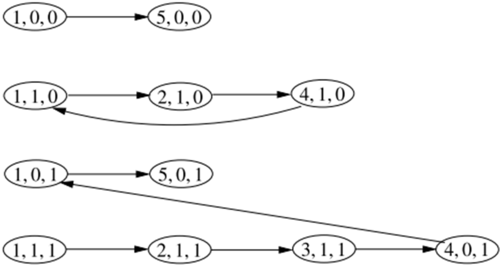
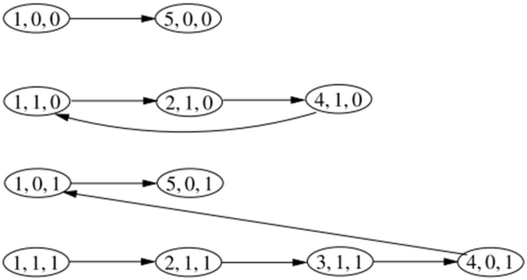
System NFA

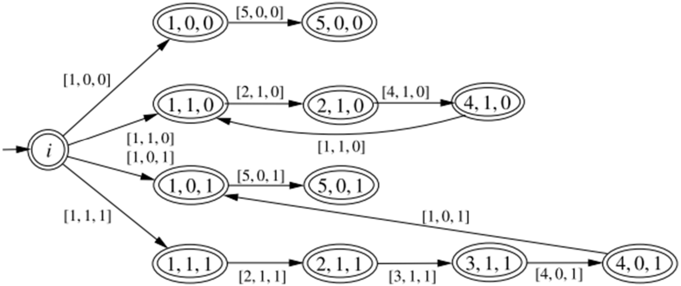
```
while x = 1 do
    if y = 1 then
        x ← 0
    y ← 1 − x
end
```



System NFA



System NFA



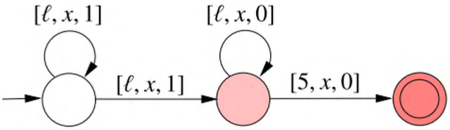
i
1, 0, 0
5, 0, 0
1, 1, 0
2, 1, 0
4, 1, 0
1, 0, 1
5, 0, 1
1, 1, 1
2, 1, 1
3, 1, 1
4, 0, 1
[1, 0, 0]
[5, 0, 0]
[2, 1, 0]
[4, 1, 0]
[1, 1, 0]
[1, 1, 0]
[1, 0, 1]
[5, 0, 1]
[1, 0, 1]
[1, 1, 1]
[2, 1, 1]
[3, 1, 1]
[4, 0, 1]

Property NFA

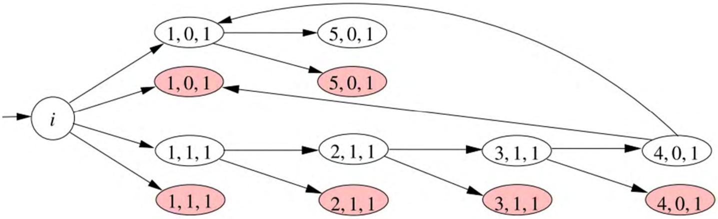
- Is there a full execution such that
  - initially $y = 1$,
  - finally $y = 0$, and
  - y never increases?
- Set of potential executions for this property:

$$[l, x, 1][l, x, 1]^* \; [l, x, 0]^* \; [5, x, 0]$$

- Automaton for this set:



Intersection of the system and property NFAs



- Automaton is empty, and so no execution satisfies the property

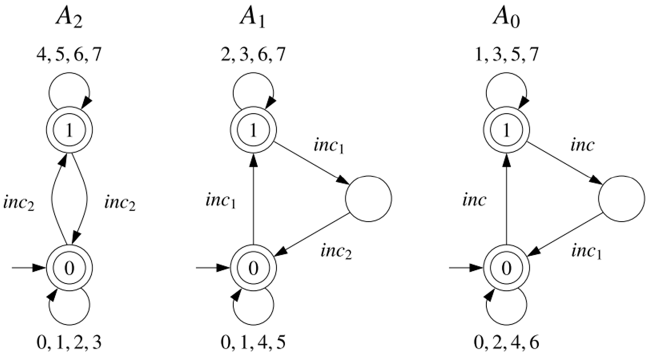
Another property

- Is the assignment $y \leftarrow x - 1$ redundant?
- Potential executions that use the assignment:

$$[l, x, y]^* ([4, x, 0][1, x, 1] + [4, x, 1][1, x, 0]) \; [l, x, y]^*$$

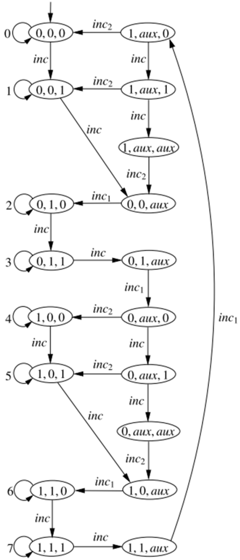
- Therefore: assignment redundant iff none of these potential executions is a real execution of the program.

