

Verification

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

Reduce the verification problem to language inclusion between implementation and specification

```
1  while  $x = 1$  do  
2      if  $y = 1$  then  
3           $x \leftarrow 0$   
4       $y \leftarrow 1 - x$   
5  end
```

Configuration

Initial configuration

Execution, full execution, potential execution

```

1  while  $x = 1$  do
2      if  $y = 1$  then
3           $x \leftarrow 0$ 
4       $y \leftarrow 1 - x$ 
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.

Formalizing the approach

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

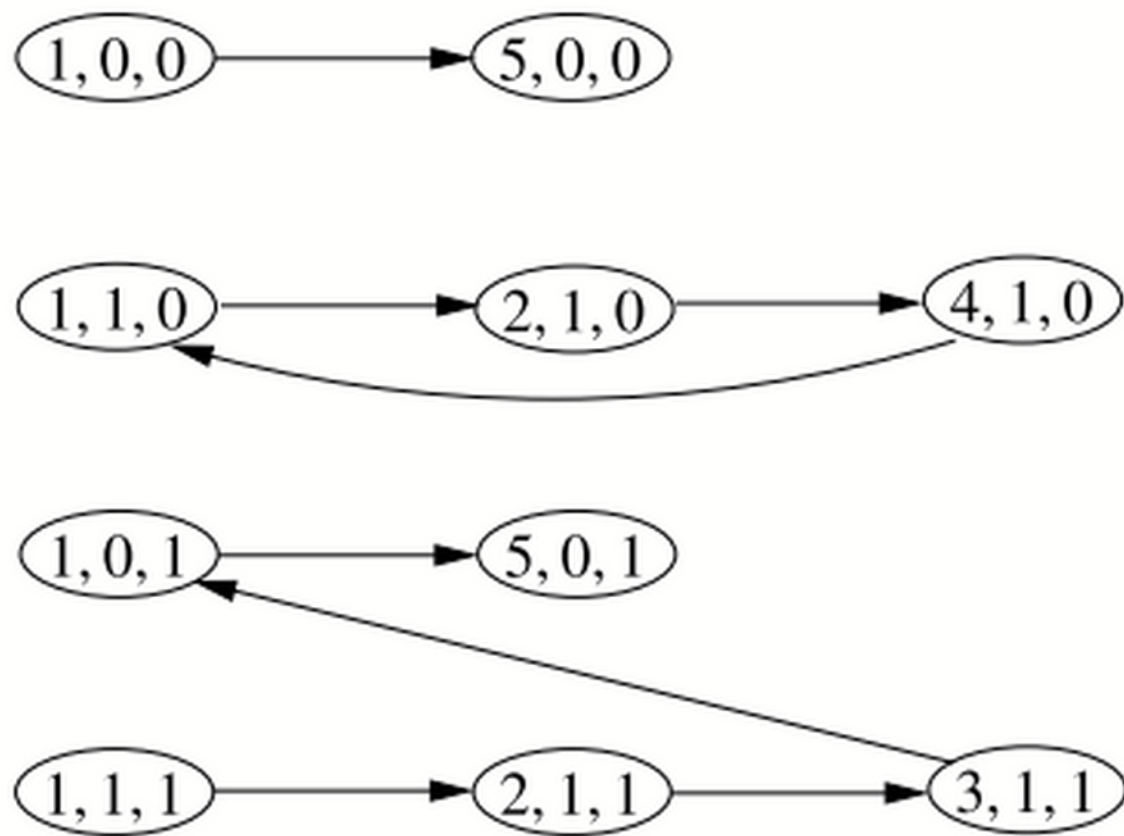
