(Place/Transition) Petri Nets

Hans.Vangheluwe@uantwerpen.be

Universiteit Antwerpen
Ansymo Antwerp Systems & Software Modelling University of Antwerp
Nexor Cyber-Physical Systems University of Antwerp
FLANDERS MAKE MANUFACTURING INNOVATION NETWORK
MPM4CPS
cost

Petri Nets

1. Finite State Automata
2. Petri net notation and definition (no dynamics)
3. Introducing State: Petri net marking
4. Petri net dynamics
5. Capacity Constrained Petri nets
6. Petri net models for ...
   - FSA
   - Nondeterminism
   - Data Flow Computation
   - Communication Protocols

7. Queueing Systems

8. Petri nets vs. State Automata

9. Analysis of Petri nets
   - Boundedness
   - Liveness and Deadlock
   - State Reachability
   - State Coverability
   - Persistence
   - Language Recognition

10. The Coverability Tree

11. Extensions: colour, time, ...

12. LTL and CTL

Finite State Automaton

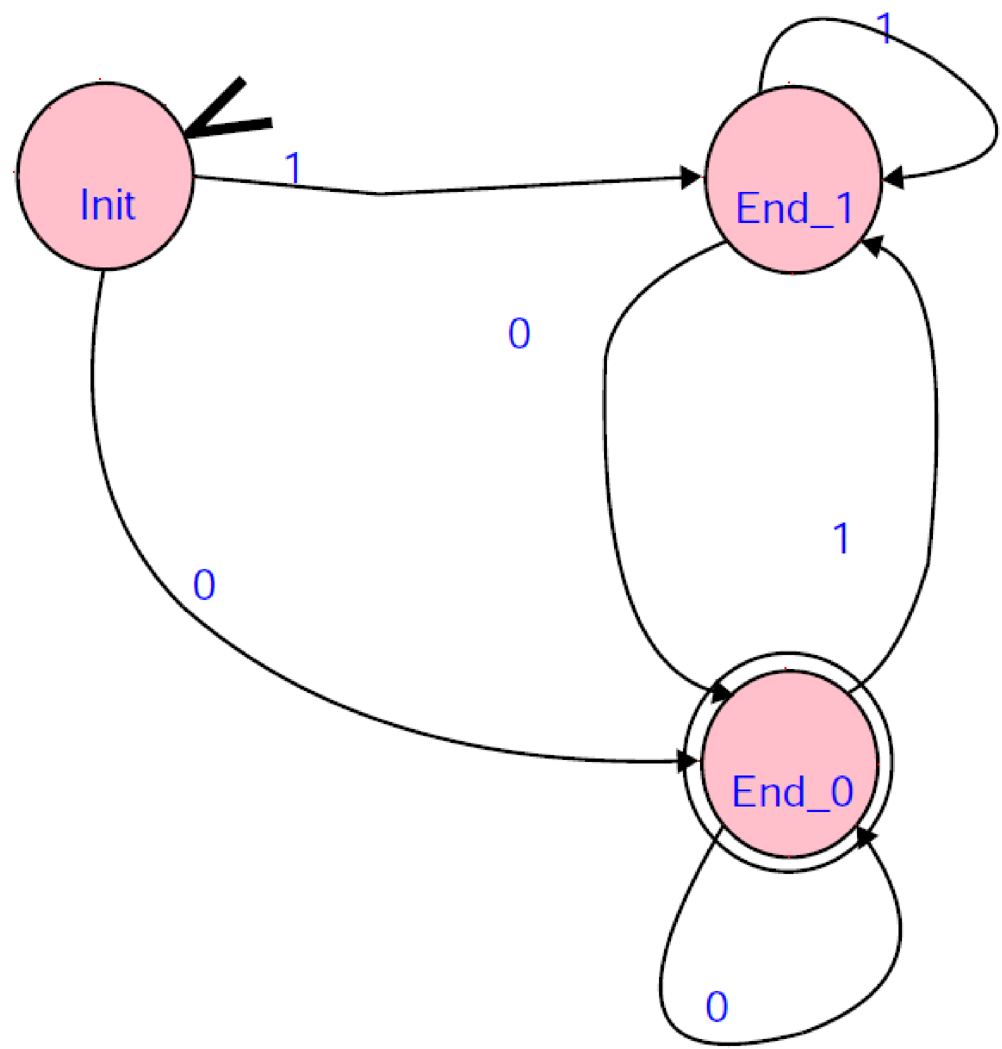
$$(E, X, f, x_0, F)$$

- E is a finite alphabet
- X is a finite state set
- f is a state transition function,
  $f : X \times E \to X$
- x_0 is an initial state, $x_0 \in X$
- F is the set of final states

Dynamics (x' is next state):

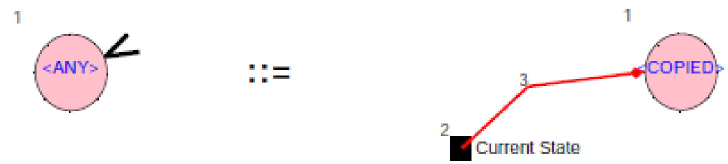
$$x' = f(x, e)$$

FSA graphical/visual notation: State Transition Diagram

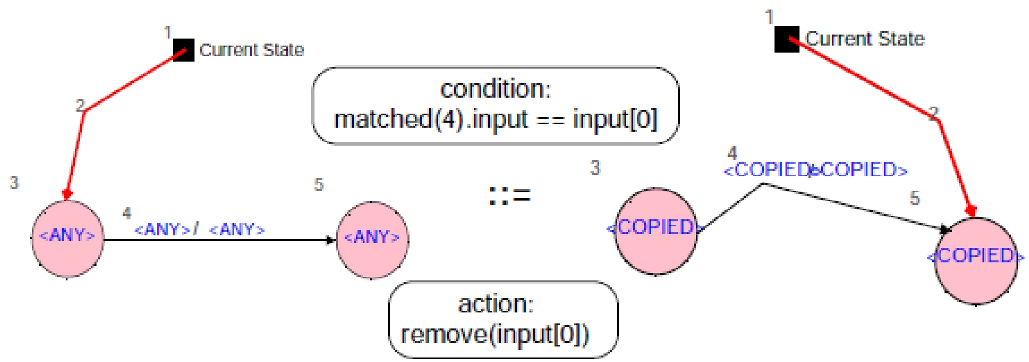


FSA Operational Semantics

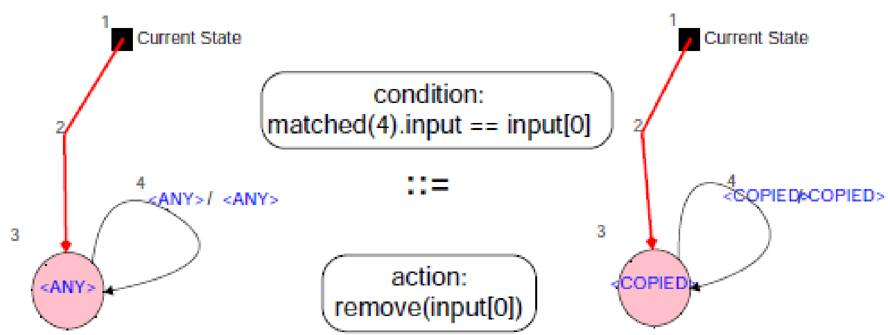
Rule 1 (priority 3) — **Locate Initial Current State**



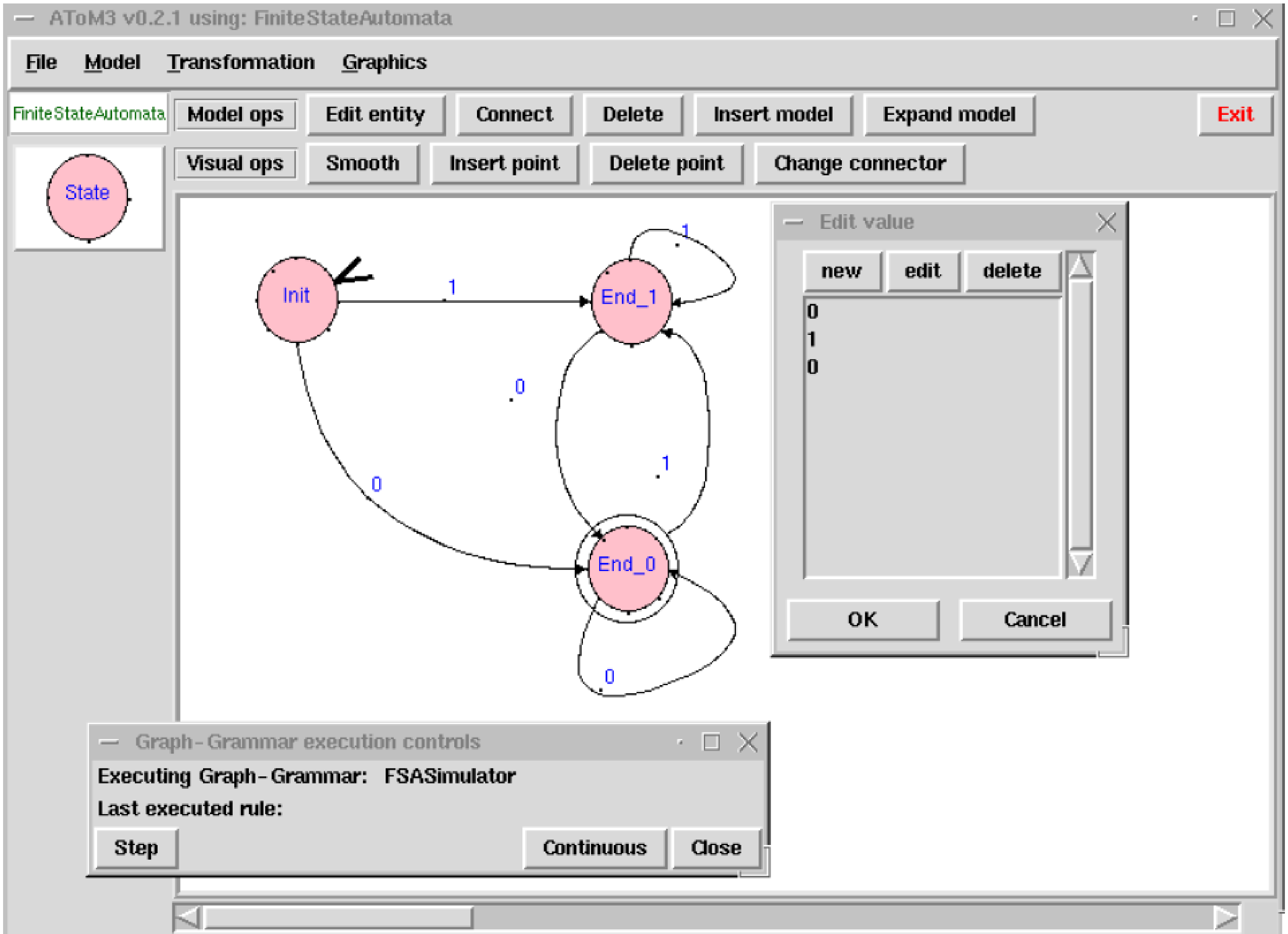
Rule 2 (priority 1) — **State Transition**

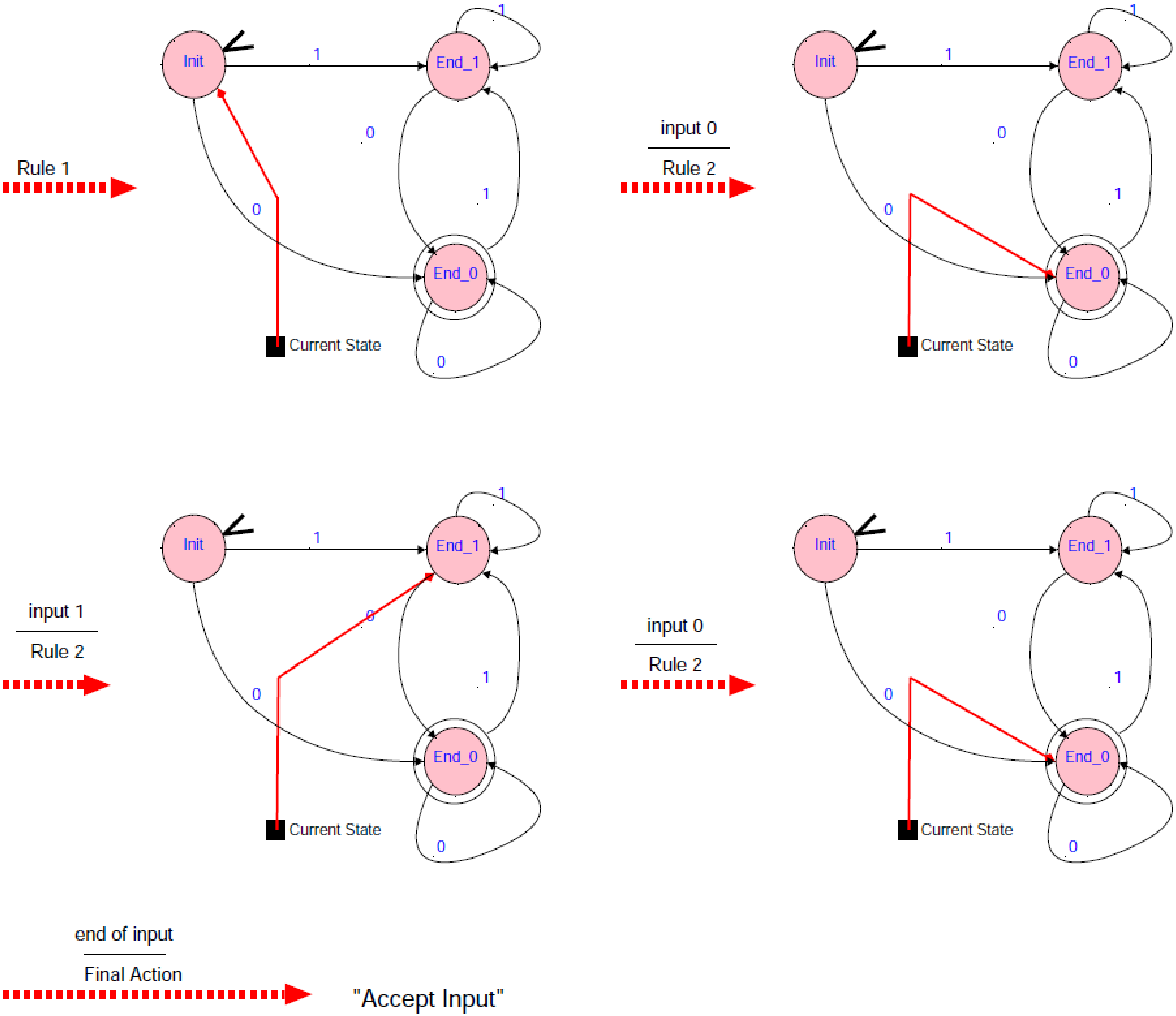


Rule 3 (priority 2) — **Local State Transition**



Simulation steps




Rule 1
Init
End_1
End_0
Current State
input 0
Rule 2
input 1
Rule 2
input 0
Rule 2
end of input
Final Action
"Accept Input"