Networks of automata



- Tuple $\mathcal{A} = \langle A_1, \ldots, A_n \rangle$ of NFAs .
- Each NFA has its own alphabet Σ_i of actions
- Alphabets usually not disjoint!
- A_i participates in action a if $a \in \Sigma_i$.
- A configuration is a tuple $\langle q_1, \ldots, q_n \rangle$ of states, one for each automaton of the network.
- $\langle q_1, \ldots, q_n \rangle$ enables a if every participant in a is in a state from which an a-transition is possible.
- Enabled actions can occur, and their occurrence simultaneously changes the states of their participants. Non-participants stay idle and don‘t change their states.

Configuration graph of the network



$AsyncProduct(A_1, \ldots, A_n)$

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 $A_1 \otimes \cdots \otimes A_n = (Q, \Sigma, \delta, q_0, F)$

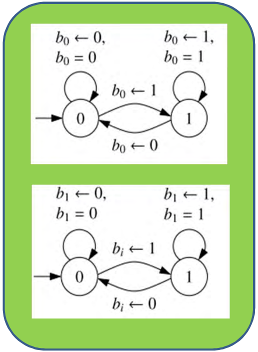
$Q, \delta, F \leftarrow \emptyset$
$q_0 \leftarrow [q_{01}, \ldots, q_{0n}]$
$W \leftarrow \{[q_{01}, \ldots, q_{0n}]\}$
while $W \neq \emptyset$ **do**
  pick $[q_1, \ldots, q_n]$ **from** W
  add $[q_1, \ldots, q_n]$ **to** Q
  add $[q_1, \ldots, q_n]$ **to** F
  for all $a \in \Sigma_1 \cup \ldots \cup \Sigma_n$ **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 $(Q, \Sigma, \delta, q_0, F)$

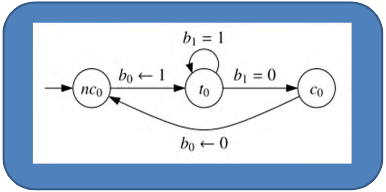
Concurrent programs as networks of automata: Lamport's 1-bit algorithm (JACM86)

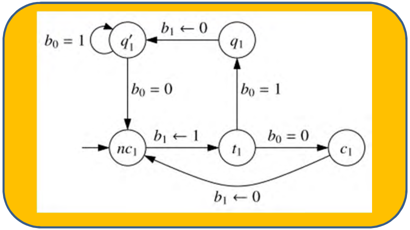
Shared variables: b[1], ..., b[n] ∈ {0,1}, initially 0
Process i ∈ {1, ...,n}

```
repeat forever
      noncritical section
   T: b[i]:=1
      for j ∈ {1, ...,i-1}
         if b[j]=1 then  b[i]:=0
                         await ¬b[j]
                         goto T
      for j ∈ {i+1, ...,N}  await ¬b[j]
      critical section
      b[i]:=0
```

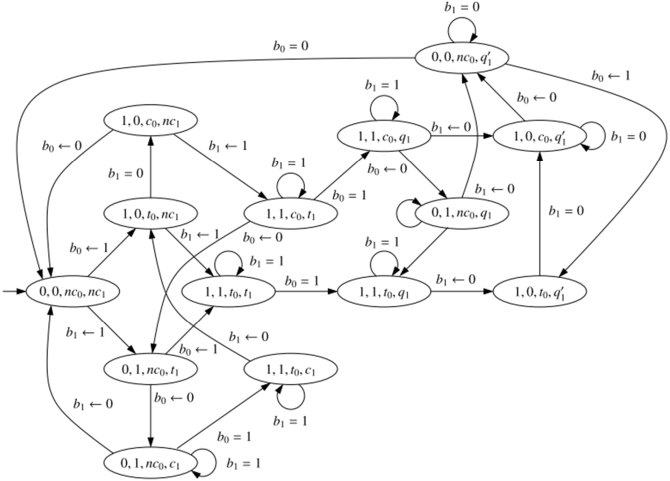
Network for the two-process case







Asynchronous product



Checking properties of the algorithm

- Deadlock freedom: every configuration has at least one successor.
- Mutual exclusion: no configuration of the form $[b_0, b_1, c_0, c_1]$ is reachable
- Bounded overtaking (for process 0): after process 0 signals interest in accessing the critical section, process 1 can enter the critical section at most one before process 0 enters.
  - Let NC_i, T_i, C_i be the configurations in which process i is non-critical, trying, or critical
  - Set of potential executions violating the property:

$$\Sigma^* \; T_0 \; (\Sigma \setminus C_0)^* \; C_1 \; (\Sigma \setminus C_0)^* \; NC_1 \; (\Sigma \setminus C_0)^* \; C_1 \; \Sigma^*$$

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

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** $q \in F_v$ **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

- In sequential programs, the number of reachable configurations grows exponentially in the number of variables.
- Proposition: The following problem is PSPACE-complete.
  - Given: a boolean program π (program with only boolean variables), and a NFA A_V recognizing a set of potential executions
  - Decide: Is $E_\pi \cap L(A_V)$ empty?

The state-explosion problem

- In concurrent programs, the number of reachable configurations also grows exponentially in the number of components.
- Proposition: The following problem is PSPACE-complete.
  - Given: a network of automata $\mathcal{A} = \langle A_1, \ldots, A_n \rangle$ and a NFA A_V recognizing a set of potential executions of $\mathcal{A}$
  - Decide: Is $L(A_1 \otimes \cdots \otimes A_n \otimes A_V) = \emptyset$?