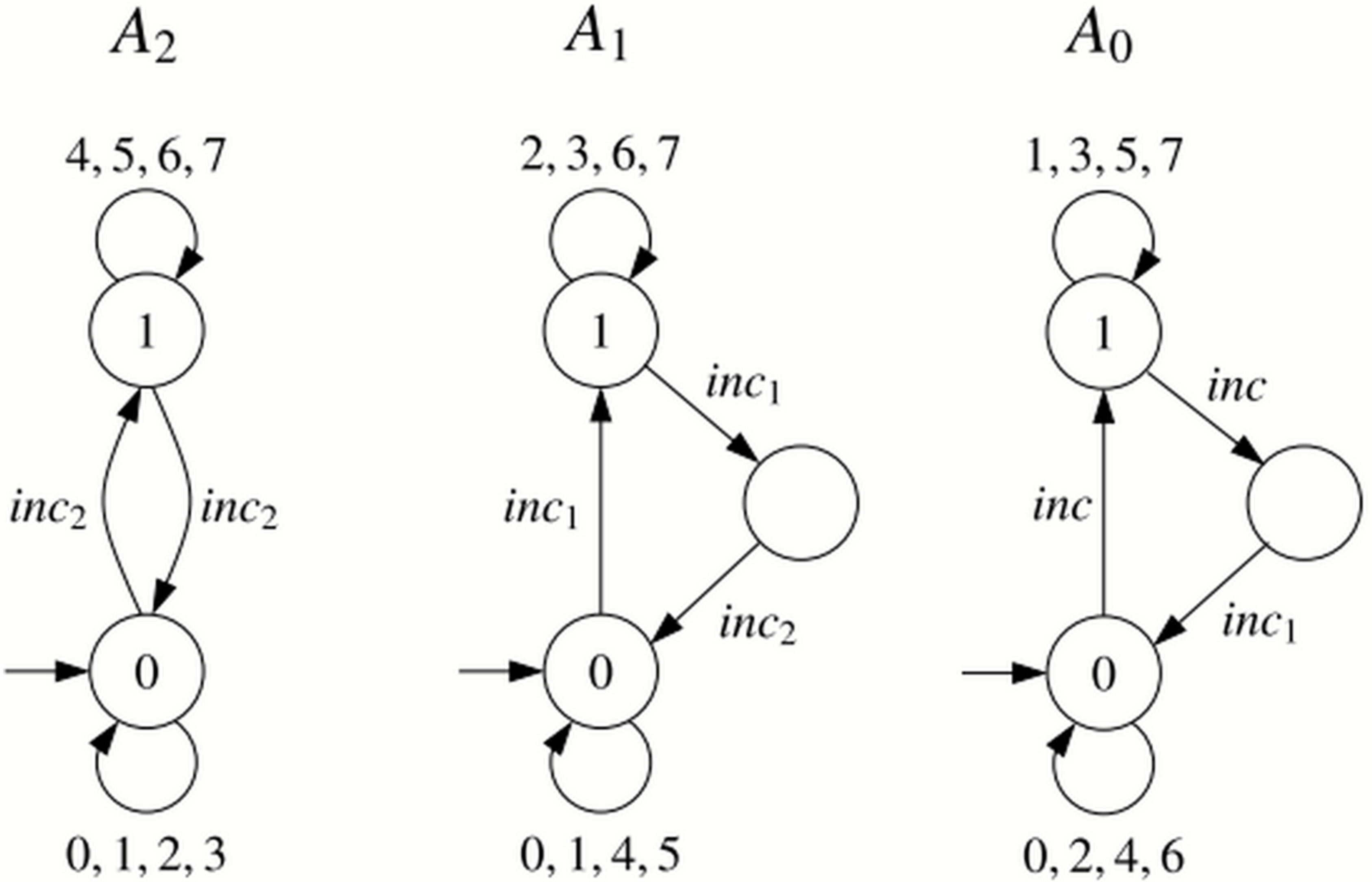
Y1 Y1* Y0* (L5 inters Y0)





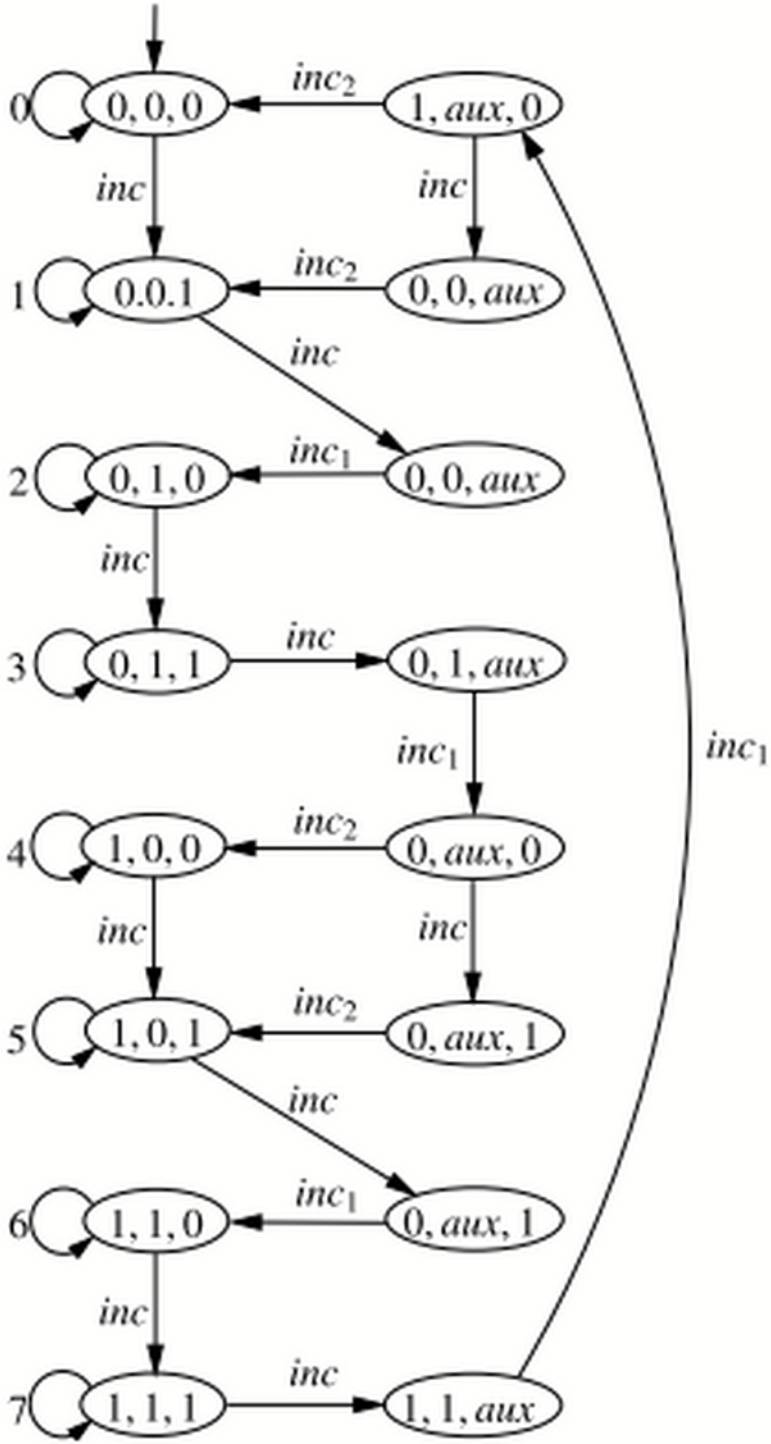
i
1, 0, 1
1, 1, 1
1, 0, 1
1, 1, 1
5, 0, 1
5, 0, 1
2, 1, 1
2, 1, 1
3, 1, 1
3, 1, 1