State Automaton

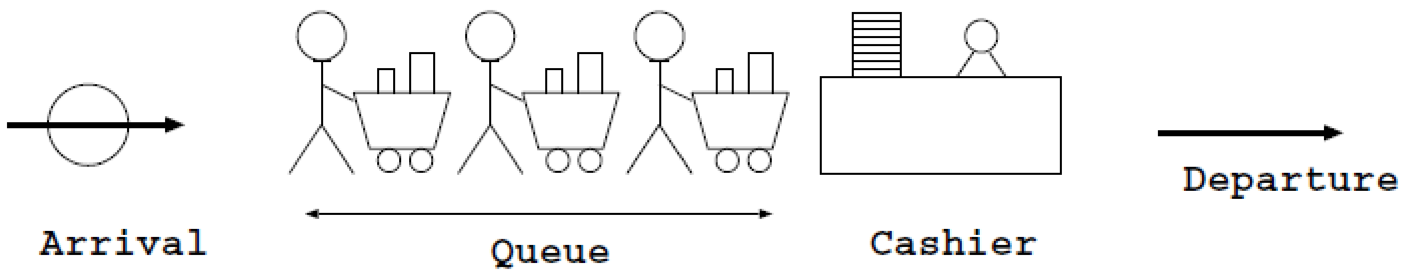
$$(E, X, \Gamma, f, x_0)$$

- E is a countable event set
- X is a countable state space
- $\Gamma(x)$ is the set of feasible or enabled events
  $x \in X, \Gamma(x) \subseteq E$
- f is a state transition function,
  $f : X \times E \rightarrow X$, only defined for $e \in \Gamma(x)$
- x_0 is an initial state, $x_0 \in X$

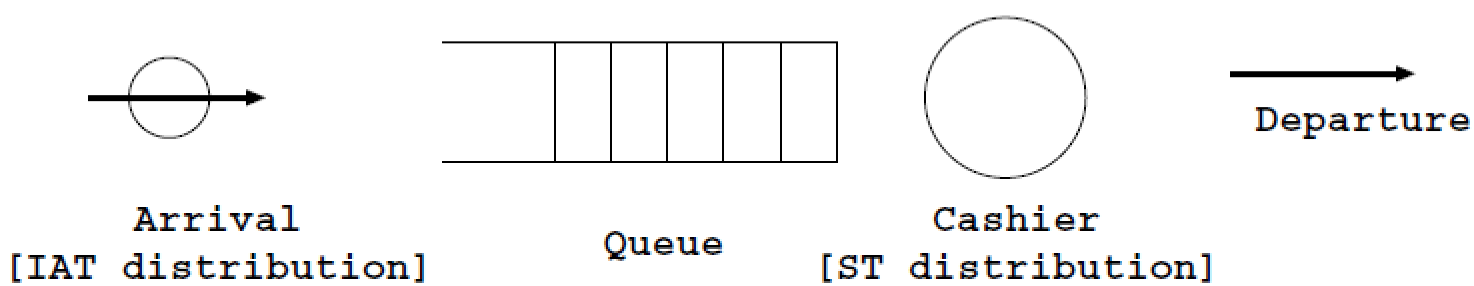
$$(E, X, \Gamma, f)$$

omits x_0 and describes a class of State Automata.

State Automata for Queueing Systems

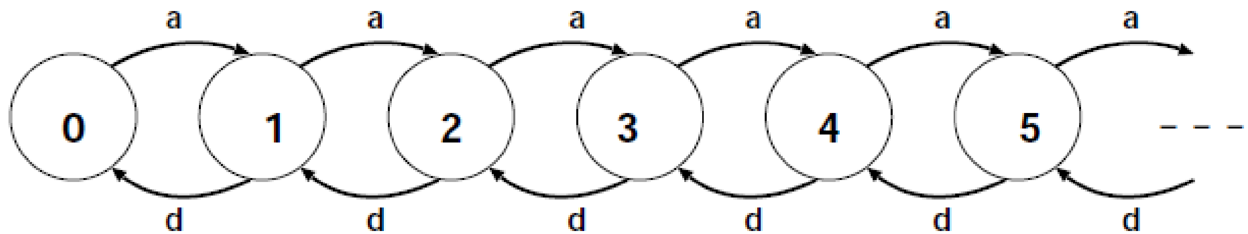


Physical View



Abstract View

State Automata for Queueing Systems: customer centered



$$E = \{a, d\}$$

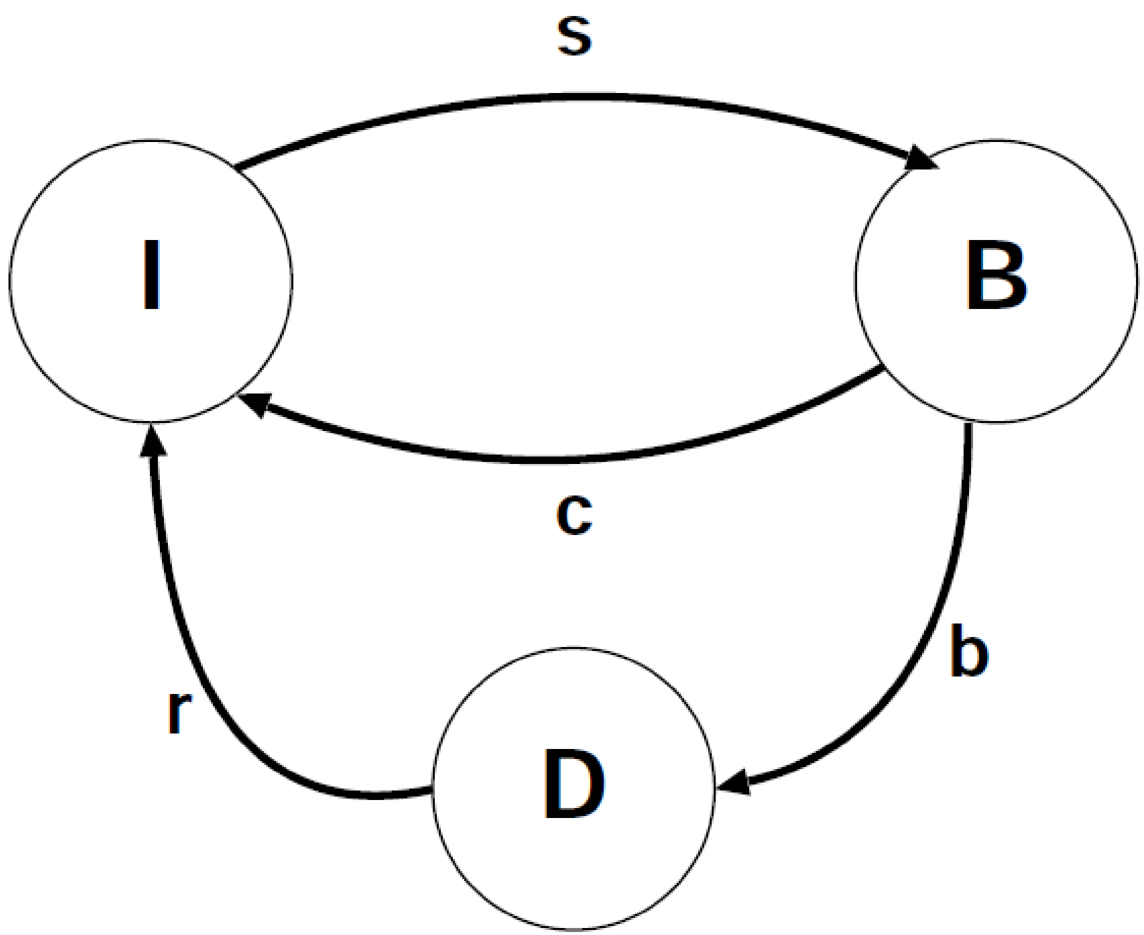
$$X = \{0, 1, 2, \ldots\}$$

$$\Gamma(x) = \{a, d\}, \forall x > 0; \Gamma(0) = \{a\}$$

$$f(x, a) = x + 1, \forall x \geq 0$$

$$f(x, d) = x - 1, \forall x > 0$$

State Automata for Queueing Systems: server centered (with breakdown)



State Automata for Queueing Systems: server centered (with breakdown)

$$E = \{s, c, b, r\}$$

Events: s denotes service starts, c denotes service completes, b denotes breakdown, r denotes repair.

$$X = \{I, B, D\}$$

State: I denotes idle, B denotes busy, D denotes broken down.

$$\Gamma(I) = \{s\}, \Gamma(B) = \{c, b\}, \Gamma(D) = \{r\}$$

$$f(I, s) = B, f(B, c) = I, f(B, b) = D, f(D, r) = I$$

Limitiations/extensions of State Automata

- Adding time ?
- Hierarchical modelling ?
- Concurrency by means of $\times$
- States are represented explicitly
- Specifying control logic, synchronisation ?

Petri nets

- Formalism similar to FSA
- Graphical/Visual notation
- C.A. Petri 1960s
- Additions to FSA:
  - Explicitly (graphically/visually) represent when event is enabled
    $\rightarrow$ describe control logic
  - Elegant notation of concurrency
  - Express non-determinism

Petri net notation and definition (no dynamics)

$$(P, T, A, w)$$

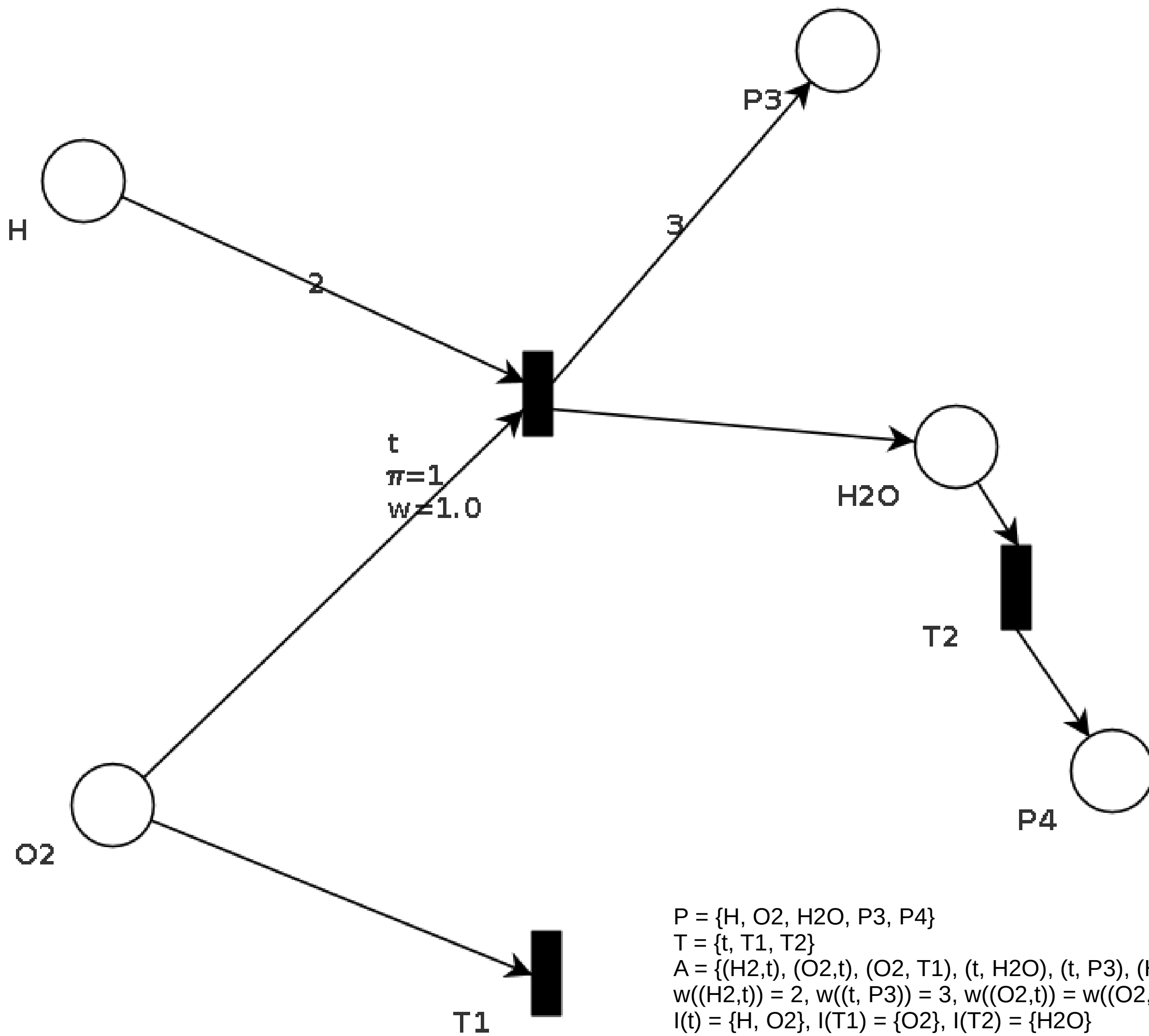
- $P = \{p_1, p_2, \ldots\}$ is a finite set of *places*
- $T = \{t_1, t_2, \ldots\}$ is a finite set of *transitions*
- $A \subseteq (P \times T) \cup (T \times P)$ is a set of *arcs*
- $w : A \rightarrow \mathbb{N}$ is a *weight function*

Note: no need for *countable* P and T.