Implementation satisfies specification if :

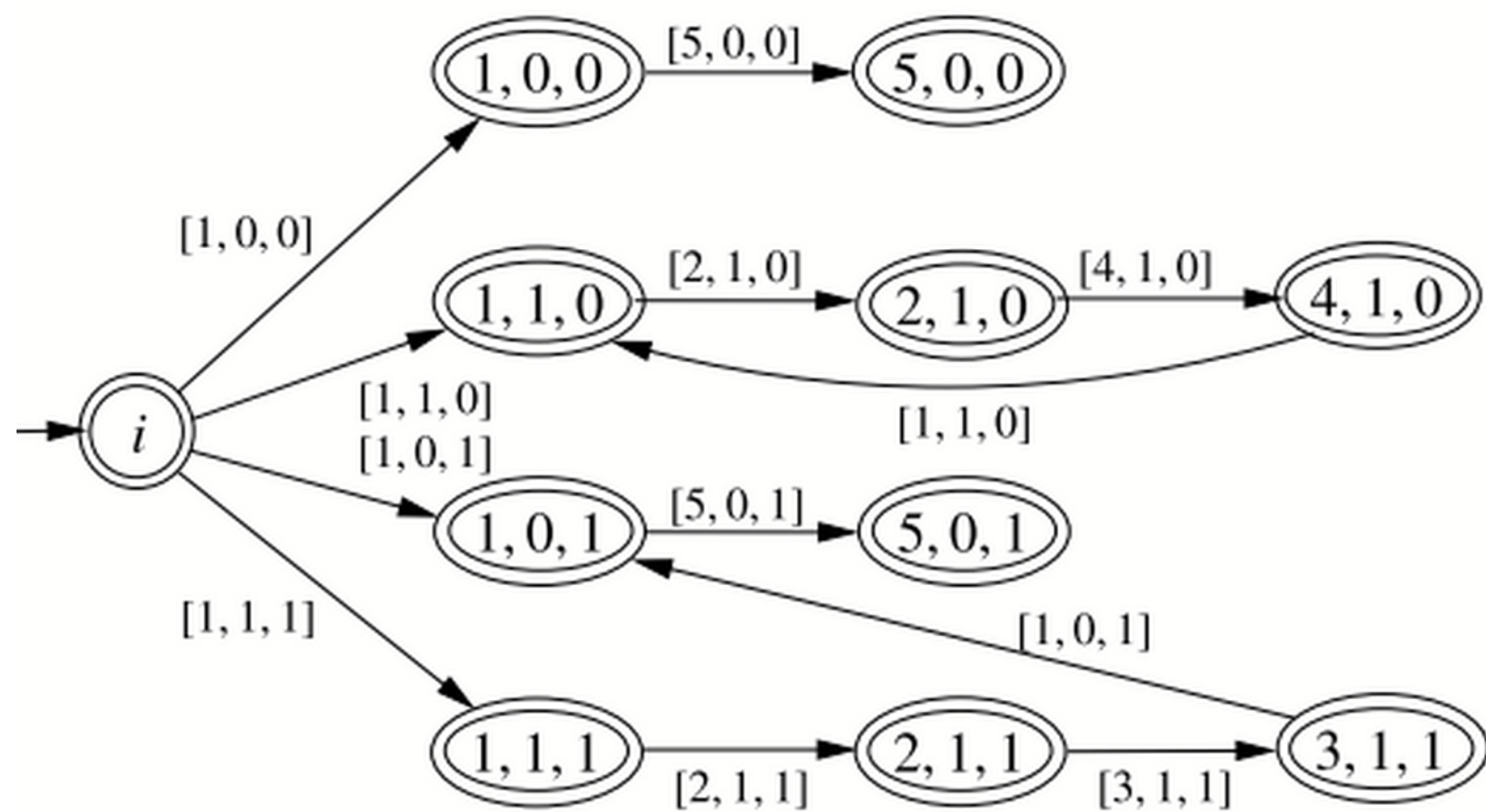
E included in P or intersection of E and V empty

If E and P regular: inclusion checkable with automata

If E and V regular: emptiness checkable with automata

How often does this happen?



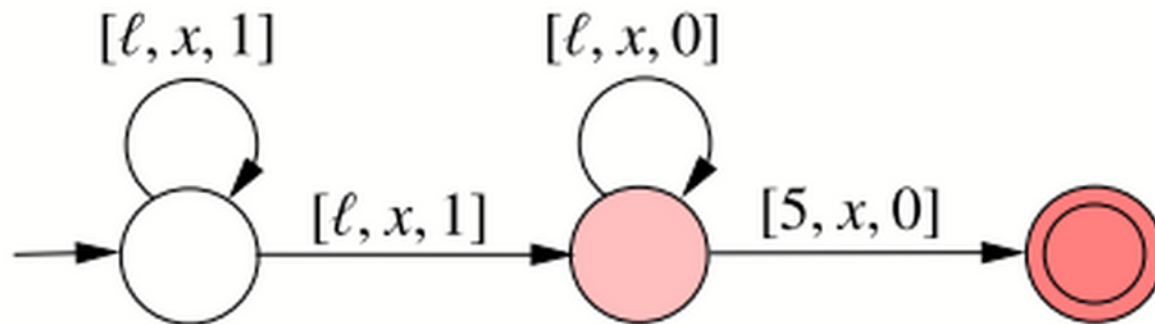


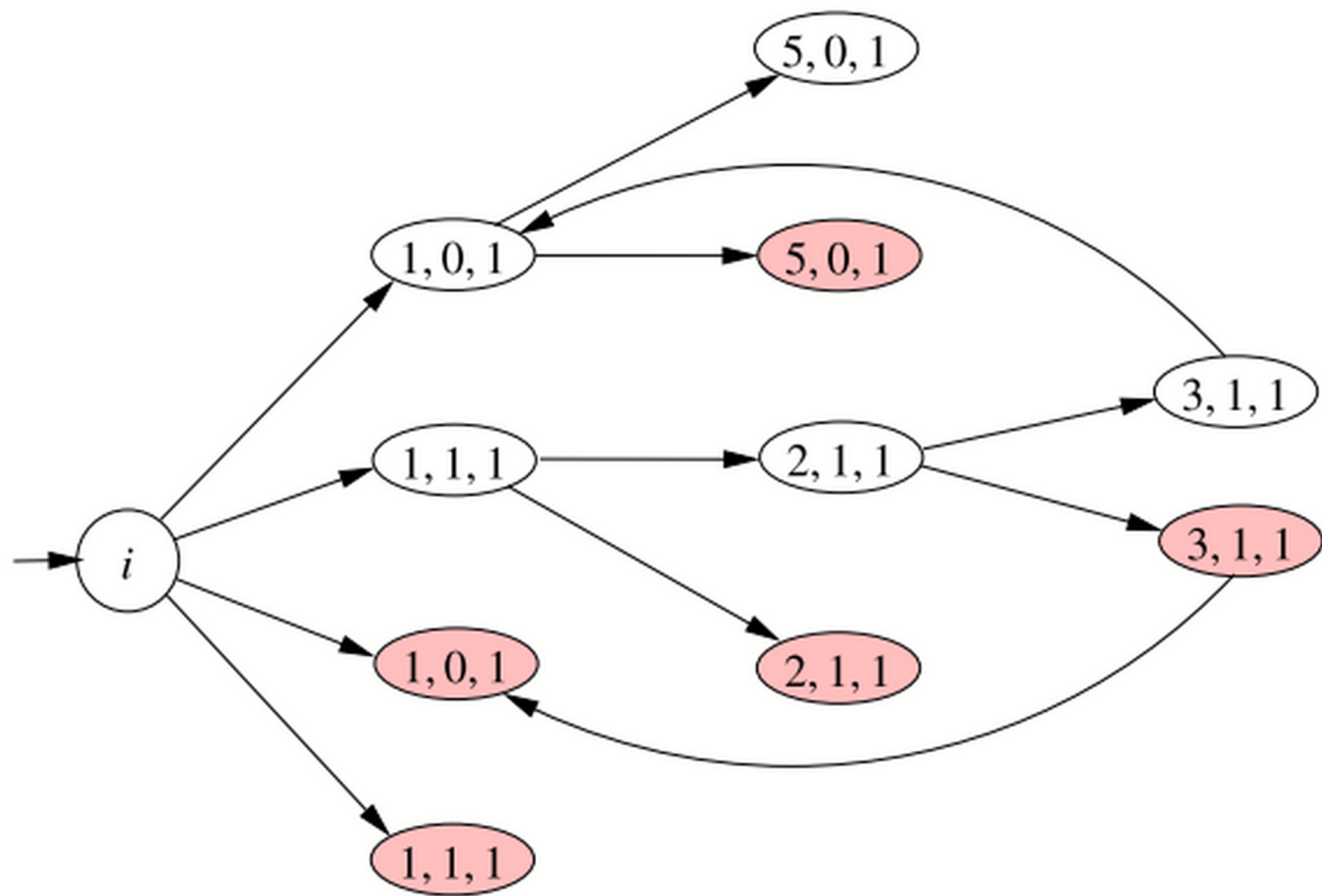
Is there a full execution such that

- initially $y=1$,
- finally $y=0$, and
- y never increases ?

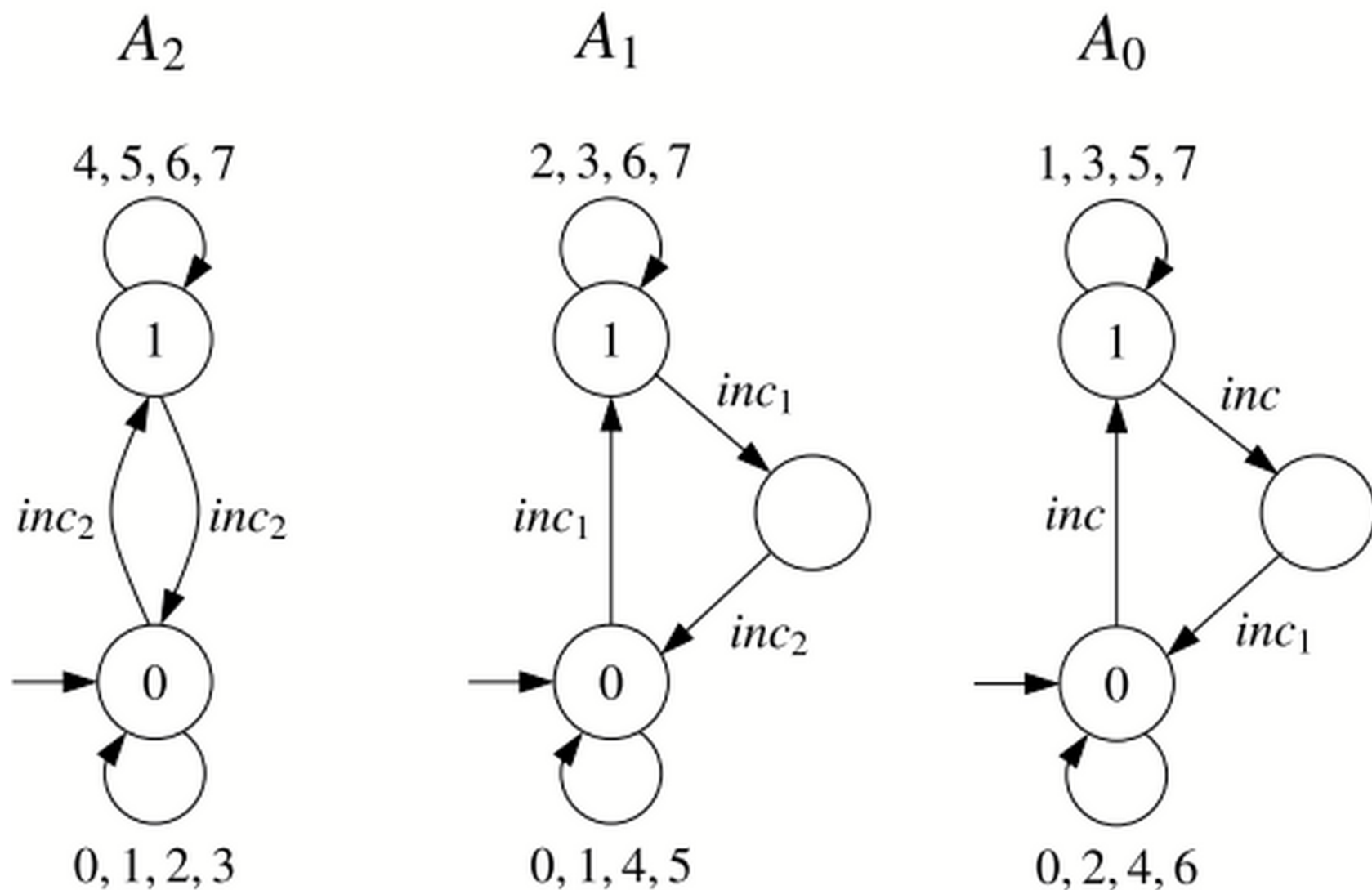
Potential executions satisfying the property:

$Y1 \ Y1^* \ Y0^* \ (L5 \text{ inters } Y0)$



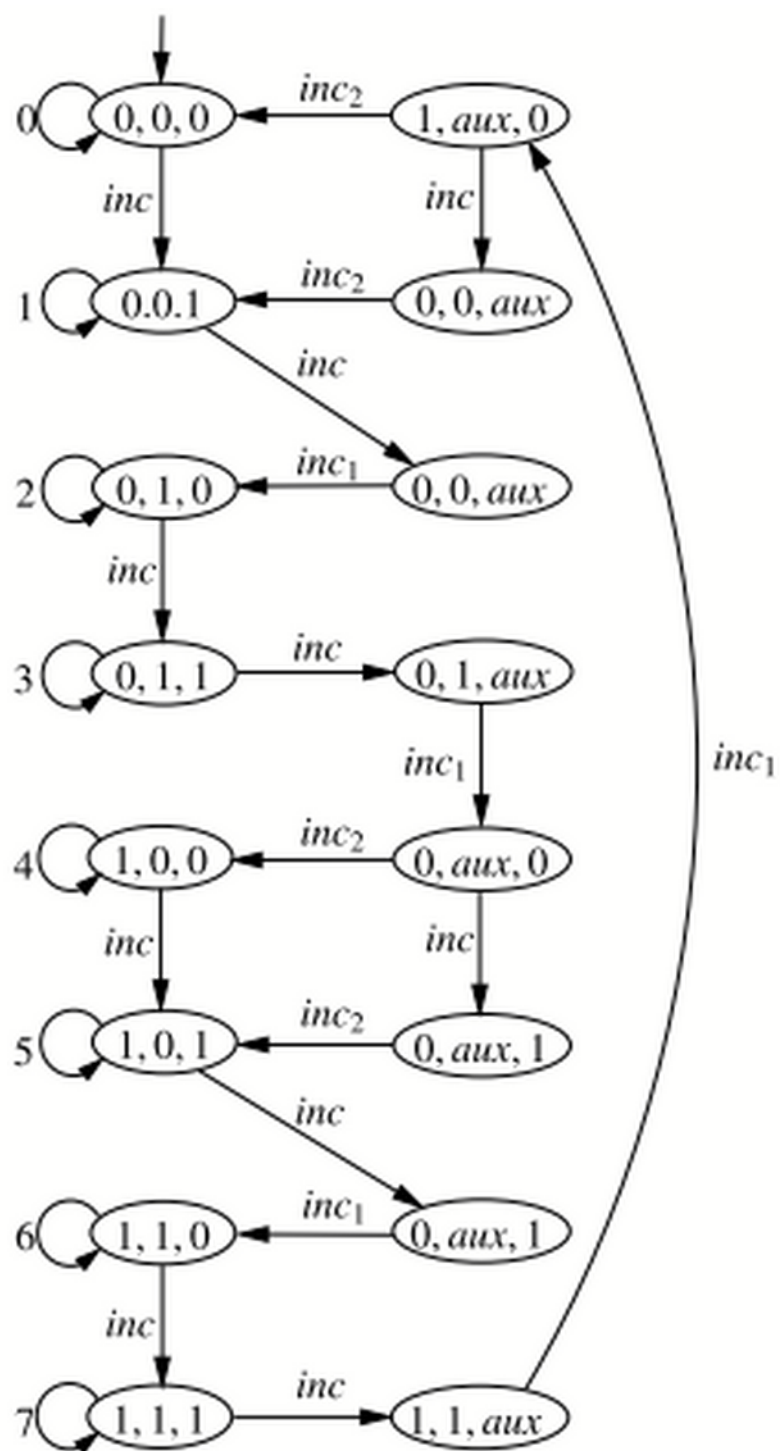


Networks of automata



A *network of automata* is a tuple $\mathcal{A} = \langle A_1, \dots, A_n \rangle$ of NFAs with pairwise disjoint sets of states. Each NFA has its own alphabet Σ_i (the alphabets $\Sigma_1, \dots, \Sigma_n$ 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 $\langle q_1, \dots, q_n \rangle$ of states, where $q_i \in Q_i$ for every $i \in \{1, \dots, n\}$. An action a is *enabled* at a configuration $\langle q_1, \dots, q_n \rangle$ if for every $i \in \{1, \dots, n\}$ 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*.



AsyncProduct($A_1, \dots, A_n$)

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

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

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