Networks of automata



A *network of automata* is a tuple $\mathcal{A} = \langle A_1, \ldots, A_n \rangle$ of NFAs with pairwise disjoint sets of states. Each NFA has its own alphabet Σ_i (the alphabets $\Sigma_1, \ldots, \Sigma_n$ are not necessarily pairwise disjoint). Alphabet letters are called *actions*. Given an action a, we say that the i-th NFA *participates in a* if $a \in \Sigma_i$.

A *configuration* of a network is a tuple $\langle q_1, \ldots, q_n \rangle$ of states, where $q_i \in Q_i$ for every $i \in \{1, \ldots, n\}$. An action a is *enabled* at a configuration $\langle q_1, \ldots, q_n \rangle$ if for every $i \in \{1, \ldots, n\}$ *such that* A_i *participates in* a there is a transition $(q_i, a, q_i') \in \delta_i$. If an action is enabled, then it can *occur*, and its occurrence makes *all participating NFAs* A_i move to the state q_i', while the non-participating NFAs *do not change their state*.



0
0, 0, 0
inc₂
1, aux, 0
inc
inc
1
0.0.1
inc₂
0, 0, aux
inc
2
0, 1, 0
inc₁
0, 0, aux
inc
3
0, 1, 1
inc
0, 1, aux
inc₁
inc₁
4
1, 0, 0
inc₂
0, aux, 0
inc
inc
5
1, 0, 1
inc₂
0, aux, 1
inc
6
1, 1, 0
inc₁
0, aux, 1
inc
7
1, 1, 1
inc
1, 1, aux

AsyncProduct$(A_1, \ldots, A_n)$

Input: a network of automata $\mathcal{A} = A_1, \ldots A_n$, where

$A_1 = (Q_1, \Sigma_1, \delta_1, q_{01}, Q_1), \ldots, A_n = (Q_n, \Sigma_n, \delta_n, q_{0n}, Q_n)$

Output: the asynchronous product $A_1 \otimes \cdots \otimes A_n = (Q, \Sigma, \delta, q_0, F)$

$Q, \delta, F \leftarrow \emptyset$
$q_0 \leftarrow [q_{01}, \ldots, q_{0n}]$
$W \leftarrow \{[q_{01}, \ldots, q_{0n}]\}$
while $W \neq \emptyset$ **do**
  pick $[q_1, \ldots, q_n]$ **from** W
  add $[q_1, \ldots, q_n]$ **to** Q
  add $[q_1, \ldots, q_n]$ **to** F
  for all $a \in \Sigma_1 \cup \ldots \cup \Sigma_n$ **do**
    for all $i \in [1..n]$ **do**
      if $a \in \Sigma_i$ **then** $Q'_i \leftarrow \delta_i(q_i, a)$ **else** $Q'_i = \{q_i\}$
    for all $[q'_1, \ldots, q'_n] \in Q'_1 \times \ldots \times Q'_n$ **do**
      if $[q'_1, \ldots, q'_n] \notin Q$ **then add** $[q'_1, \ldots, q'_n]$ **to** W
      add $([q_1, \ldots, q_n], a, [q'_1, \ldots, q'_n])$ **to** δ
return $(Q, \Sigma, \delta, q_0, F)$

Modelling concurrent programs

Lamport-Burns mutex algorithm:

Shared variables:

for every $i \in \{1,\ldots,n\}$:

$flag(i) \in \{0,1\}$, initially 0, writable by i, readable by all $j \neq i$

Process i:

try_i

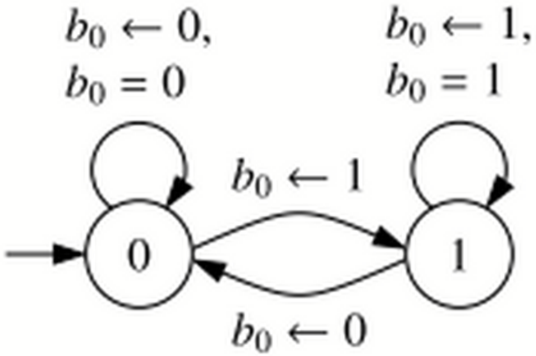
```
L:  flag(i) := 0
    for j ∈ {1,…,i-1} do
        if flag(j) = 1 then go to L
    flag(i) := 1
    for j ∈ {1,…,i-1} do
        if flag(j) = 1 then go to L
M:  for j ∈ {i+1,…,n} do
    if flag(j) = 1 then go to M
```