Derived Entities

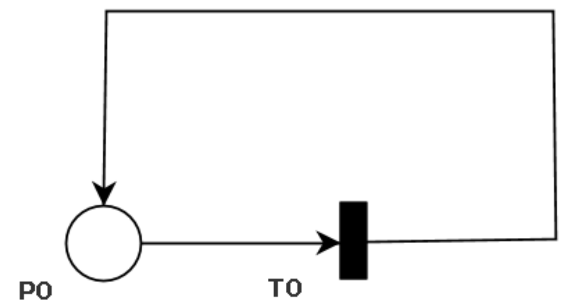
- $I(t_j) = \{p_i : (p_i, t_j) \in A\}$ set of *input places* to transition t_j ($\equiv$ conditions for transition)
- $O(t_j) = \{p_i : (t_j, p_i) \in A\}$ set of *output places* from transition t_j ($\equiv$ affected by transition)
- Transitions $\equiv$ events
- similarly: input- and output-transitions for p_i
- graphical/visual representation: *Petri net graph* (multigraph)

Example Petri Net



P = {H, O2, H2O, P3, P4}
T = {t, T1, T2}
A = {(H2,t), (O2,t), (O2, T1), (t, H2O), (t, P3), (H2O, T2), (T2, P4)}
w((H2,t)) = 2, w((t, P3)) = 3, w((O2,t)) = w((O2, T1)) = w((t, H2O)) = w((H2O, T2)) = w((T2, P4)) = 1
I(t) = {H, O2}, I(T1) = {O2}, I(T2) = {H2O}
O(t) = {P3, H2O}, O(T1) = {}, O(T2) = {P4}

Pure Petri net



- No self-loops:

$$\nexists p_i \in P, t_j \in T : (p_i, t_j) \in A, (t_j, p_i) \in A$$

- Can convert impure to pure Petri net

Introducing State: Petri net Markings

- Conditions met ? Use *tokens* in places
- Token assignment $\equiv$ *marking* x

$$x : P \rightarrow \mathbb{N}$$

- A marked Petri net

$$(P, T, A, w, x_0)$$

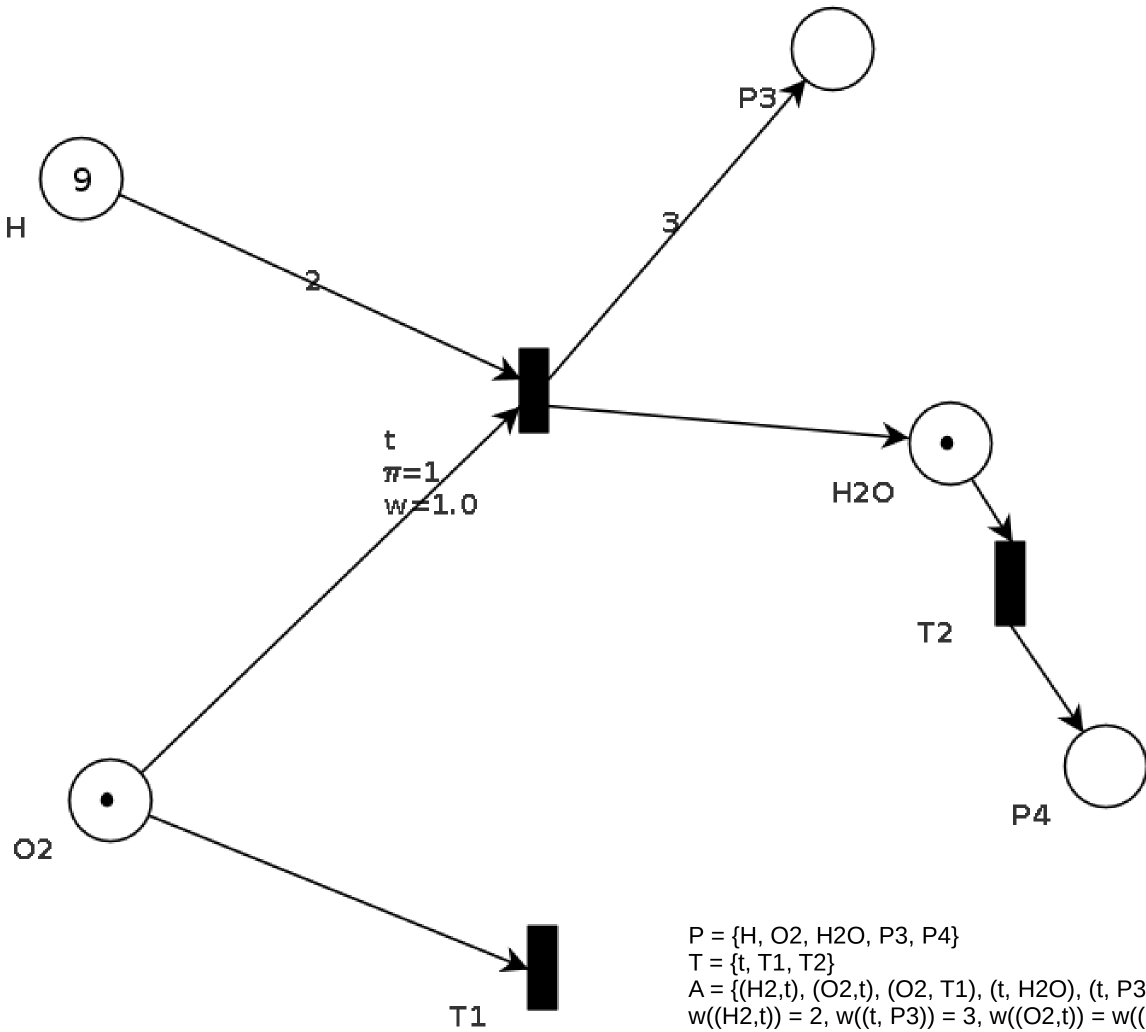
  x_0 is the *initial marking*

- The *state* $\mathbf{x}$ of a marked Petri net

$$\mathbf{x} = [x(p_1), x(p_2), \ldots, x(p_n)]$$

  Number of tokens need not be bounded (cfr. State Automata states).

Example Marked Petri Net



P = {H, O2, H2O, P3, P4}
T = {t, T1, T2}
A = {(H2,t), (O2,t), (O2, T1), (t, H2O), (t, P3), (H2O, T2), (T2, P4)}
w((H2,t)) = 2, w((t, P3)) = 3, w((O2,t)) = w((O2, T1)) = w((t, H2O)) = w((H2O, T2)) = w((T2, P4)) = 1
I(t) = {H, O2}, I(T1) = {O2}, I(T2) = {H2O}
O(t) = {P3, H2O}, O(T1) = {}, O(T2) = {P4}
x = [9, 1, 1, 0, 0] corresponding to places [H, O2, H2O, P3, P4]

State Space of Marked Petri net

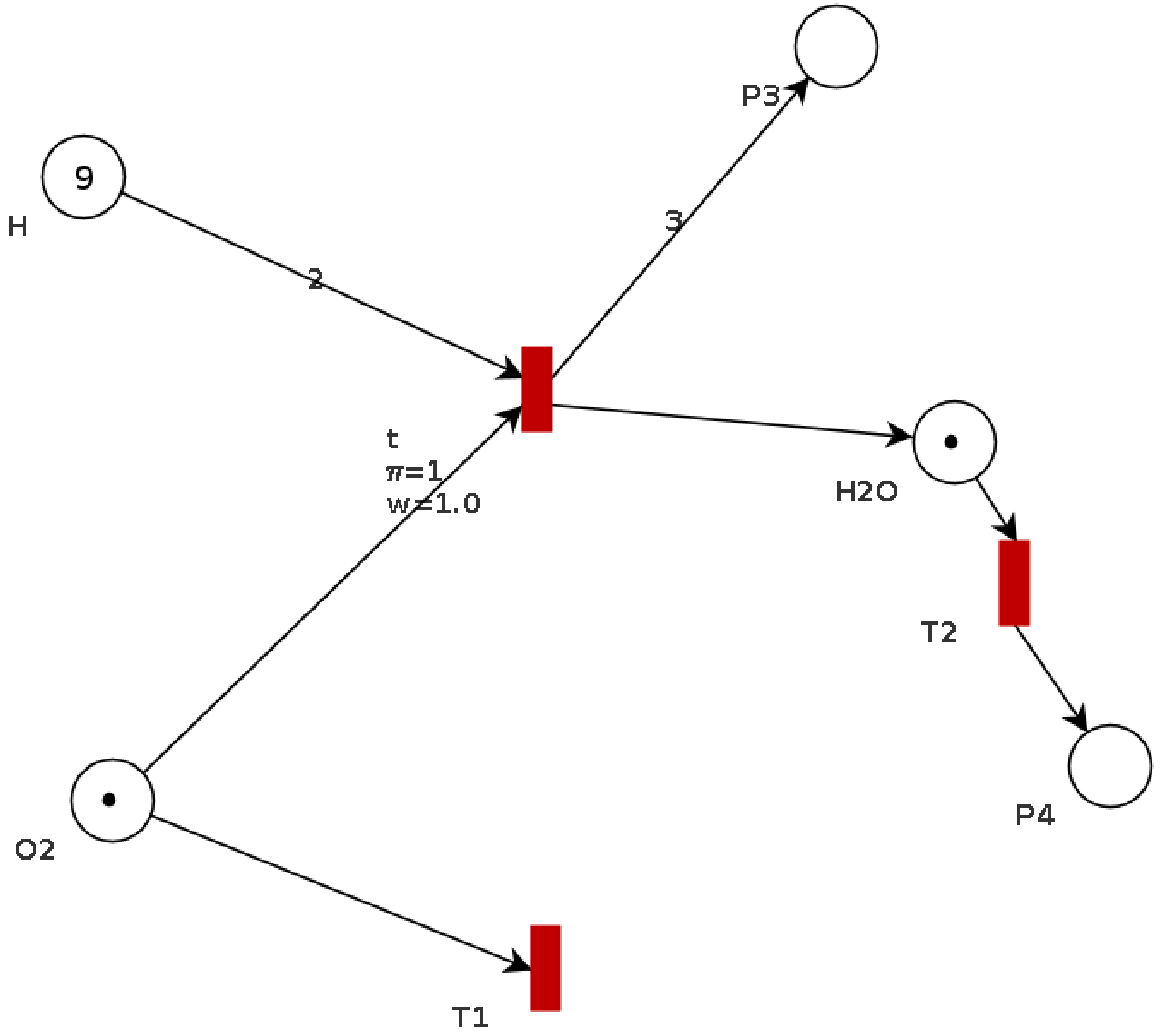
- All n-dimensional vectors of nonnegative integer markings

$$X = \mathbb{N}^n$$

- Transition $t_j \in T$ is *enabled* if

$$x(p_i) \geq w(p_i, t_j), \forall p_i \in I(t_j)$$

Example Marked Petri Net
Enabled transitions in red



Petri Net Dynamics

State Transition Function f of marked Petri net (P, T, A, w, x_0)

$$f : \mathbb{N}^n \times T \to \mathbb{N}^n$$

is defined for transition $t_j \in T$ if and only if

$$x(p_i) \geq w(p_i, t_j), \forall p_i \in I(t_j)$$

If $f(\mathbf{x}, t_j)$ is defined, set $\mathbf{x}' = f(\mathbf{x}, t_j)$ where

$$x'(p_i) = x(p_i) - w(p_i, t_j) + w(t_j, p_i)$$

- State transition function f based on *structure* of Petri net
- Number of tokens *need not be conserved* (but can)