```

1   $Q, \delta, F \leftarrow \emptyset$ 
2   $q_0 \leftarrow [q_{01}, \dots, q_{0n}]$ 
3   $W \leftarrow \{[q_{01}, \dots, q_{0n}]\}$ 
4  while  $W \neq \emptyset$  do
5      pick  $[q_1, \dots, q_n]$  from  $W$ 
6      add  $[q_1, \dots, q_n]$  to  $Q$ 
7      add  $[q_1, \dots, q_n]$  to  $F$ 
8      for all  $a \in \Sigma_1 \cup \dots \cup \Sigma_n$  do
9          for all  $i \in [1..n]$  do
10             if  $a \in \Sigma_i$  then  $Q'_i \leftarrow \delta_i(q_i, a)$  else  $Q'_i = \{q_i\}$ 
11             for all  $[q'_1, \dots, q'_n] \in Q'_1 \times \dots \times Q'_n$  do
12                 if  $[q'_1, \dots, q'_n] \notin Q$  then add  $[q'_1, \dots, q'_n]$  to  $W$ 
13                 add  $([q_1, \dots, q_n], a, [q'_1, \dots, q'_n])$  to  $\delta$ 
14 return  $(Q, \Sigma, \delta, q_0, F)$ 
```

Modelling concurrent programs

Lamport-Burns mutex algorithm:

Shared variables:

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

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

Process i :

try_i

L: $\text{flag}(i) := 0$

for $j \in \{1, \dots, i-1\}$ do

if $\text{flag}(j) = 1$ then go to L

$\text{flag}(i) := 1$

for $j \in \{1, \dots, i-1\}$ do

if $\text{flag}(j) = 1$ then go to L

M: for $j \in \{i+1, \dots, n\}$ do

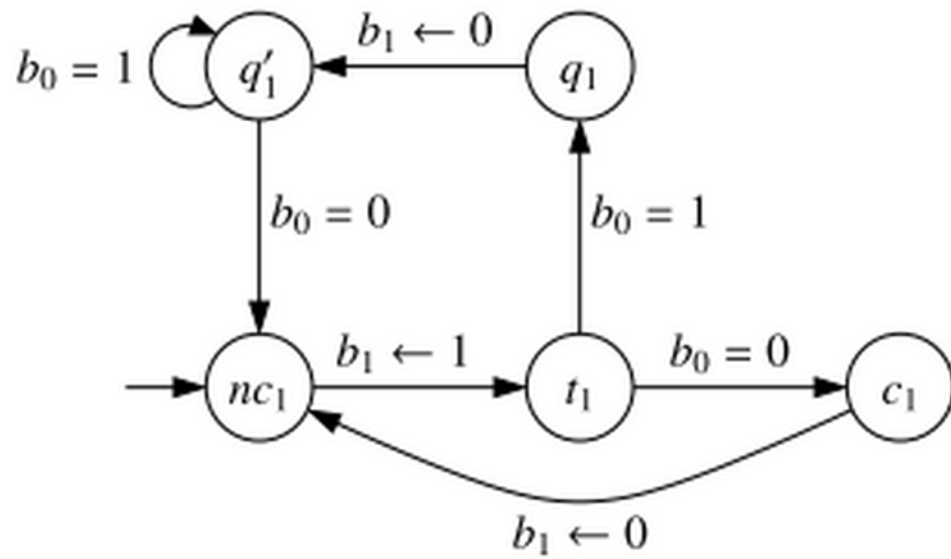
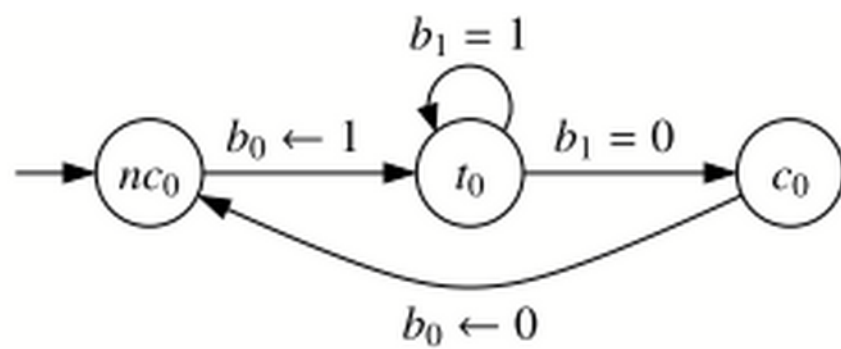
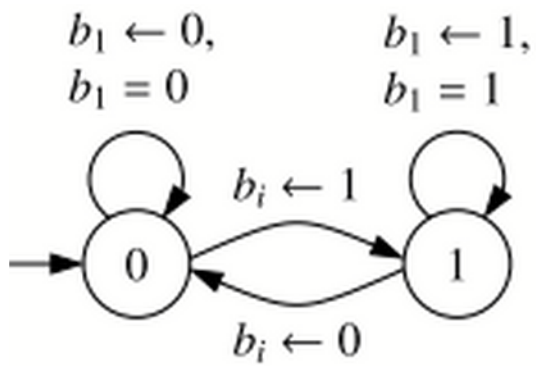
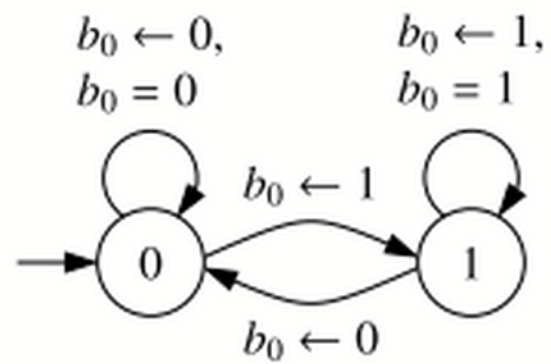
if $\text{flag}(j) = 1$ then go to M

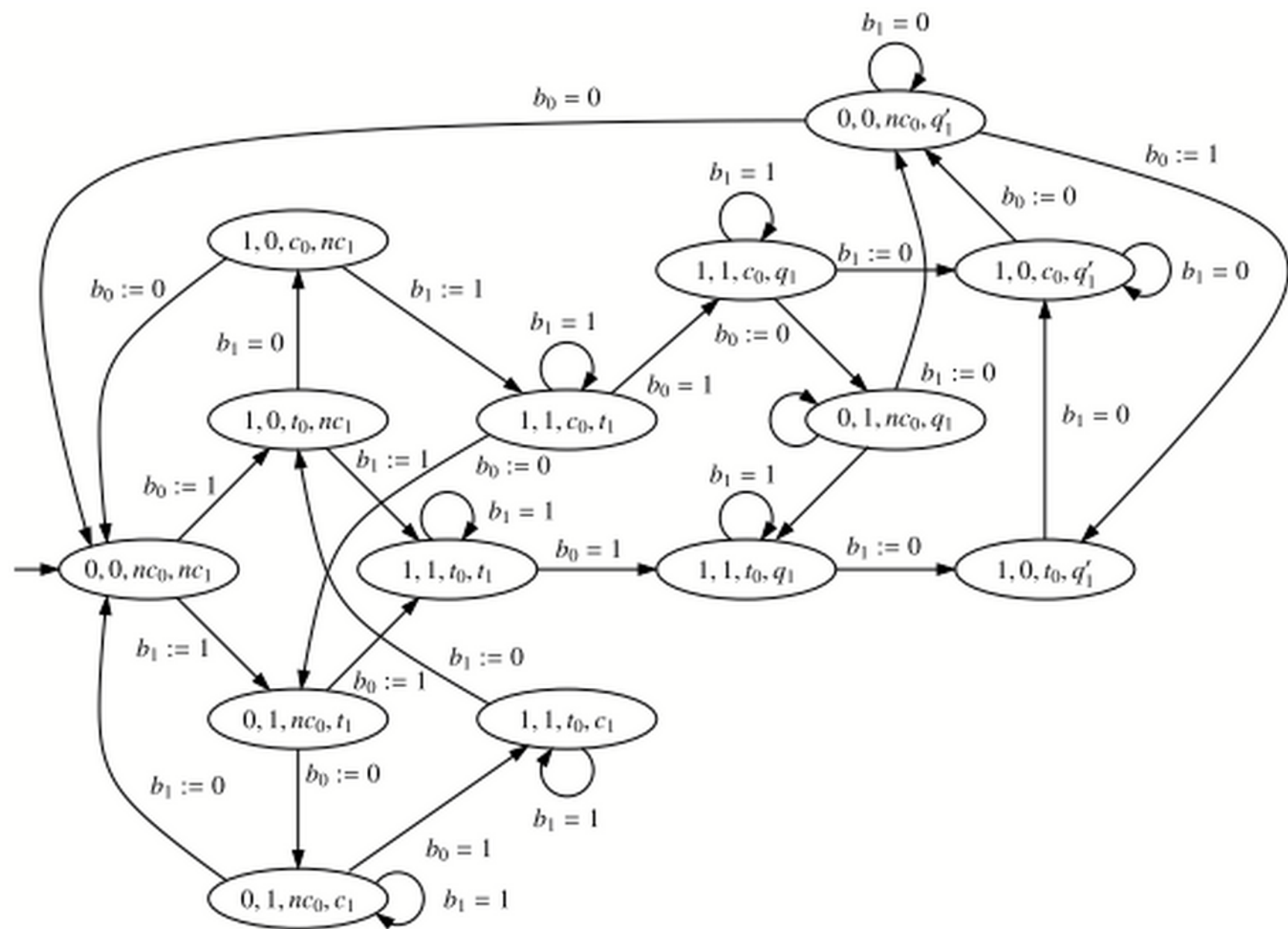
crit_i

exit_i

$\text{flag}(i) := 0$

rem_i





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($A_1, \dots, A_n, V$)

Input: a network $\langle A_1, \dots, A_n \rangle$, where

$A_i = (Q_i, \Sigma_i, \delta_i, q_{0i}, Q_i)$;

an NFA $V = (Q_V, \Sigma_1 \cup \dots \cup \Sigma_n, \delta_V, q_{0v}, F_v)$.

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