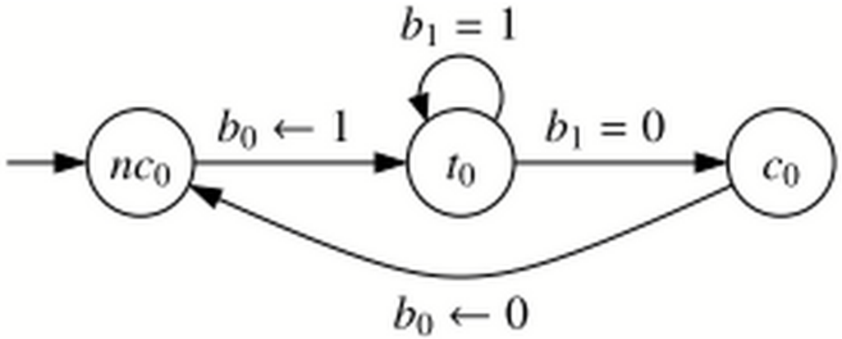
$crit_i$

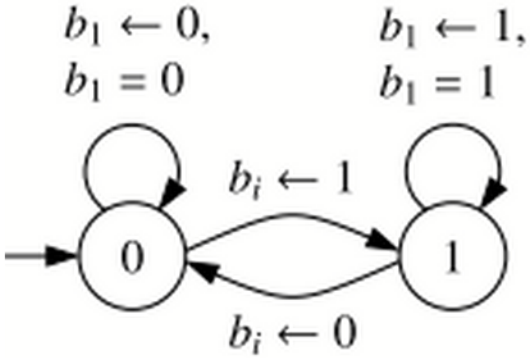
$exit_i$

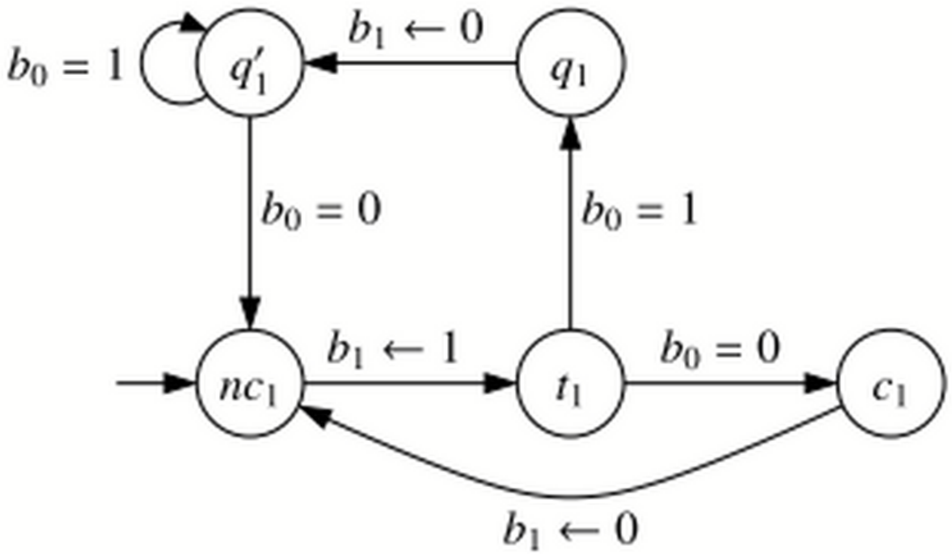
$flag(i) := 0$

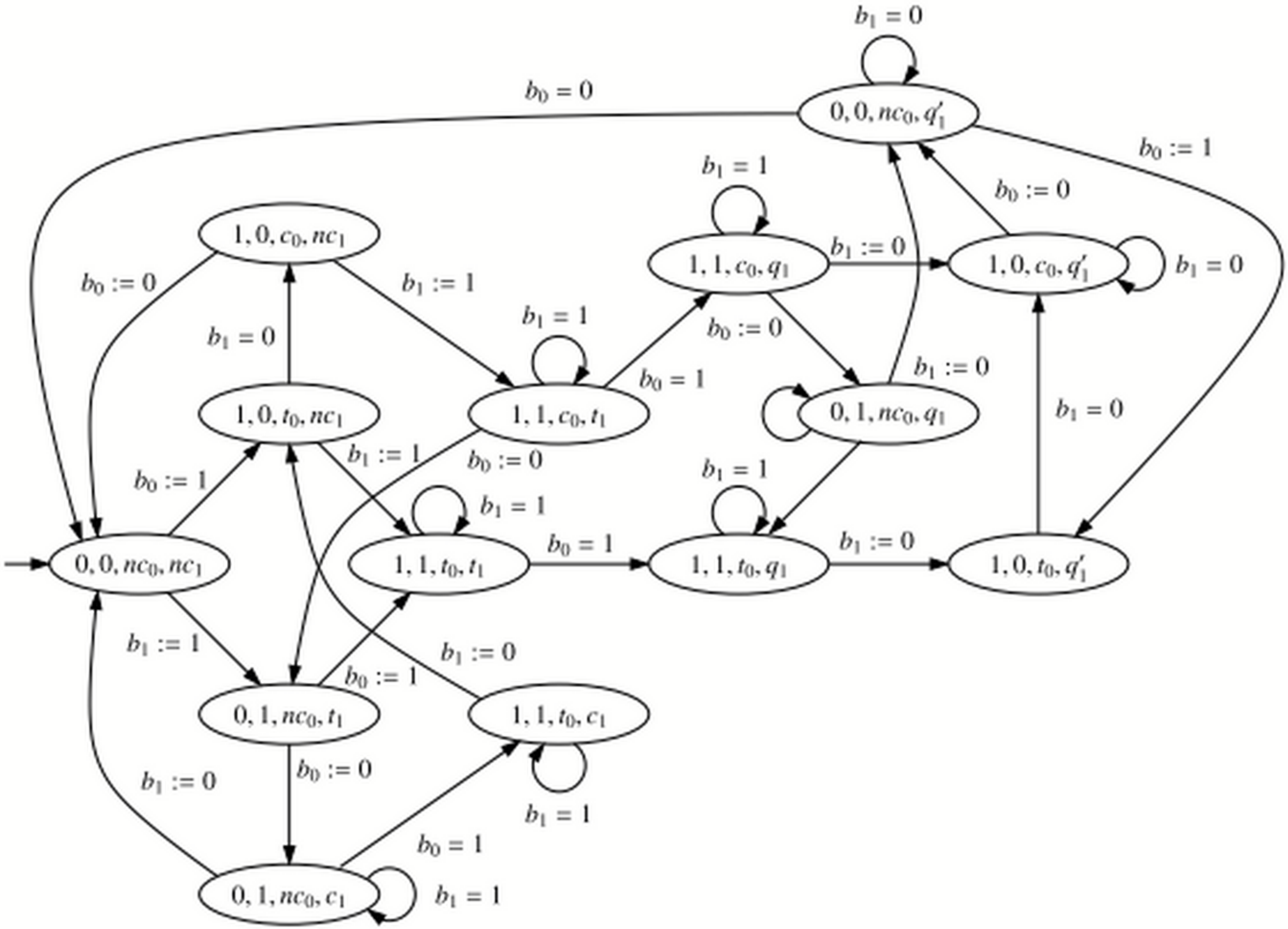
rem_i


$b_0 \leftarrow 0,$
$b_0 = 0$
$b_0 \leftarrow 1,$
$b_0 = 1$
$b_0 \leftarrow 1$
0
1
$b_0 \leftarrow 0$


$b_1 = 1$
$b_0 \leftarrow 1$
$b_1 = 0$
nc_0
t_0
c_0
$b_0 \leftarrow 0$


$b_1 \leftarrow 0,$
$b_1 = 0$
$b_1 \leftarrow 1,$
$b_1 = 1$
$b_i \leftarrow 1$
0
1
$b_i \leftarrow 0$


$b_0 = 1$
q'_1
$b_1 \leftarrow 0$
q_1
$b_0 = 0$
$b_0 = 1$
nc_1
$b_1 \leftarrow 1$
t_1
$b_0 = 0$
c_1
$b_1 \leftarrow 0$



$b_1 = 0$
$b_0 = 0$
$0, 0, nc_0, q'_1$
$b_0 := 1$
$b_1 = 1$
$b_0 := 0$
$1, 0, c_0, nc_1$
$b_1 := 0$
$1, 1, c_0, q_1$
$1, 0, c_0, q'_1$
$b_1 = 0$
$b_0 := 0$
$b_1 := 1$
$b_1 = 1$
$b_1 = 0$
$b_0 := 0$
$b_0 = 1$
$b_1 := 0$
$1, 0, t_0, nc_1$