Algebraic Description of Dynamics

- Firing vector $\mathbf{u}$: transition j firing

$$\mathbf{u} = [0, 0, \ldots, 1, 0, \ldots, 0]$$

- Incidence matrix $\mathbf{A}$:

$$a_{ji} = w(t_j, p_i) - w(p_i, t_j)$$

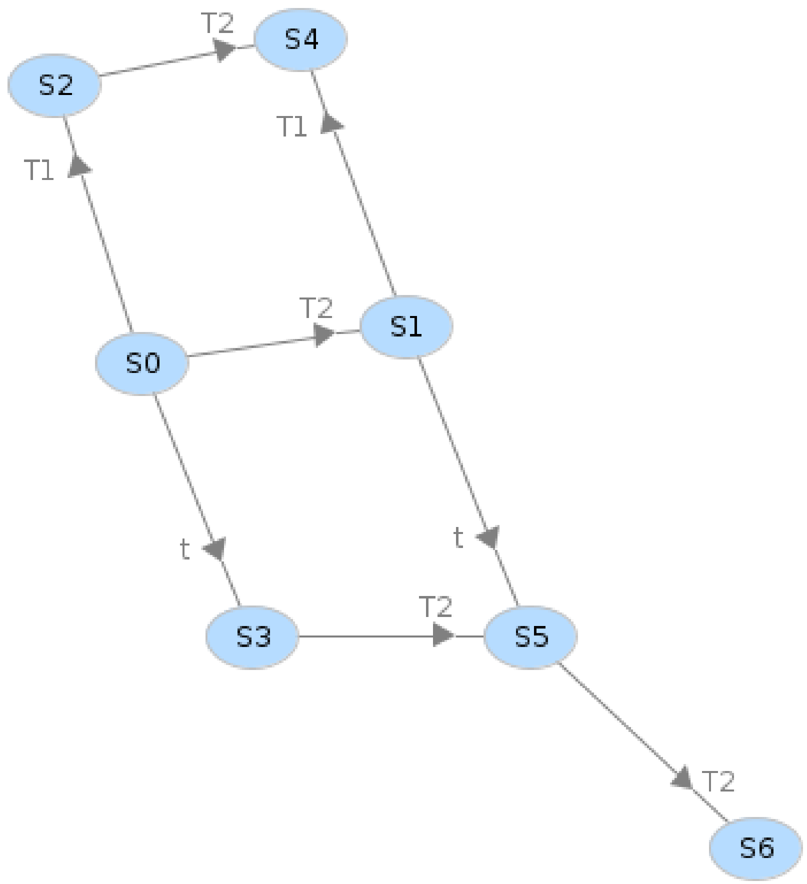
- State Equation

$$\mathbf{x}' = \mathbf{x} + \mathbf{u}\mathbf{A}$$

Semantics

- *sequential* vs. *parallel*
- Handle nondeterminism:
  1. User choice
  2. Priorities
  3. Probabilities (Monte Carlo)
  4. Reachability Graph (enumerate all choices)

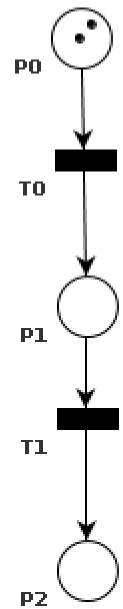
Example Marked Petri Net



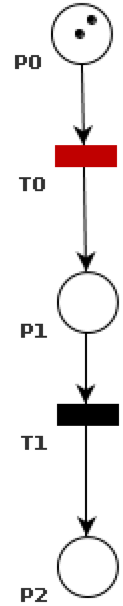
S0 = [9, 1, 1, 0, 0]
S1 = [9, 1, 0, 0, 1]
S2 = [9, 0, 1, 0, 0]
S3 = [7, 0, 2, 3, 0]
S4 = [9, 0, 0, 0, 1]
S5 = [7, 0, 1, 3, 1]
S6 = [7, 0, 0, 3, 2]

Marking corresponds to [H, O2, H2O, P3, P4]

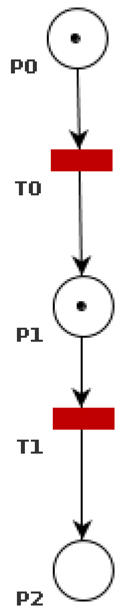
Pattern: sequence



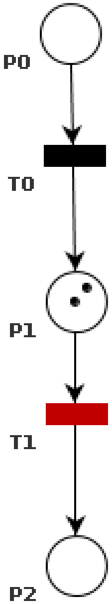
Pattern: sequence

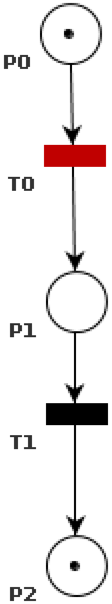


Pattern: sequence

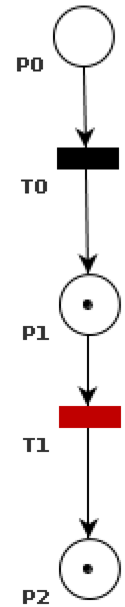


Pattern: sequence

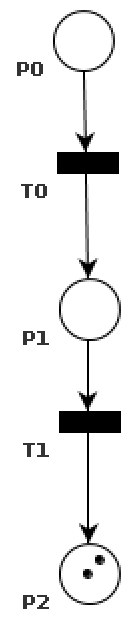




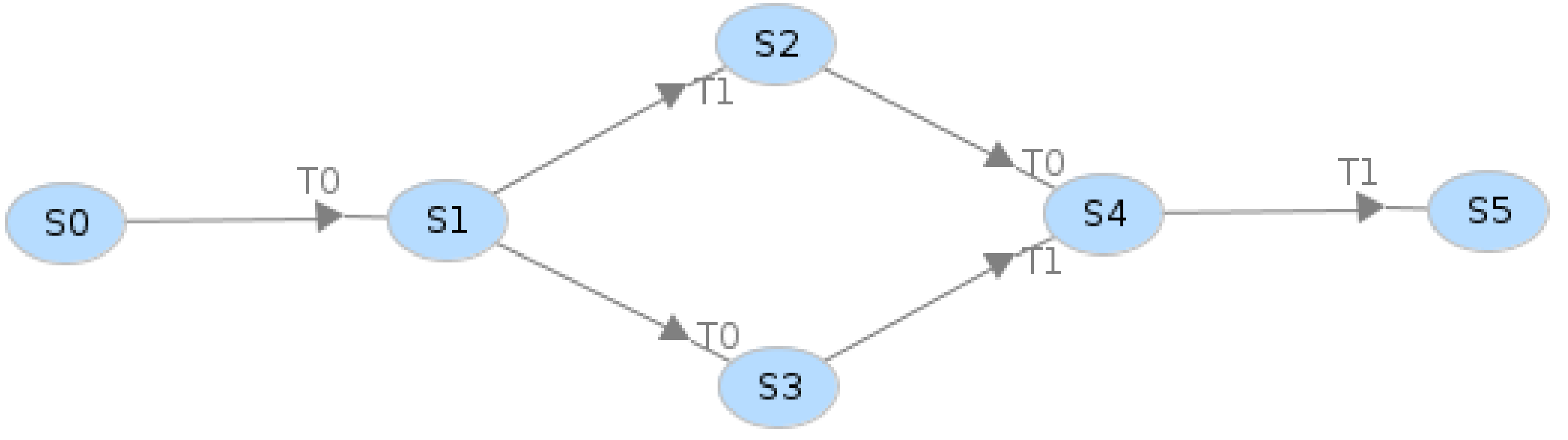
Pattern: sequence



Pattern: sequence

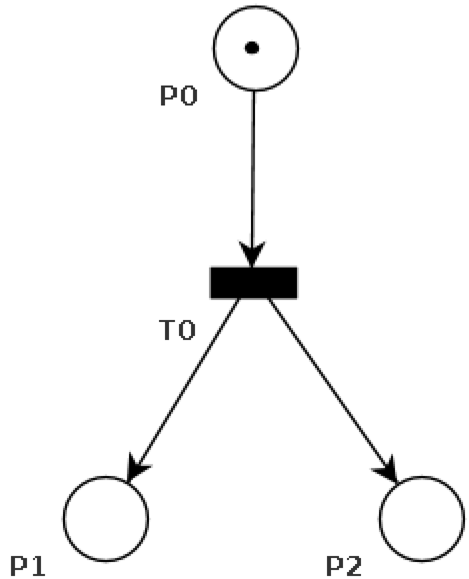


Pattern: sequence

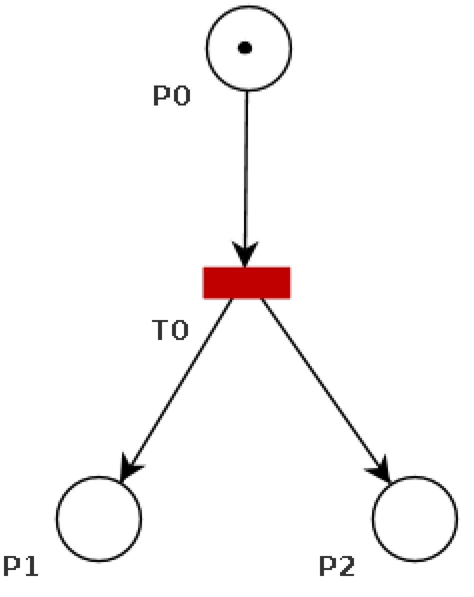


S0 = [2, 0, 0]
S1 = [1, 1, 0]
S2 = [1, 0, 1]
S3 = [0, 2, 0]
S4 = [0, 1, 1]
S5 = [0, 0, 2]

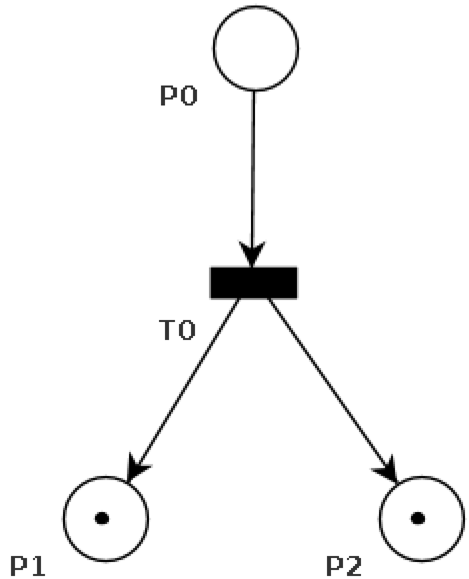
Pattern: split



Pattern: split



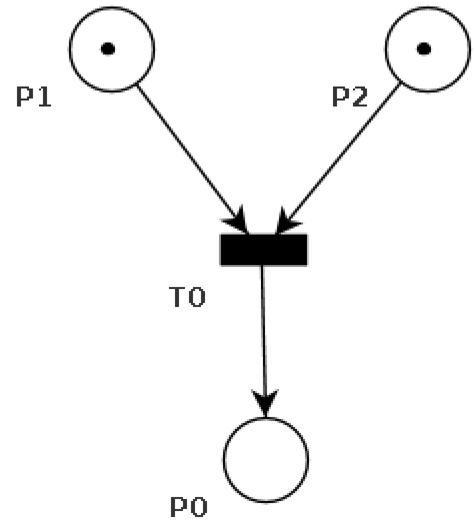
Pattern: split



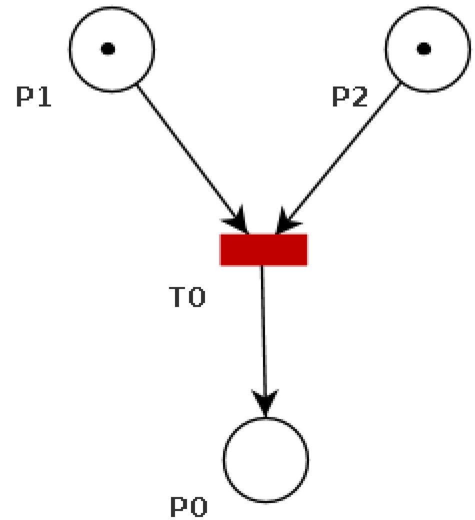
Pattern: split

S0 = [1, 0, 0]
S1 = [0, 1, 1]

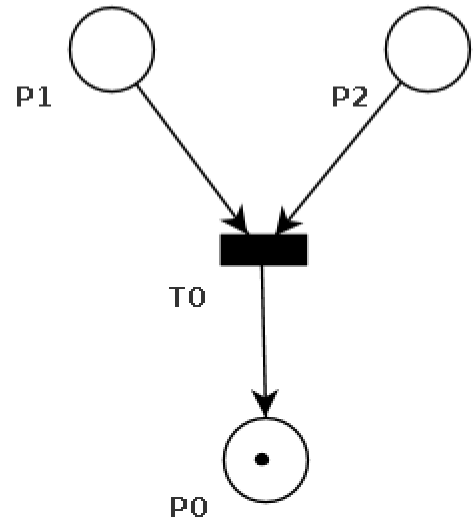
Pattern: join



Pattern: join



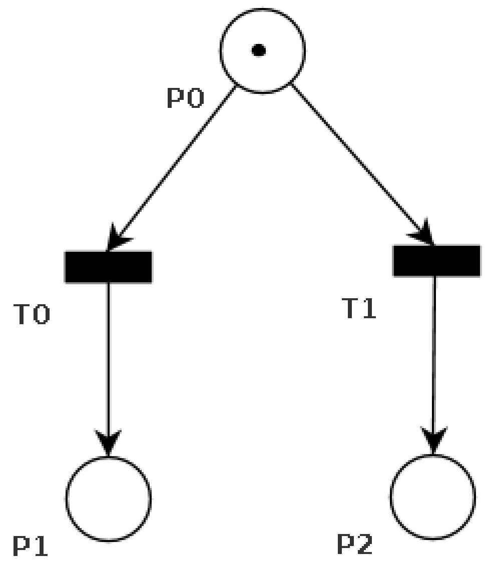
Pattern: join



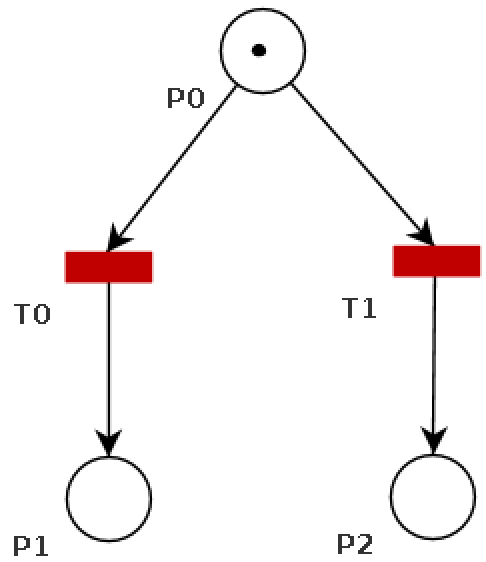
Pattern: join

S0 = [1, 1, 0]
S1 = [0, 0, 1]