```

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

```

The state-explosion problem

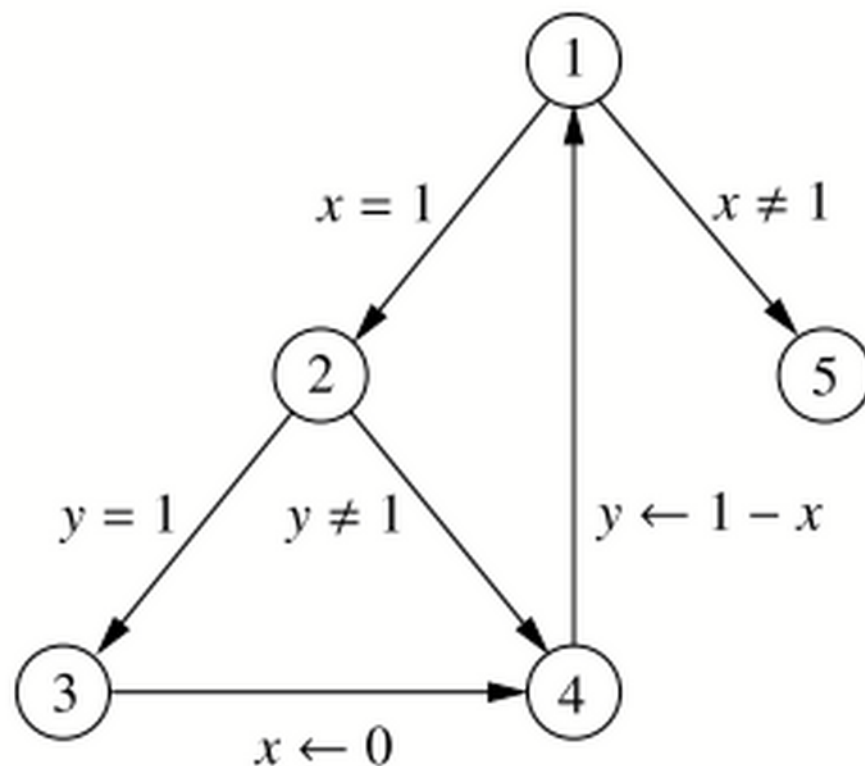
Theorem 9.7 *The following problem is PSPACE-complete.*

Given: A network of automata $A_1, \dots, A_n$ over alphabets $\Sigma_1, \dots, \Sigma_n$, a NFA V over $\Sigma_1 \cup \dots \cup \Sigma_n$.

Decide: if $\mathcal{L}(A_1 \otimes \dots \otimes A_n \otimes V) \neq \emptyset$.

Symbolic exploration