$1, 1, c_0, t_1$
$0, 1, nc_0, q_1$
$b_1 = 0$
$b_1 := 1$
$b_0 := 0$
$b_1 = 1$
$b_0 := 1$
$b_1 = 1$
$b_0 = 1$
$b_1 := 0$
$0, 0, nc_0, nc_1$
$1, 1, t_0, t_1$
$1, 1, t_0, q_1$
$1, 0, t_0, q'_1$
$b_1 := 1$
$b_1 := 0$
$b_0 := 1$
$0, 1, nc_0, t_1$
$1, 1, t_0, c_1$
$b_0 := 0$
$b_1 := 0$
$b_1 = 1$
$b_0 = 1$
$0, 1, nc_0, c_1$
$b_1 = 1$

Checking properties

Deadlock freedom

Bounded overtaking: potential executions violating

$$\Sigma^* T_0 (\Sigma \setminus C_0)^* C_1 (\Sigma \setminus C_0)^* N C_1 (\Sigma \setminus C_0)^* C_1 \Sigma^*$$

$CheckViol(A_1, \ldots, A_n, V)$

Input: a network $\langle A_1, \ldots A_n \rangle$, where
$A_i = (Q_i, \Sigma_i, \delta_i, q_{0i}, Q_i)$;
an NFA $V = (Q_V, \Sigma_1 \cup \ldots \cup \Sigma_n, \delta_V, q_{0v}, F_v)$.

Output: **true** if $A_1 \otimes \cdots \otimes A_n \otimes V$ is nonempty, *false* otherwise.