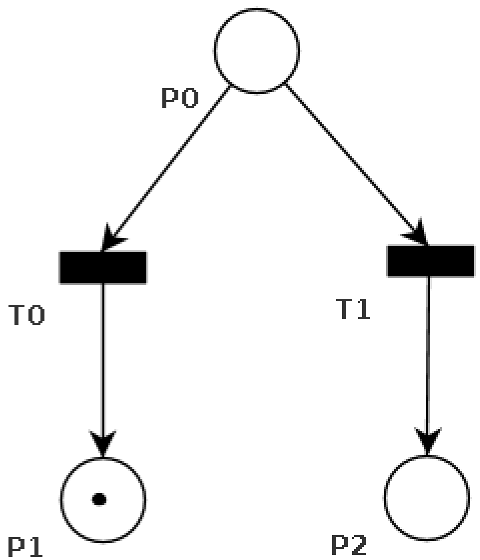
Pattern: conflict, choice, decision

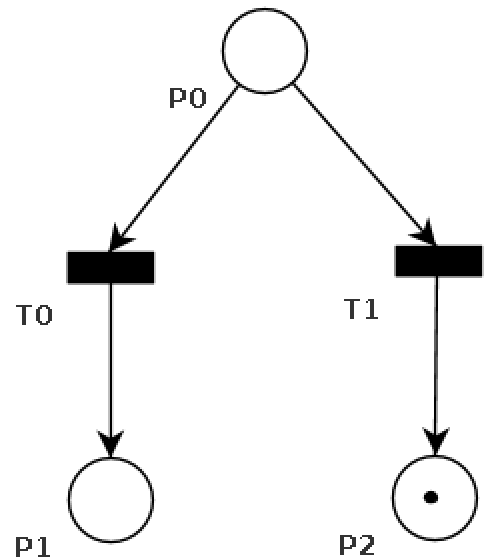


Pattern: conflict, choice, decision



Pattern: conflict, choice, decision

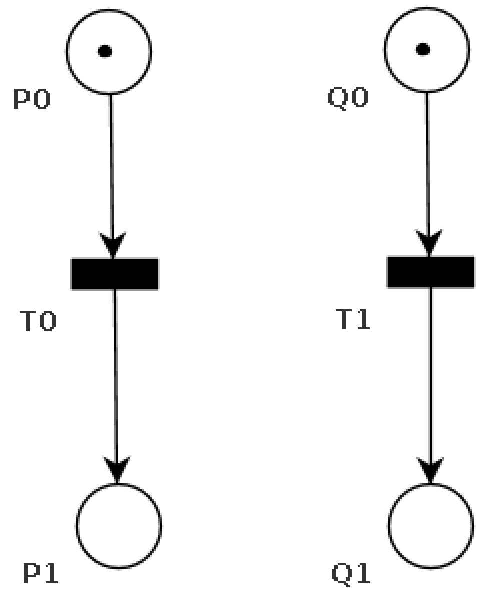




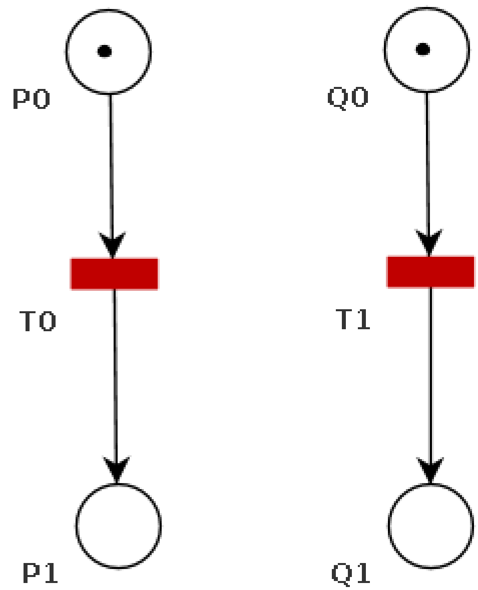
Pattern: conflict, choice, decision

S0 = [1, 0, 0]
S1 = [0, 0, 1]
S2 = [0, 1, 0]

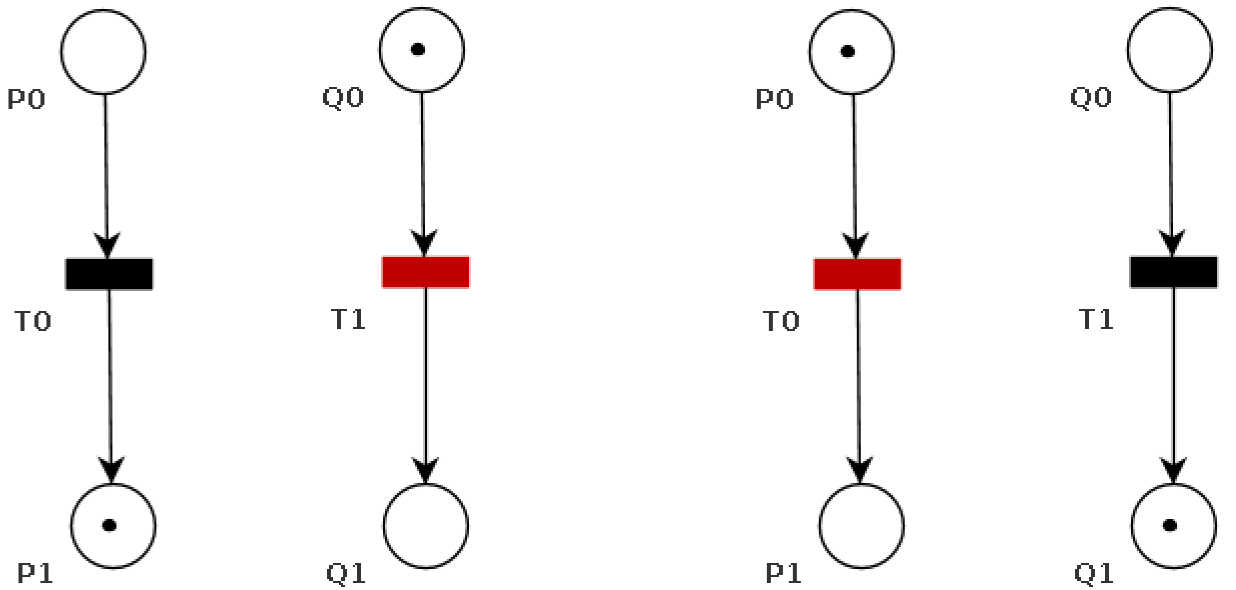
parallel indepencence, confluence



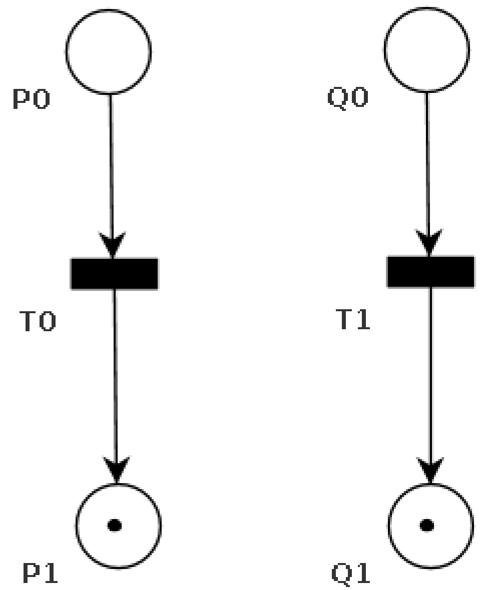
parallel indepencence, confluence



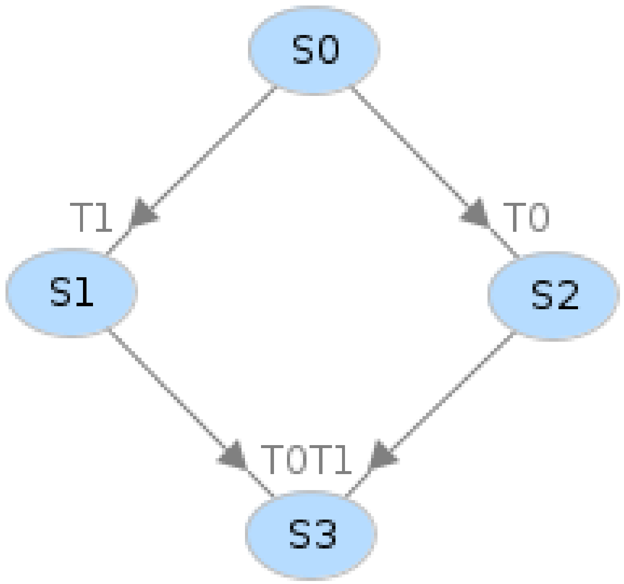
parallel indepencence, confluence



parallel indepencence, confluence

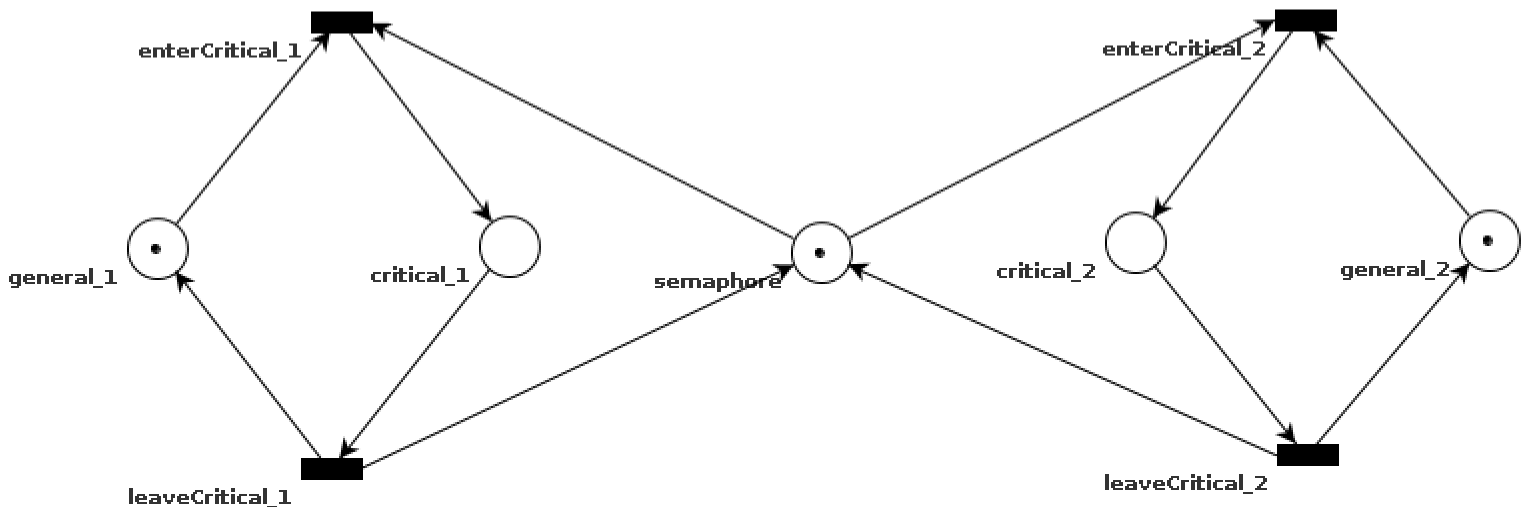


parallel indepencence, confluence
“diamond” pattern

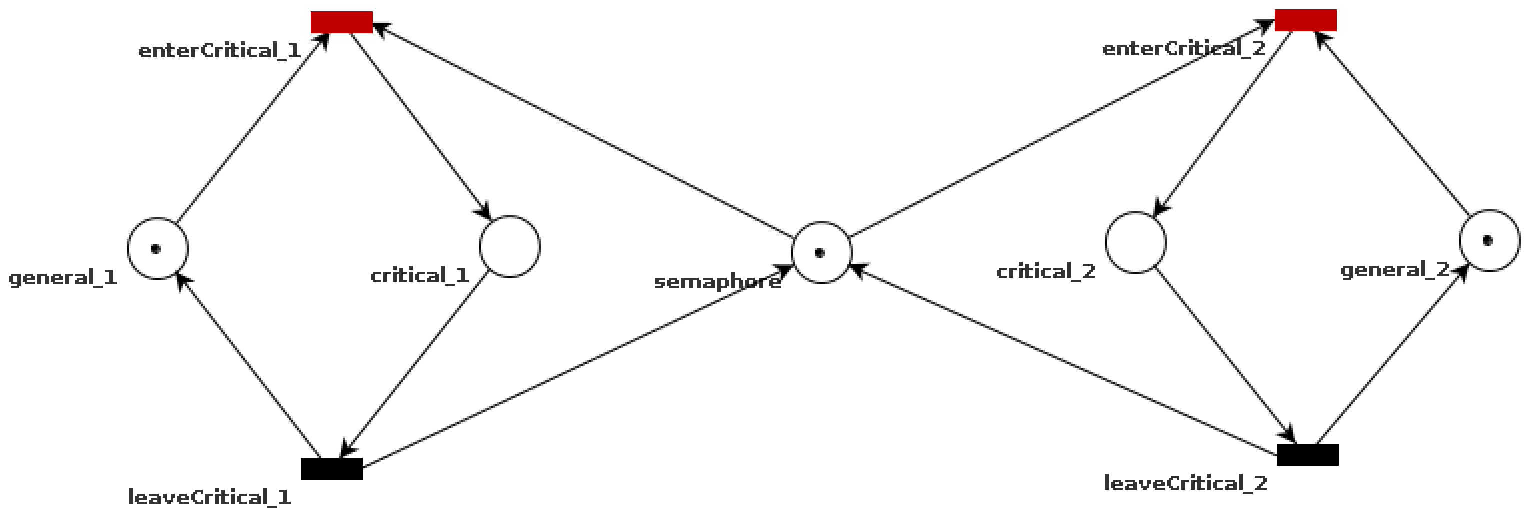


S0 = [1, 0, 1, 0]
S1 = [0, 1, 1, 0]
S2 = [1, 0, 0, 1]
S3 = [0, 1, 0, 1]