```
1  while  $x = 1$  do  
2    if  $y = 1$  then  
3       $x \leftarrow 0$   
4     $y \leftarrow 1 - x$   
5  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

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

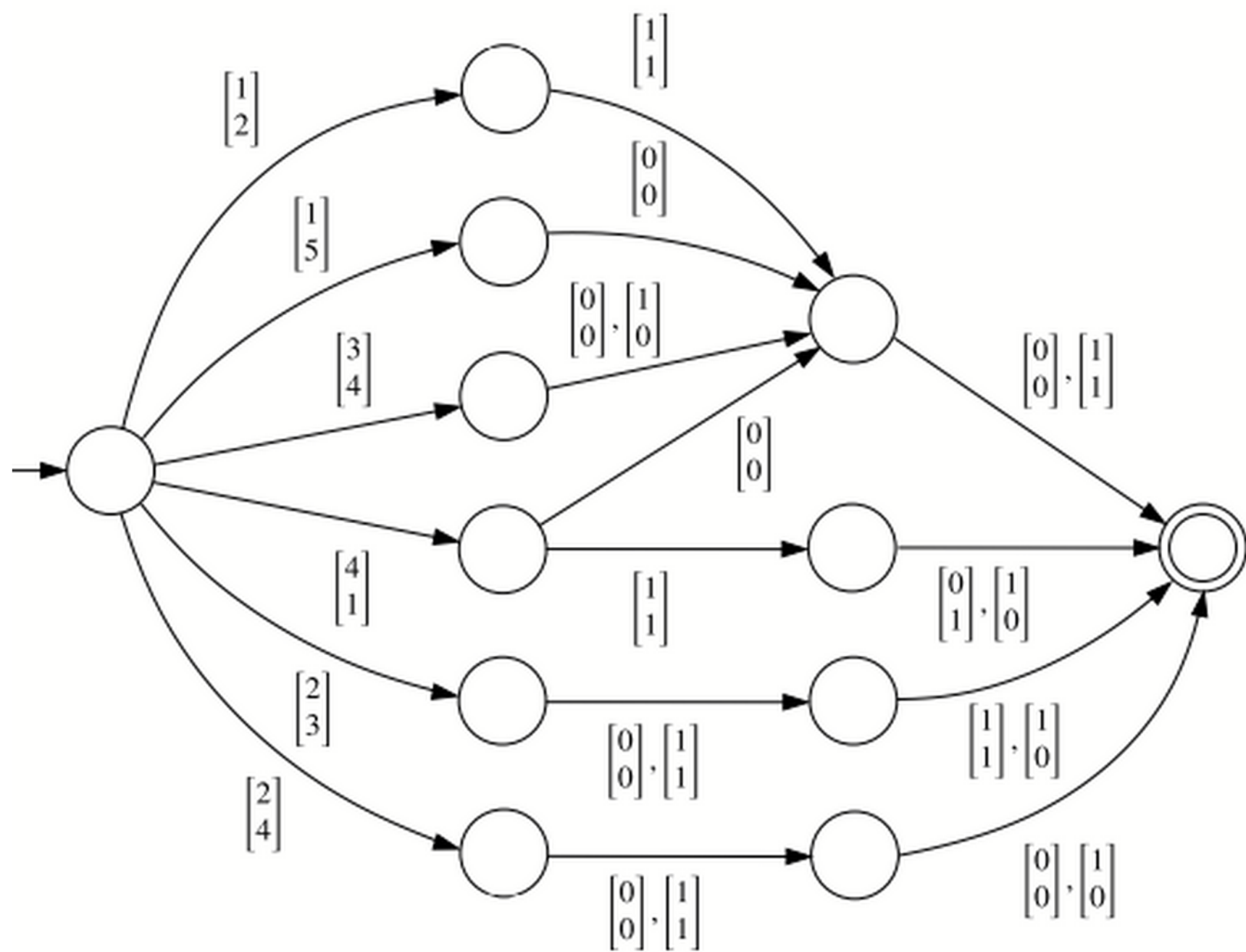
$$S = \bigcup_{a,b \in C} S_{a,b}$$

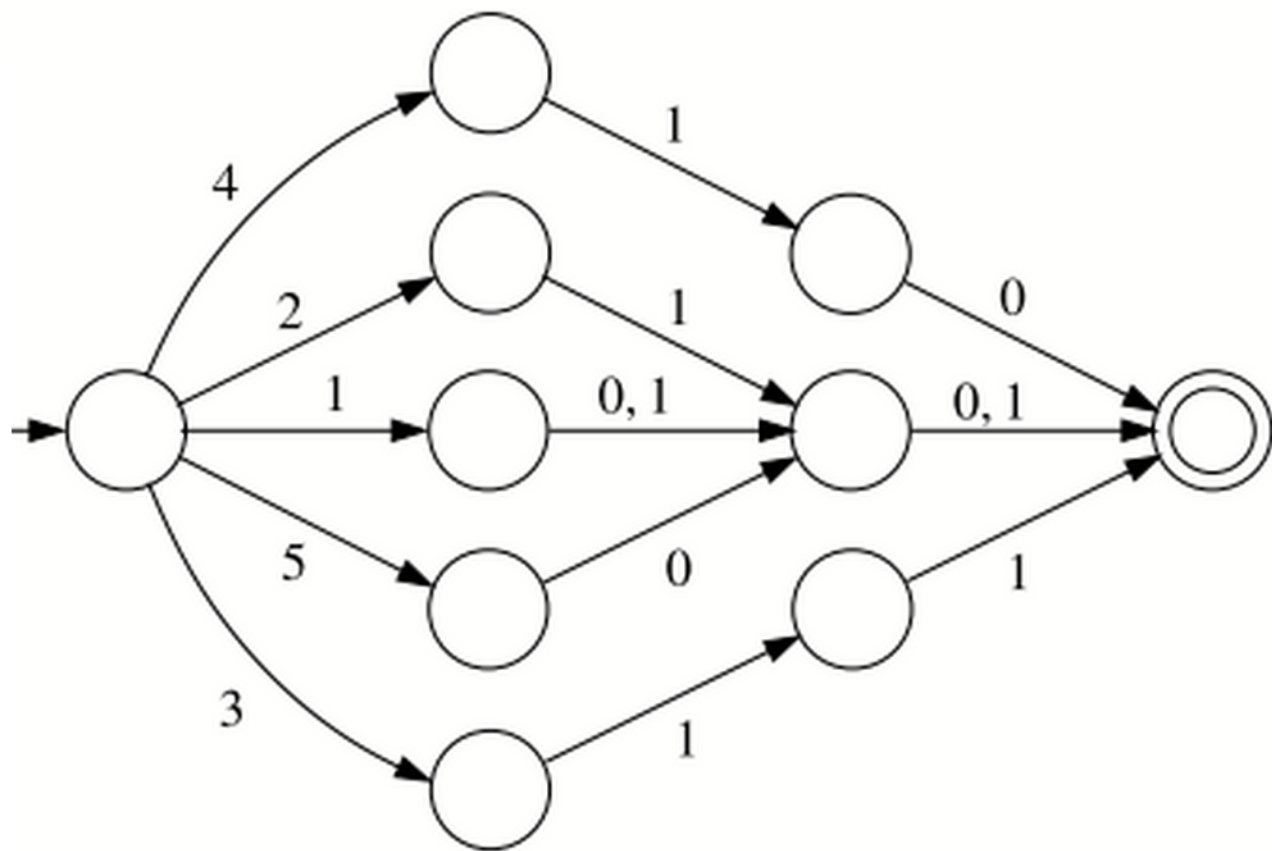
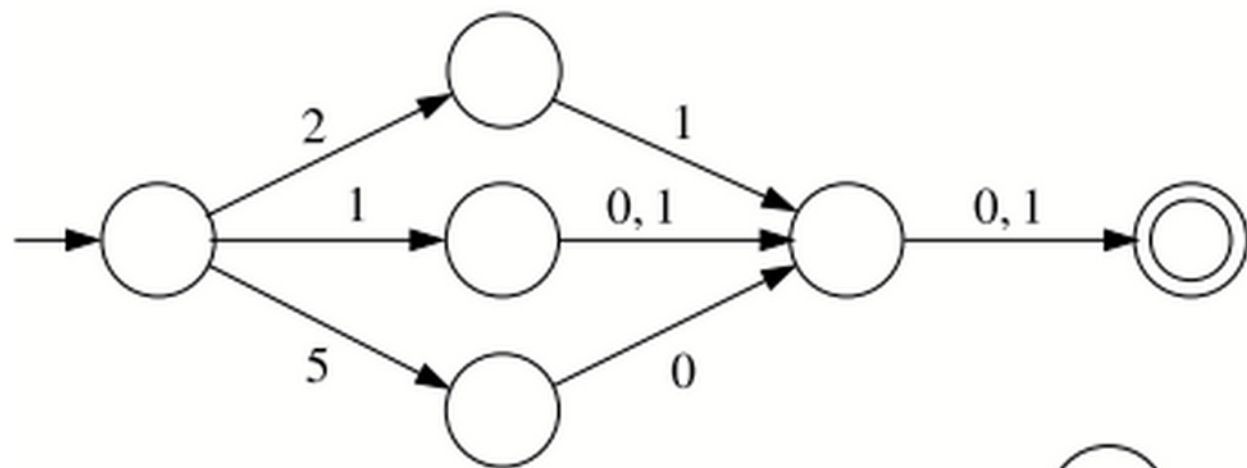
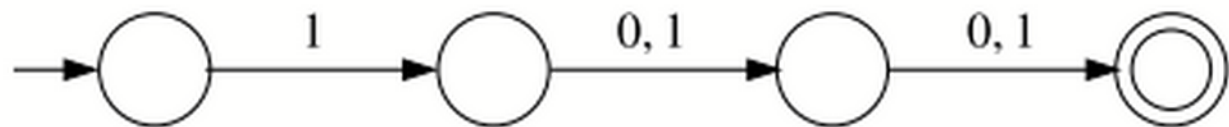