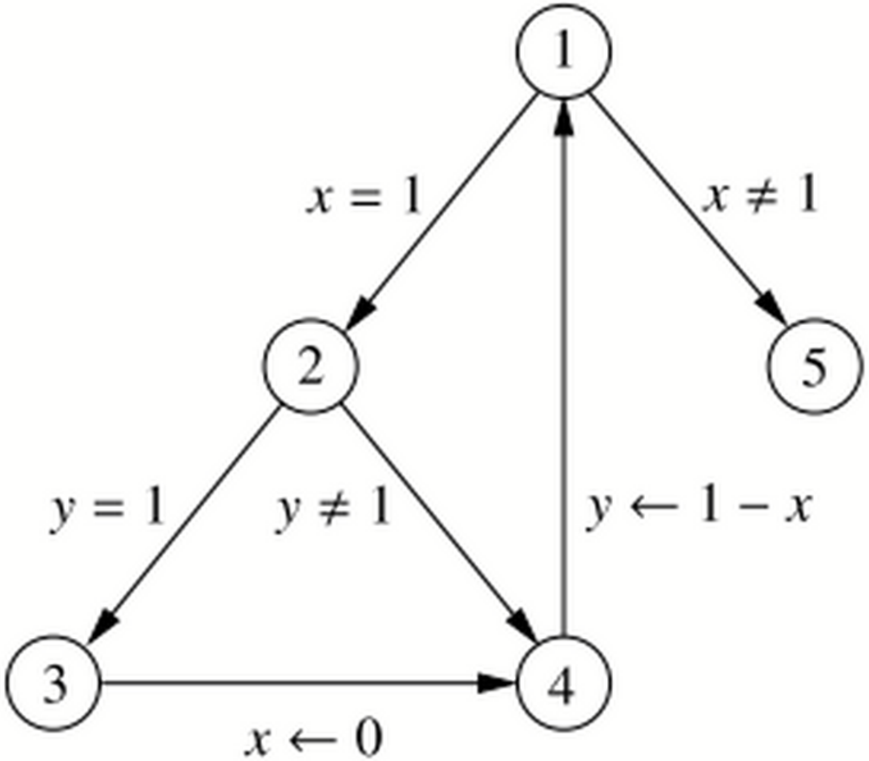
$Q \leftarrow \emptyset$; $q_0 \leftarrow [q_{01}, \ldots, q_{0n}, q_{0v}]$
$W \leftarrow \{q_0\}$
while $W \neq \emptyset$ **do**
    pick $[q_1, \ldots, q_n, q]$ **from** W
    add $[q_1, \ldots, q_n, q]$ **to** Q
    for all $a \in \Sigma_1 \cup \ldots \cup \Sigma_n$ **do**
        for all $i \in [1..n]$ **do**
            if $a \in \Sigma_i$ **then** $Q'_i \leftarrow \delta_i(q_i, a)$ **else** $Q'_i = \{q_i\}$
        $Q' \leftarrow \delta_V(q, a)$
        for all $[q'_1, \ldots, q'_n, q'] \in Q'_1 \times \ldots \times Q'_n \times Q'$ **do**
            if $\bigwedge_{i=1}^{n} q'_i \in F_i$ **and**$q \in F$ **then return true**
            if $[q'_1, \ldots, q'_n, q'] \notin Q$ **then add** $[q'_1, \ldots, q'_n, q']$ **to** W
return false

The state-explosion problem

Theorem 9.7 *The following problem is PSPACE-complete.*
Given: A network of automata $A_1, \ldots, A_n$ *over alphabets* $\Sigma_1, \ldots, \Sigma_n$*, a NFA* V *over* $\Sigma_1 \cup \ldots \cup \Sigma_n$.
Decide: if $\mathcal{L}(A_1 \otimes \cdot \otimes A_n \otimes V) \neq \emptyset$.

Symbolic exploration

```
while x = 1 do
   if y = 1 then
      x ← 0
   y ← 1 − x
end
```



An edge of the flowgraph leading from node ℓ to node ℓ' can be associated a *step relation* $S_{\ell,\ell'}$ containing all pairs of configurations $([\ell, x_0, y_0], [\ell', x'_0, y'_0])$ such that if at control point ℓ the current values of the variables are x_0, y_0, then the program can take a step after which the new control point is ℓ', and the new values are x'_0, y'_0. For instance, for the edge leading from node 4 to node 1 we have

$$S_{4,1} = \left\{ \left([4, x_0, y_0], [1, x'_0, y'_0] \right) \mid x'_0 = x_0, y'_0 = 1 - x_0 \right\}$$

and for the edge leading from 1 to 2

$$S_{1,2} = \left\{ \left([1, x_0, y_0], [2, x'_0, y'_0] \right) \mid x_0 = 1 = x'_0, y'_0 = y_0 \right\}$$