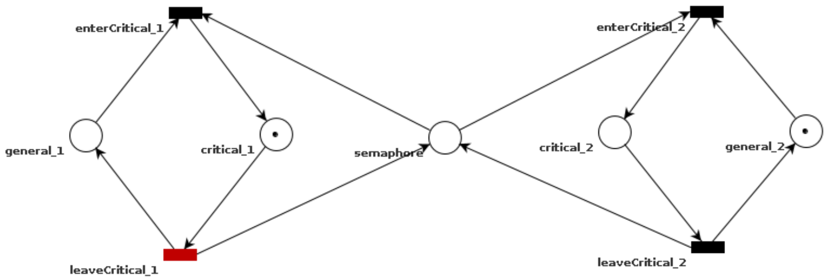
critical section, semaphore, mutex

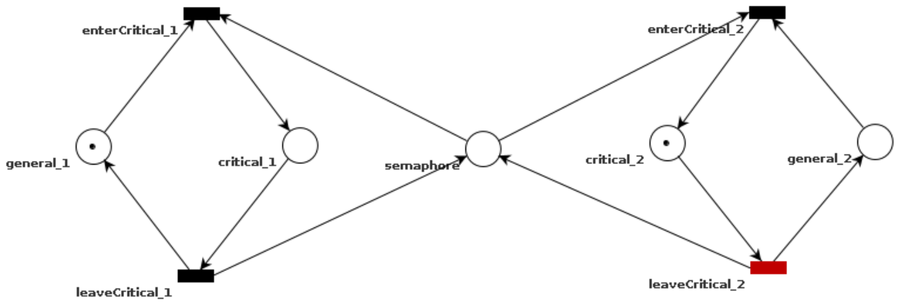


critical section, semaphore, mutex

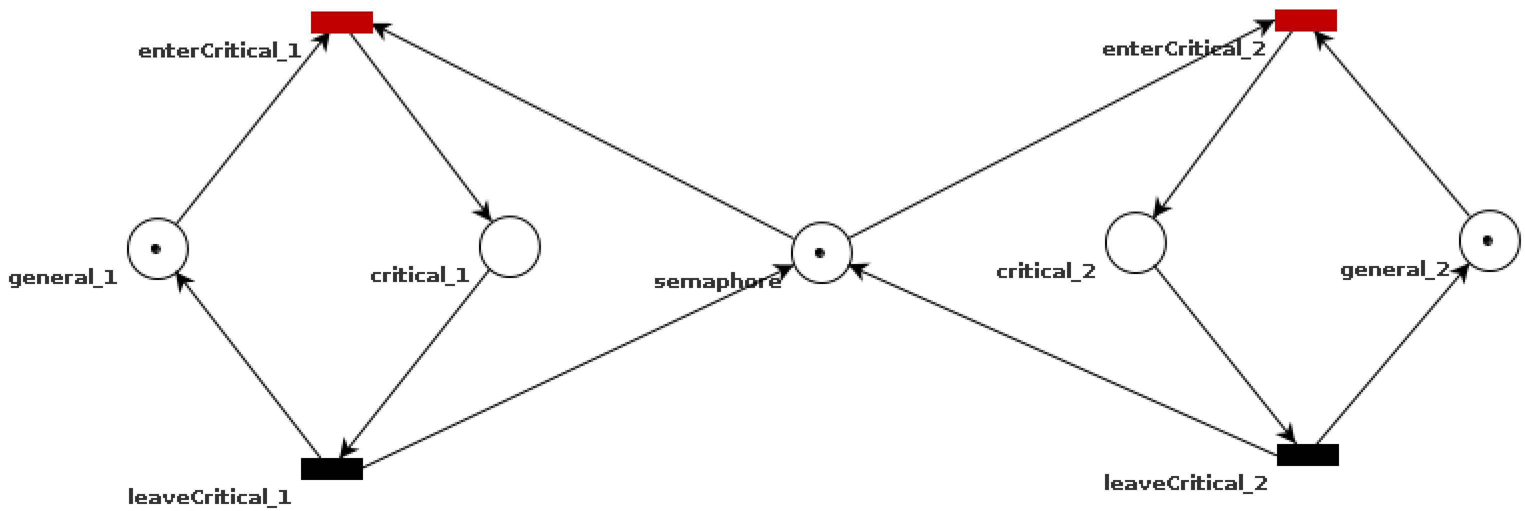


critical section, semaphore, mutex

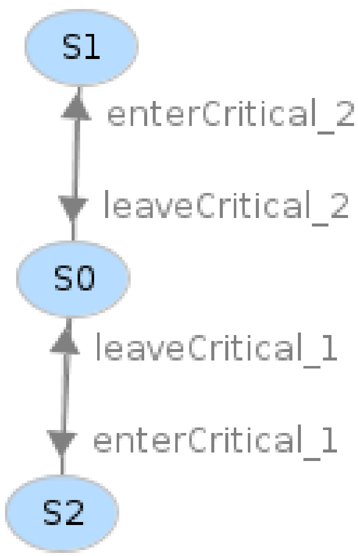




critical section, semaphore, mutex

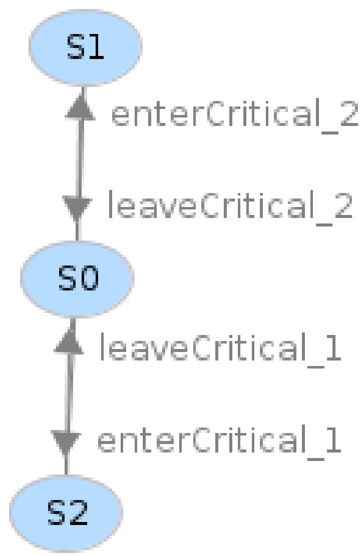


critical section, semaphore, mutex



S0 = [1, 0, 1, 0, 1]
S1 = [1, 0, 0, 1, 0]
S2 = [0, 1, 0, 0, 1]

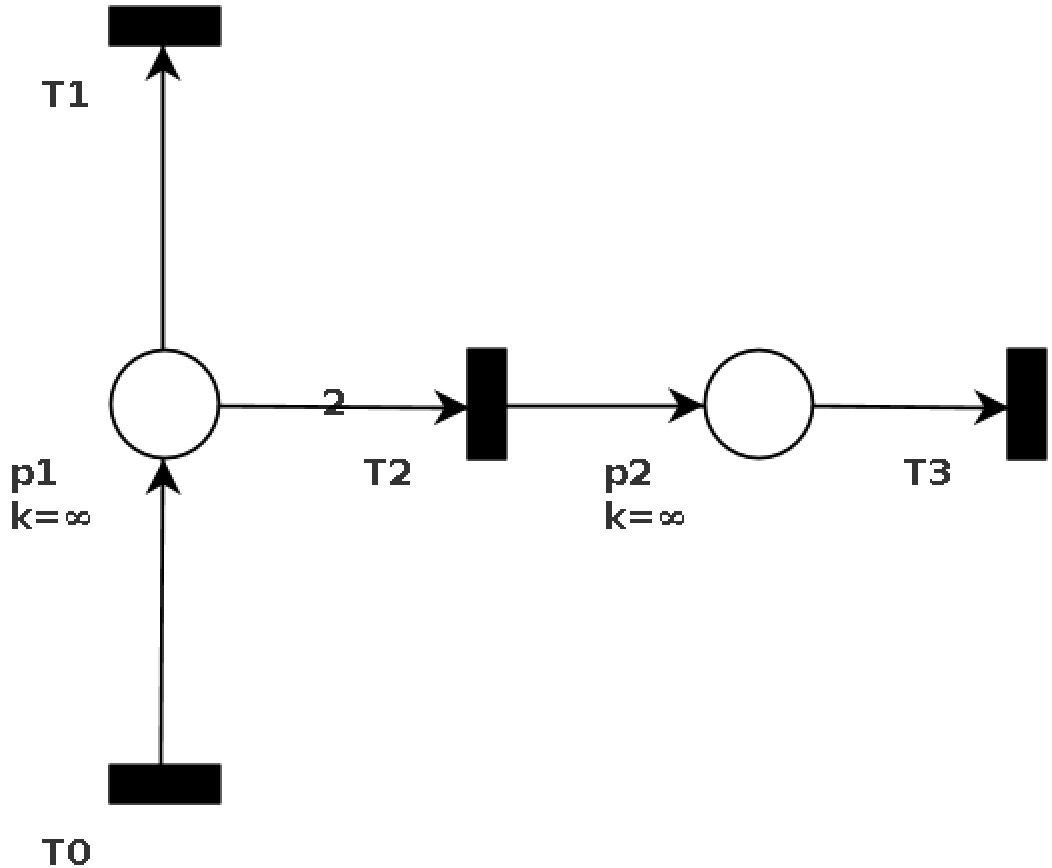
critical section, semaphore, mutex



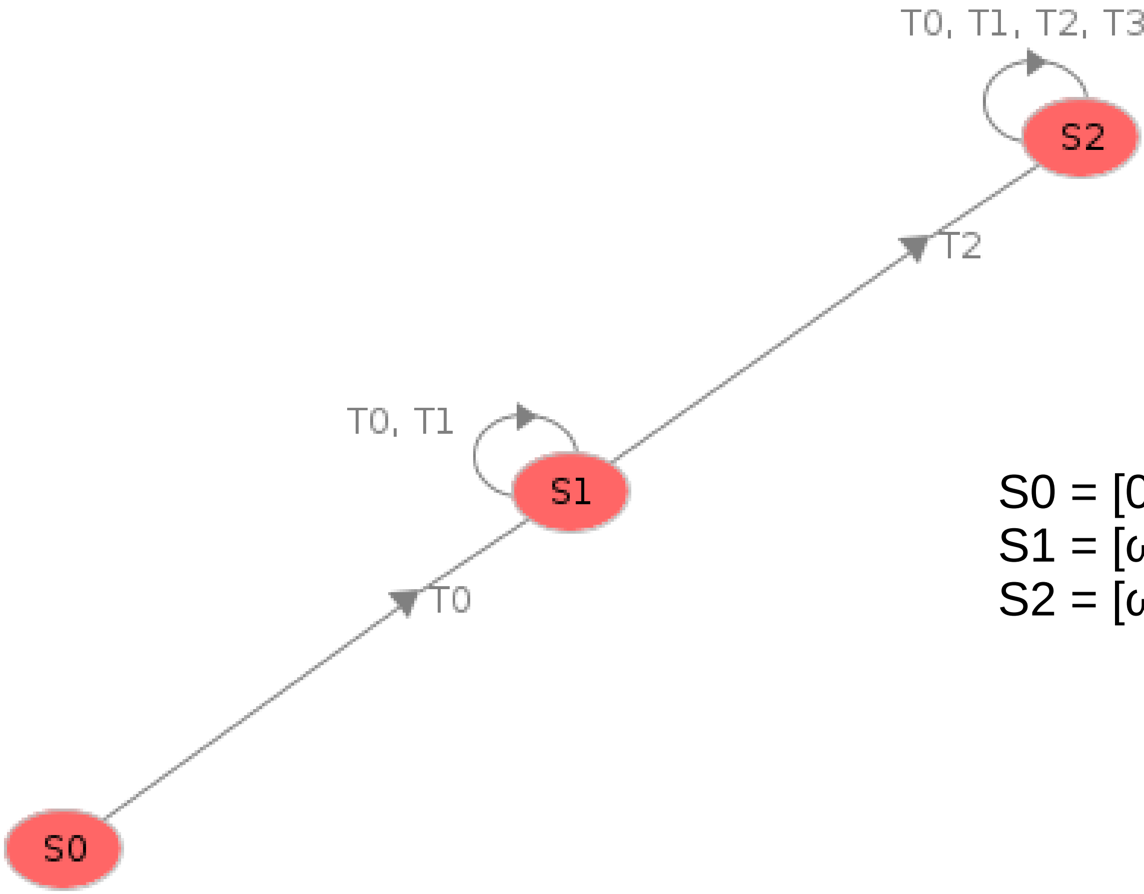
S0 = [1, 0, 1, 0, 1]
S1 = [1, 0, 0, 1, 0]
S2 = [0, 1, 0, 0, 1]

[*, 1, *, 1, *]
reachable in some path?