Reach(I, R)

Input: set I of initial configurations; relation R

Output: set of configurations reachable from I

```
1   $OldP \leftarrow \emptyset; P \leftarrow I$   
2  while  $P \neq OldP$  do  
3       $OldP \leftarrow P$   
4       $P \leftarrow \text{Union}(P, \text{Post}(P, S))$   
5  return  $P$ 
```





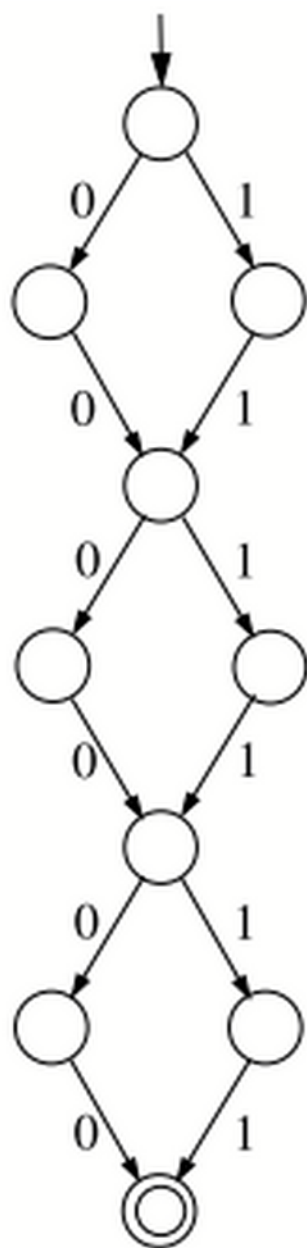
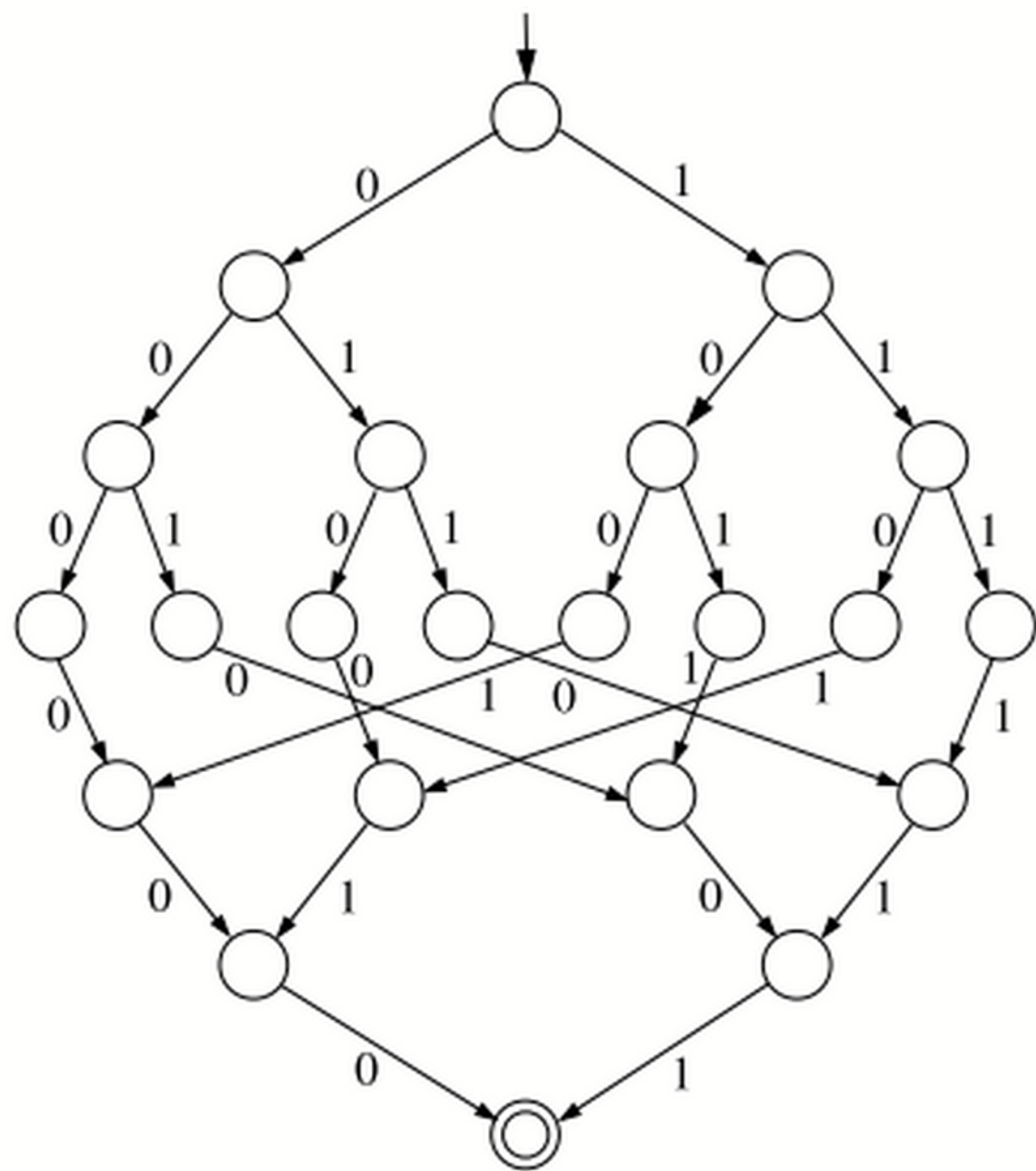
Variable orders