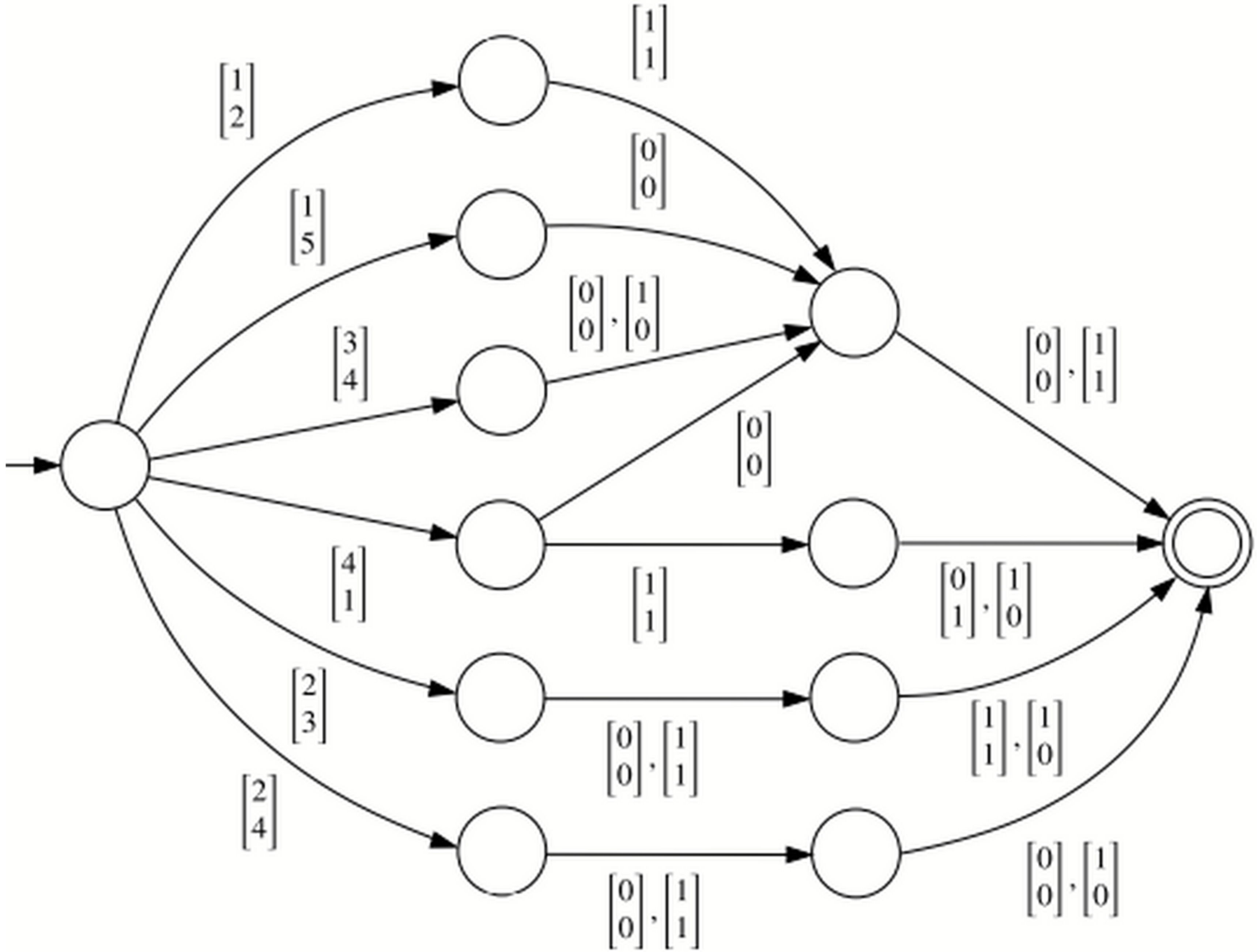
$$S = \bigcup_{a,b \in C} S_{a,b}$$

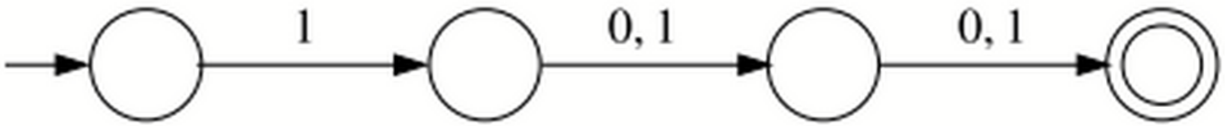
Reach(I, R)

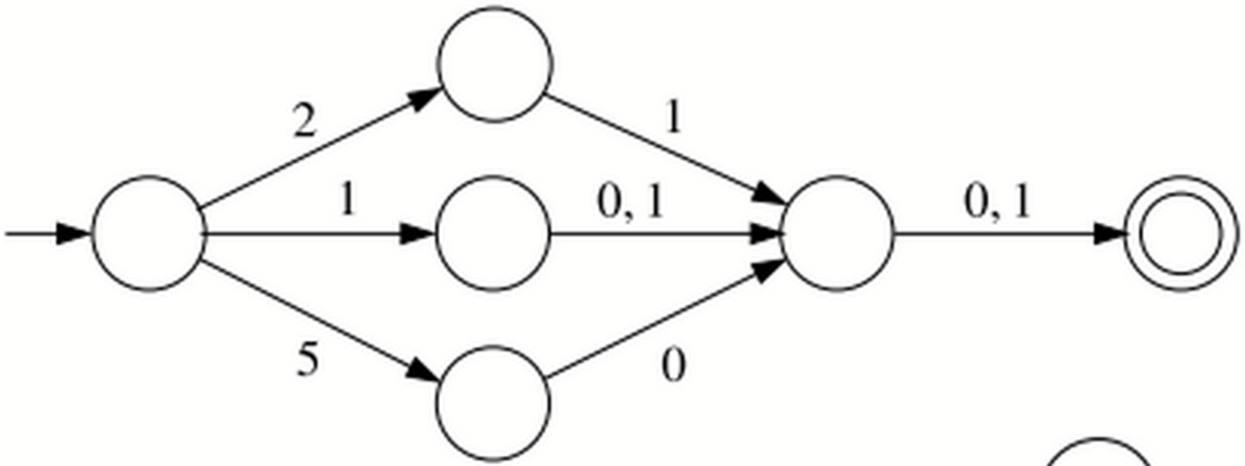
Input: set I of initial configurations; relation R

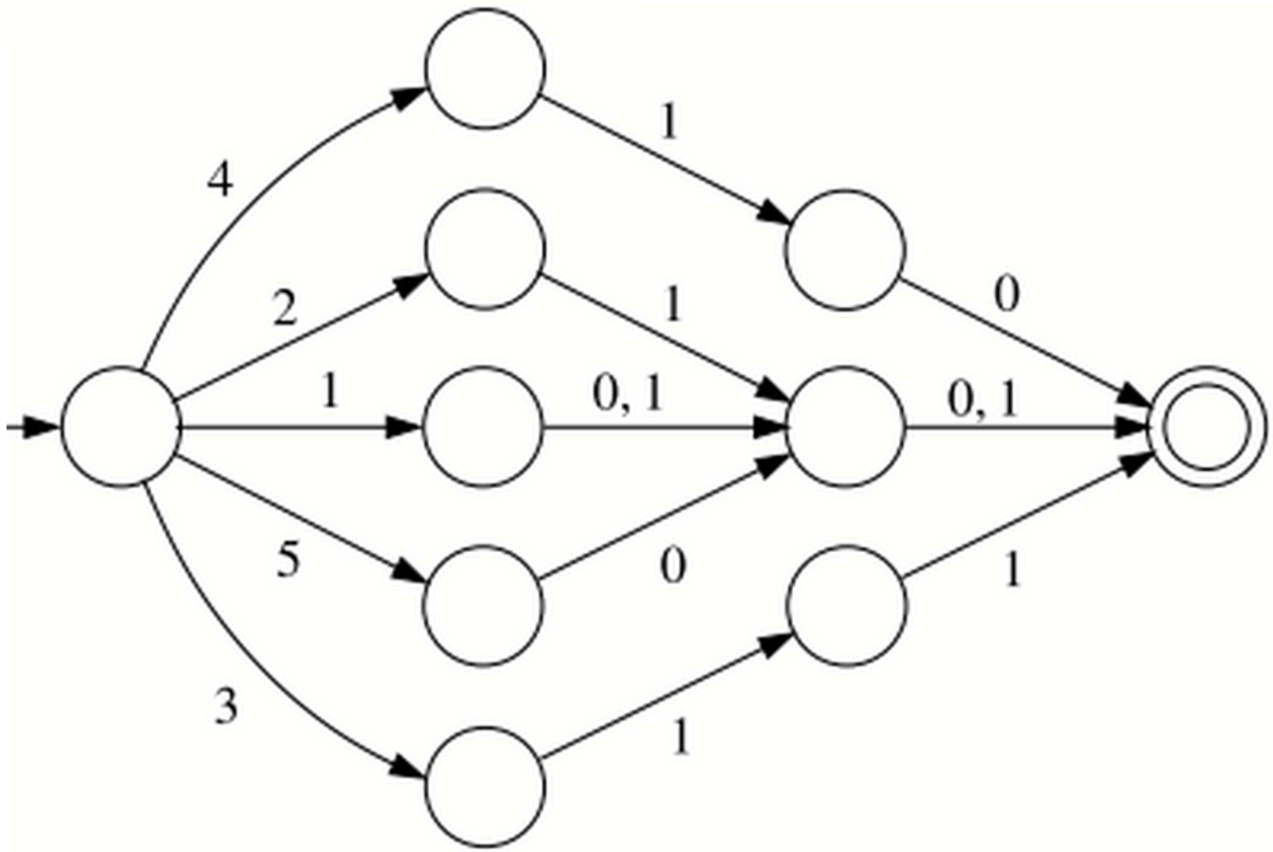
Output: set of configurations reachable form I