Infinite Capacity Petri net

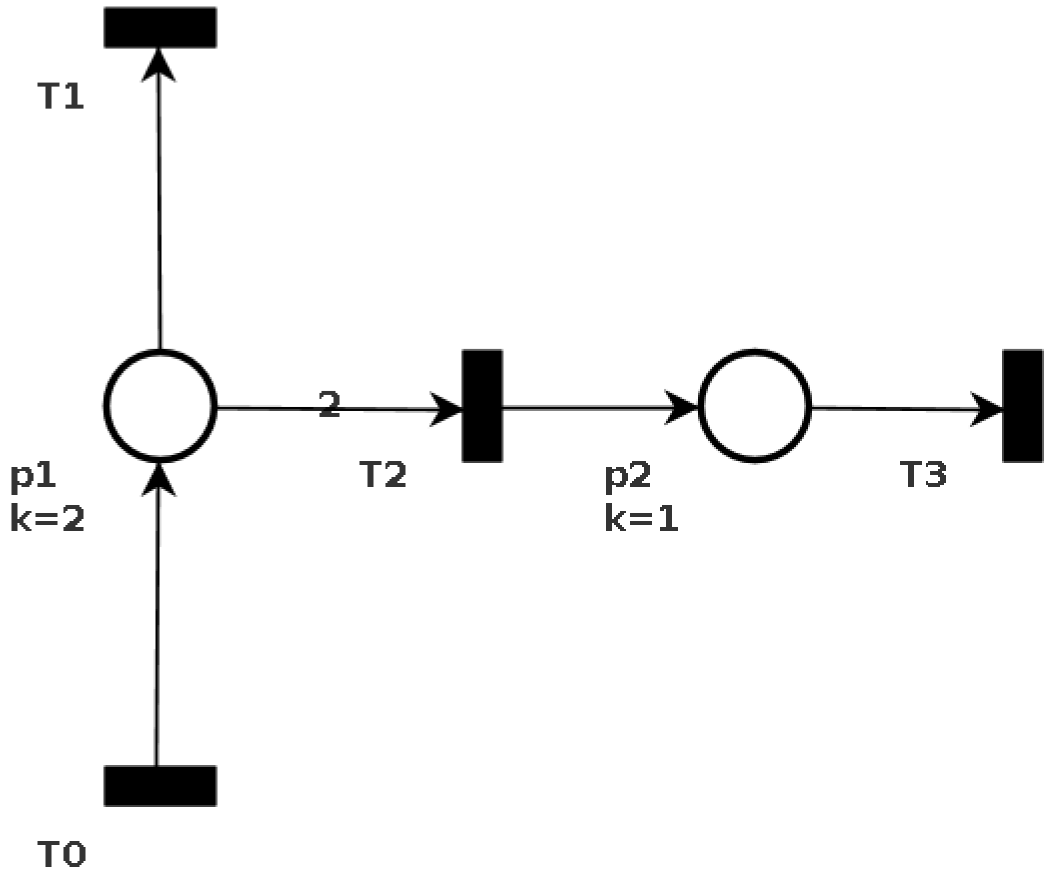


Infinite Capacity Petri net

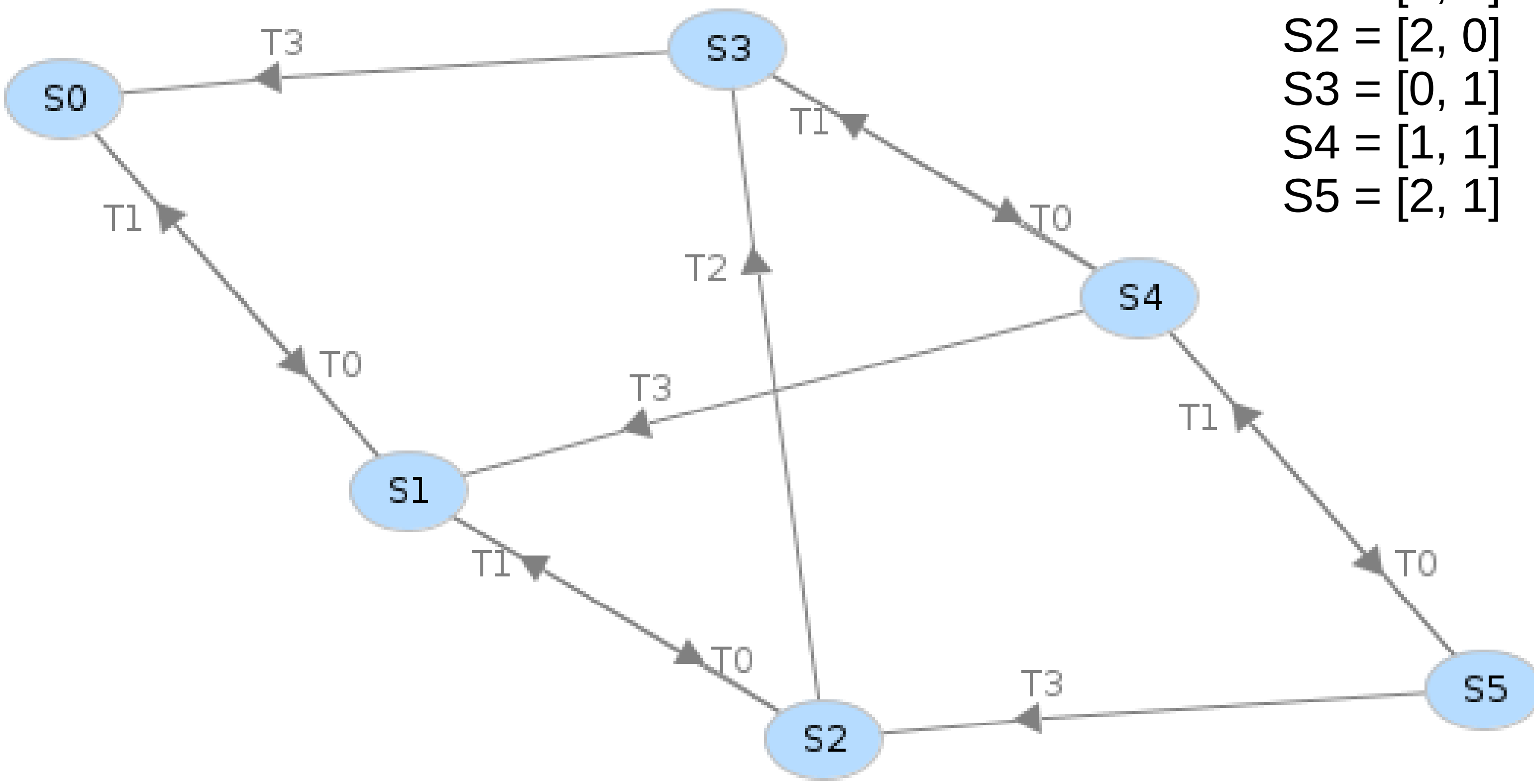


S0 = [0, 0]
S1 = [ω, 0]
S2 = [ω, ω]

Finite Capacity Petri net (FCPN)



Finite Capacity Petri net (FCPN)



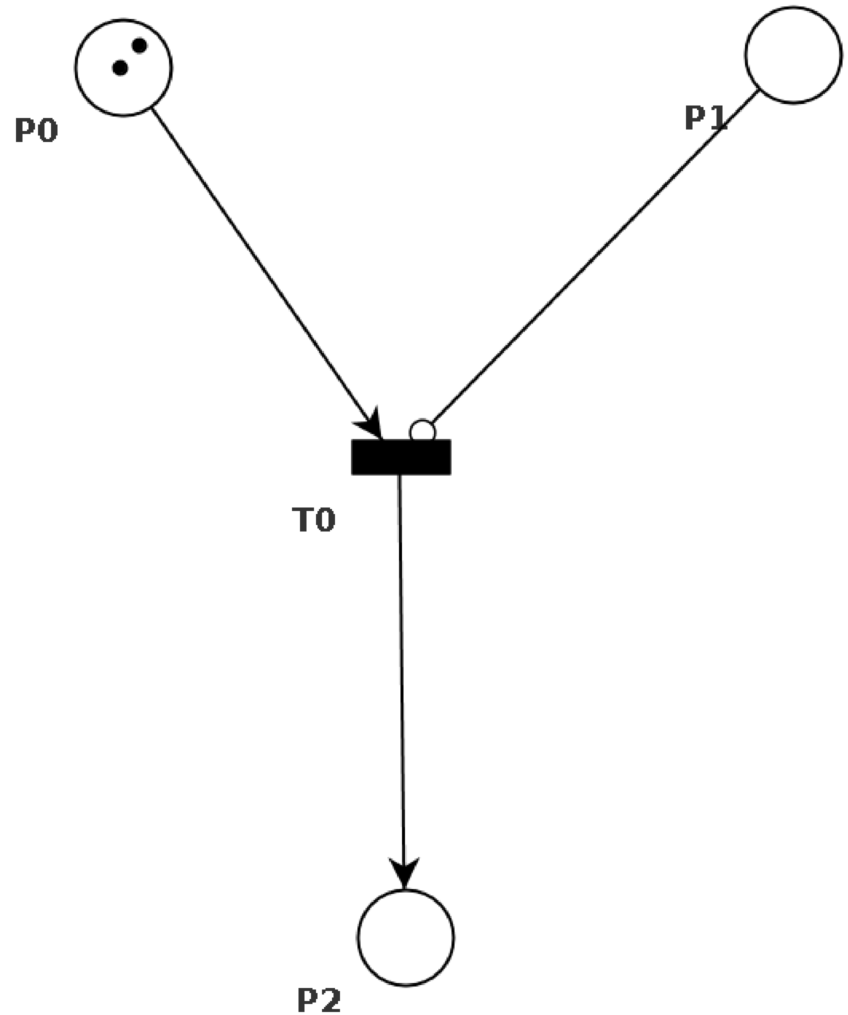
S0 = [0, 0]
S1 = [1, 0]
S2 = [2, 0]
S3 = [0, 1]
S4 = [1, 1]
S5 = [2, 1]

Finite Capacity Petri net as Infinite Capacity net

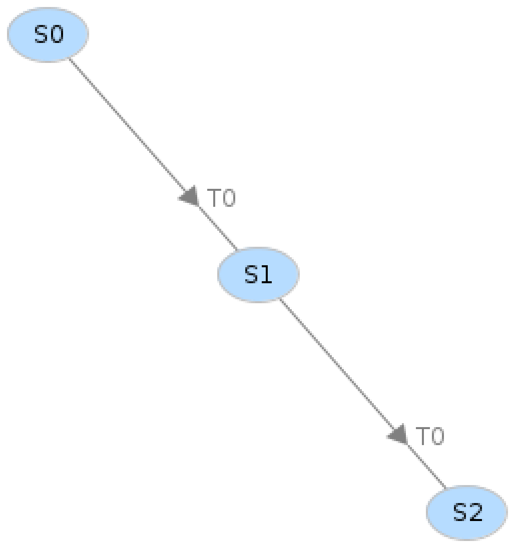
Can transform to infinite capacity net

1. Add complimentary place p' with initial marking $x_0(p') = K(p)$
2. Between each transition t and complimentary places p'
   - add arcs (t, p') or (p', t) where
   - $w(t, p') = w(p, t)$
   - $w(p', t) = w(t, p)$

P/T PN with Inhibitor Arc (makes Turing equiv.)



P/T PN with Inhibitor Arc (makes Turing equiv.)



S0 = [2, 0, 0]
S1 = [1, 0, 1]
S2 = [0, 0, 2]

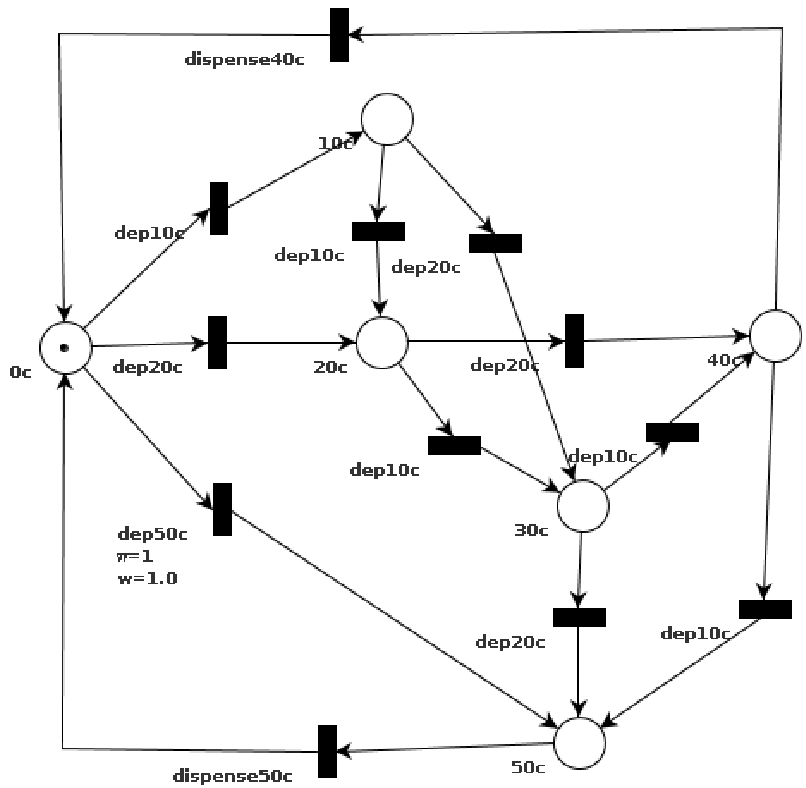
P/T PN with Inhibitor Arc (finite capacity)

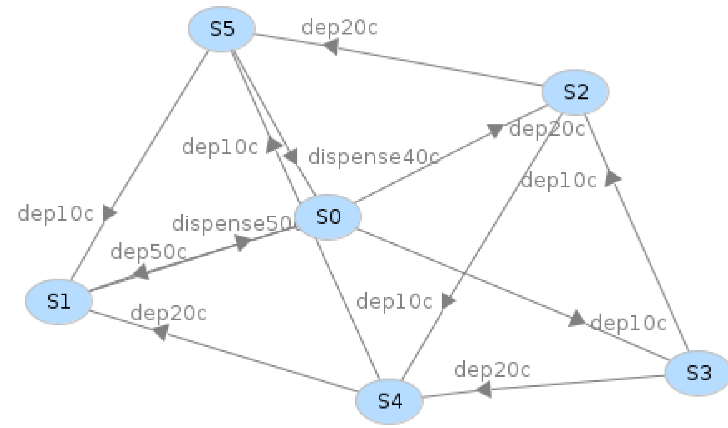
Representing a Petri net as a State Machine