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

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



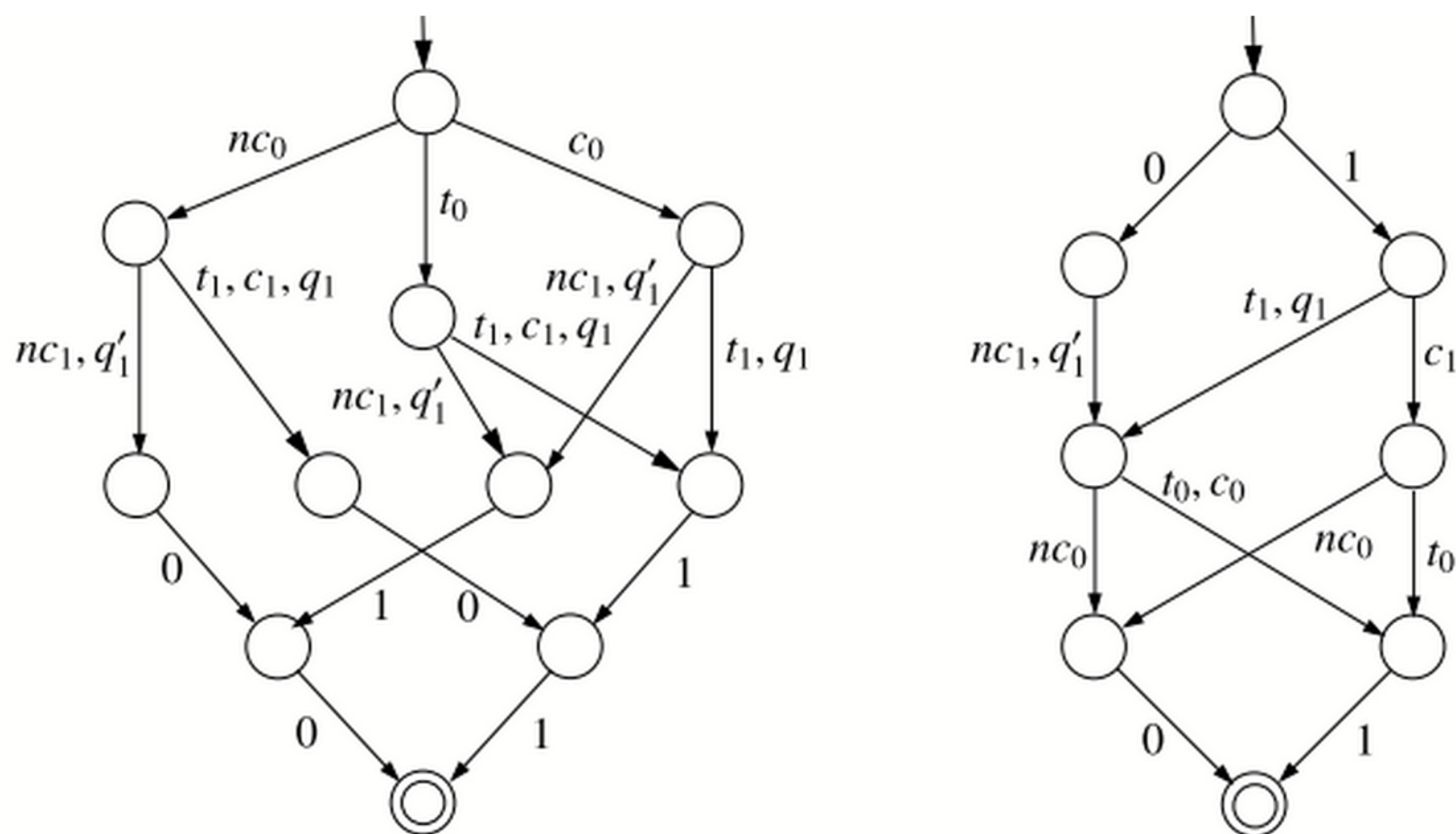


Figure 9.10: Minimal DFAs for the reachable configurations of Lamport's algorithm. On the left a configuration $\langle s_0, s_1, v_0, v_1, q \rangle$ is encoded by the word $s_0 s_1 v_0 v_1 q$, on the right by $v_1 s_1 s_0 v_0$.

Safety and liveness properties

Safety: nothing bad can happen

Liveness: something good eventually happens

More formally:

- safety property: violations are finite executions
- liveness properties: violations are infinite executions