$OldP \leftarrow \emptyset$; $P \leftarrow I$
while $P \neq OldP$ **do**
    $OldP \leftarrow P$
    $P \leftarrow$ **Union**$(P,$ **Post**$(P, S))$
return P


$\begin{bmatrix}1\\2\end{bmatrix}$
$\begin{bmatrix}1\\1\end{bmatrix}$
$\begin{bmatrix}0\\0\end{bmatrix}$
$\begin{bmatrix}1\\5\end{bmatrix}$
$\begin{bmatrix}0\\0\end{bmatrix},\begin{bmatrix}1\\0\end{bmatrix}$
$\begin{bmatrix}3\\4\end{bmatrix}$
$\begin{bmatrix}0\\0\end{bmatrix},\begin{bmatrix}1\\1\end{bmatrix}$
$\begin{bmatrix}0\\0\end{bmatrix}$
$\begin{bmatrix}4\\1\end{bmatrix}$
$\begin{bmatrix}1\\1\end{bmatrix}$
$\begin{bmatrix}0\\1\end{bmatrix},\begin{bmatrix}1\\0\end{bmatrix}$
$\begin{bmatrix}2\\3\end{bmatrix}$
$\begin{bmatrix}1\\1\end{bmatrix},\begin{bmatrix}1\\0\end{bmatrix}$
$\begin{bmatrix}0\\0\end{bmatrix},\begin{bmatrix}1\\1\end{bmatrix}$
$\begin{bmatrix}2\\4\end{bmatrix}$
$\begin{bmatrix}0\\0\end{bmatrix},\begin{bmatrix}1\\0\end{bmatrix}$
$\begin{bmatrix}0\\0\end{bmatrix},\begin{bmatrix}1\\1\end{bmatrix}$


1
0, 1
0, 1


2
1
1
0, 1
0, 1
5
0


4
1
2
1
0
1
0, 1
0, 1
5
0
1
3
1

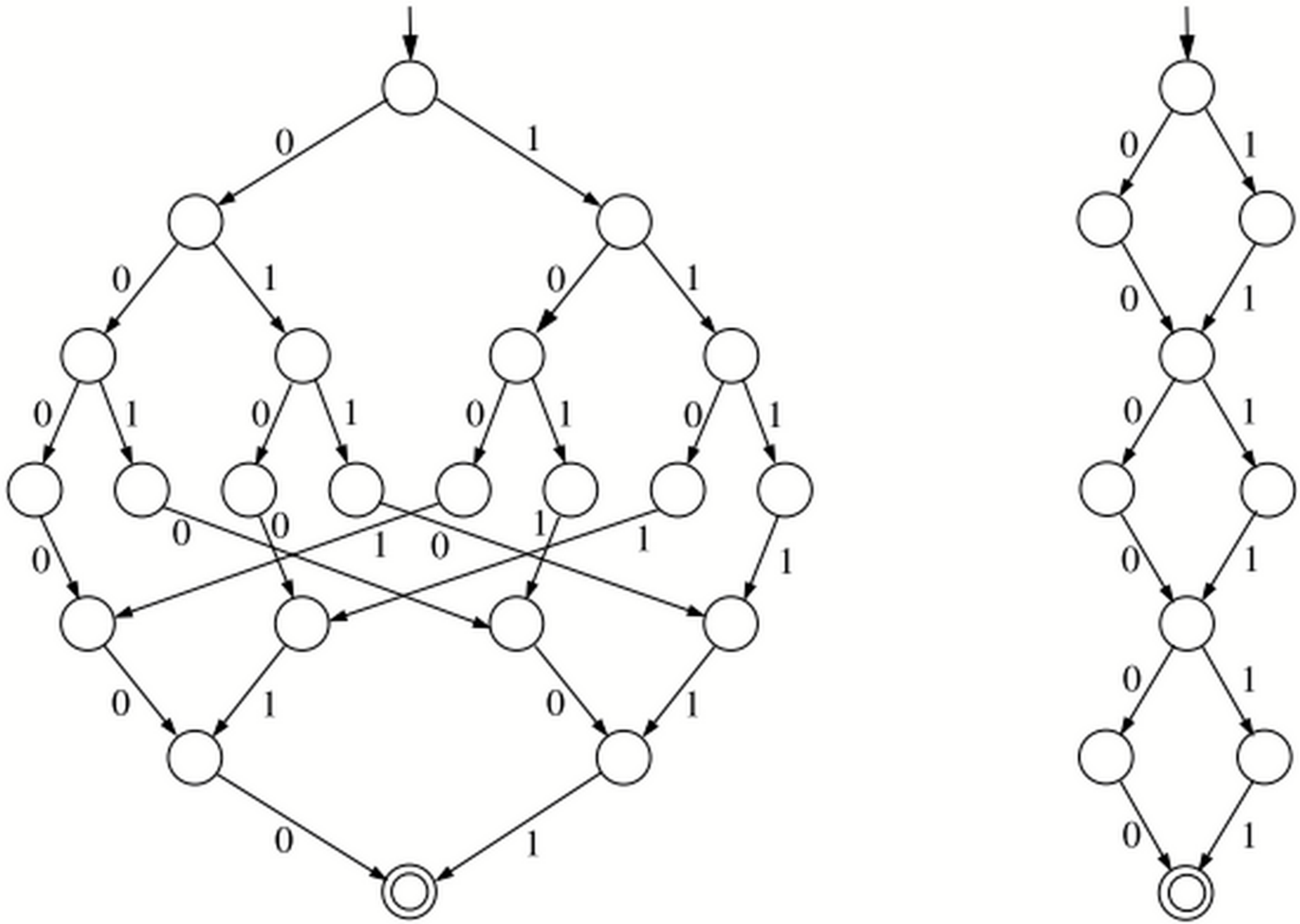
Variable orders

Example 9.8 Consider the set of tuples $X = \{[x_1, x_2, \ldots, x_{2k}] \mid x_1, \ldots, x_{2k} \in \{0, 1\}\}$, and the subset $Y \subseteq X$ of tuples satisfying $x_1 = x_k, x_2 = x_{k+1}, \ldots, x_k = x_{2k}$. Consider two possible encodings of a tuple $[x_1, x_2, \ldots, x_{2k}]$: by the word $x_1 x_2 \ldots x_{2k}$, and by the word $x_1 x_{k+1} x_2 x_{k+2} \ldots x_k x_{2k}$. In the first case, the encoding of Y for $k = 3$ is the language

$$L_1 = \{000000, 001001, 010010, 011011, 100100, 101101, 110110, 111111\}$$

and in the second the language

$$L_2 = \{000000, 000011, 001100, 001111, 110000, 110011, 111100, 111111\}$$



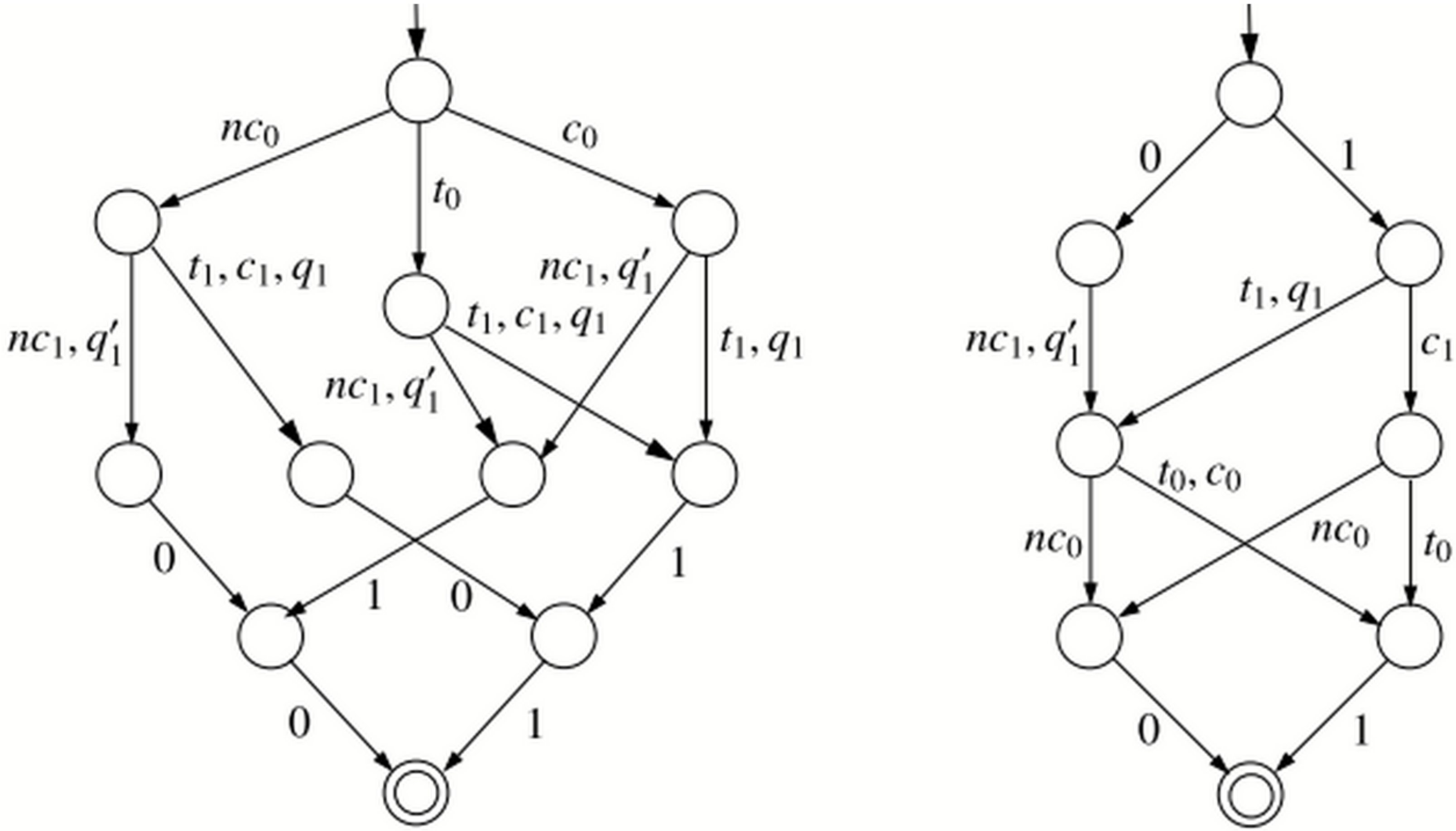


Figure 9.10: Minimal DFAs for the reachable configurations of Lamport's algorithm. On the left a configuration $\langle s_0, s_1, v_0, v_1, q\rangle$ is encoded by the word $s_0 s_1 v_0 v_1 q$, on the right by $v_1 s_1 s_0 v_0$.

Safety: nothing bad can happen
Liveness: something good eventually happens

More formally:

- safety property: violations are finite executions

- liveness properties: violations are infinite executions