Construct Reachability Graph

- Reachability Graph is State Machine
- States are tuples $(p_1, p_2, \ldots, p_n)$
- Events correspond to t_i firing
- May be infinite (ω)

Finite State Automaton represented as a Petri Net





S0 = [1, 0, 0, 0, 0, 0]
S1 = [0, 0, 0, 0, 0, 1]
S2 = [0, 0, 1, 0, 0, 0]
S3 = [0, 1, 0, 0, 0, 0]
S4 = [0, 0, 0, 1, 0, 0]
S5 = [0, 0, 0, 0, 1, 0]

[0c, 10c, 20c, 30c, 40c, 50c]

modelling the “current state” → single token

Representing a State Machine as a Petri net

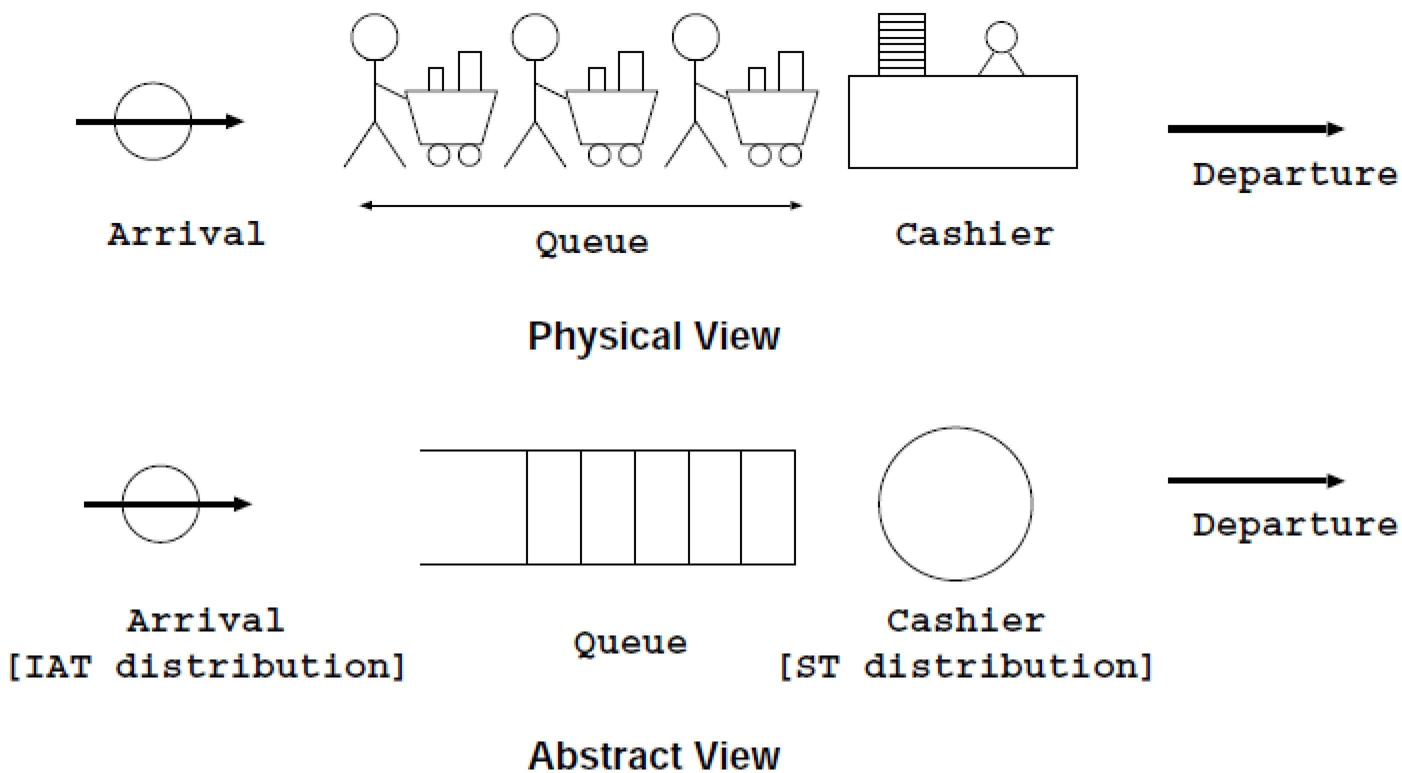
1. no output
2. with output

$\Rightarrow$ automatic (though inefficient) transformation

FSA without output

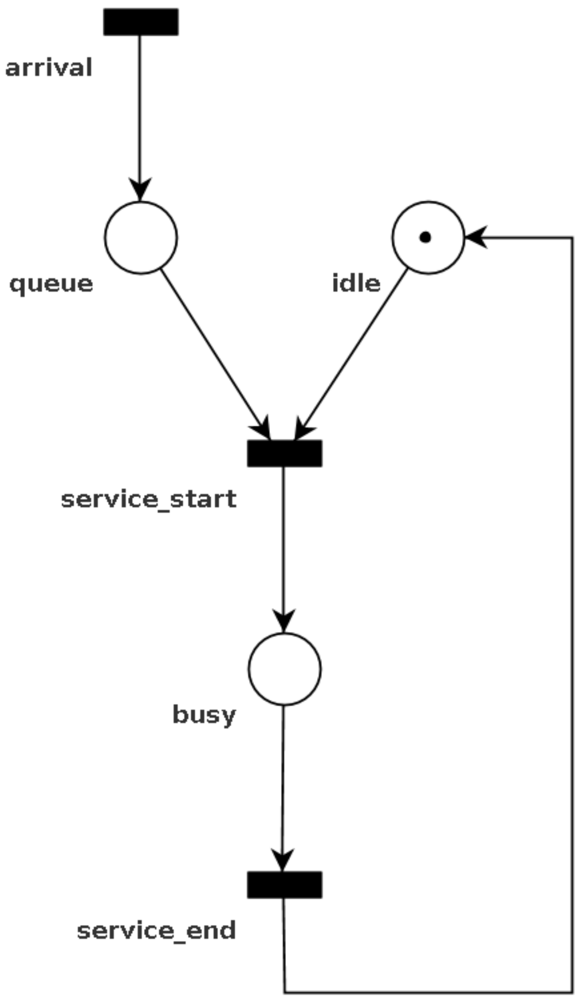
FSA with output

Petri net models for Queueing Systems

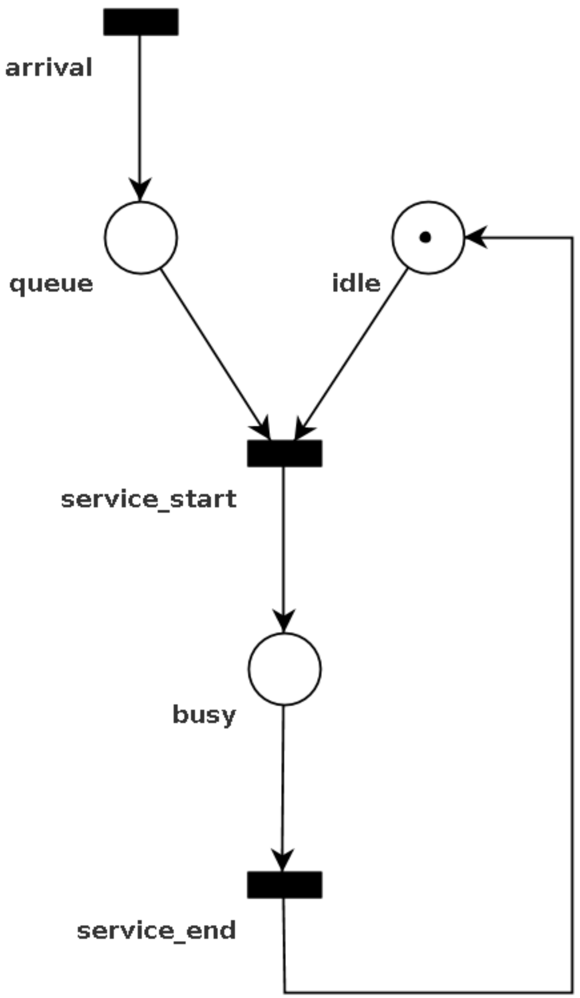


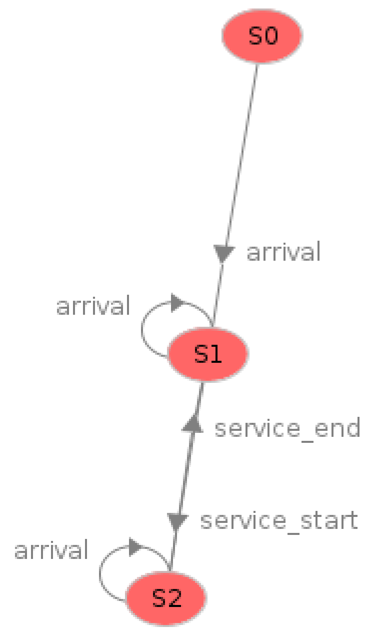
Capacity Constraints for Resource Conservation

Simple Server/Queue



Simple Server/Queue

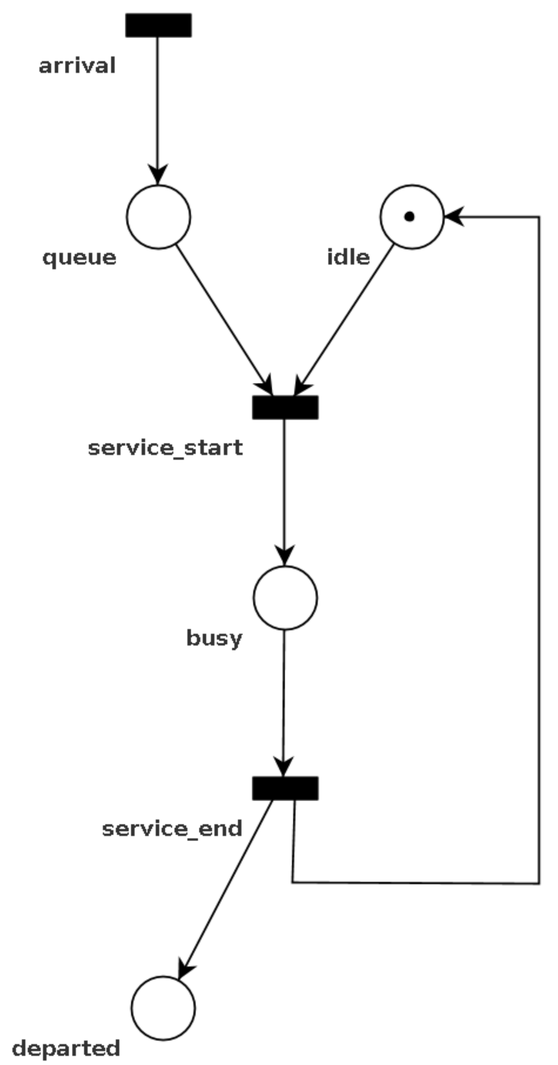




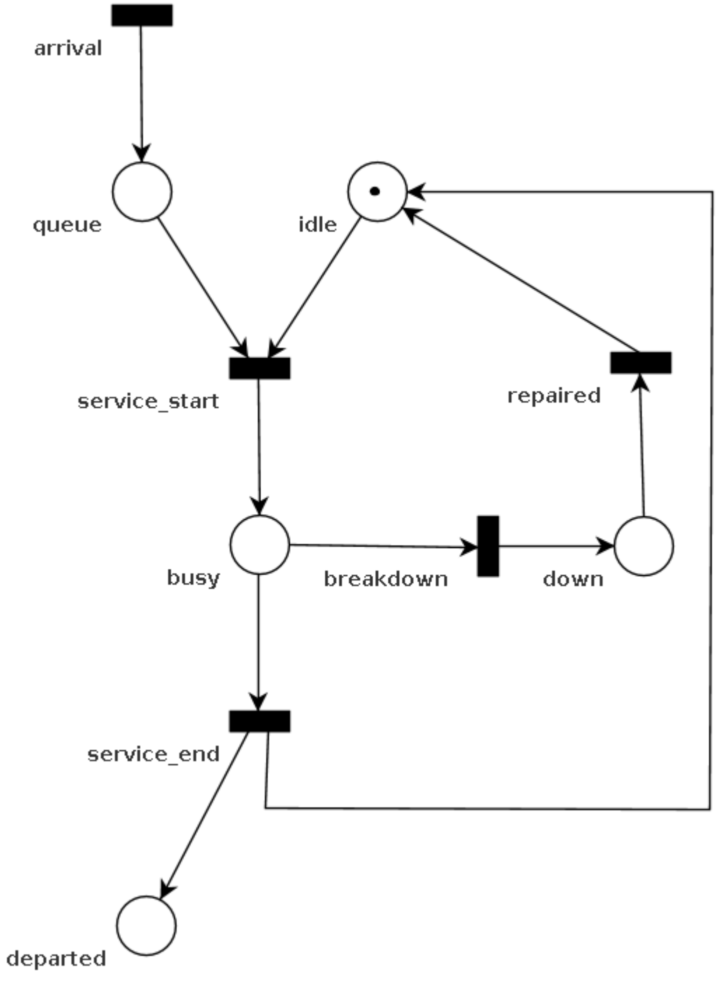
S0 = [0, 1, 0]
S1 = [ω, 1, 0]
S2 = [ω, 0, 1]

[queue, idle, busy]

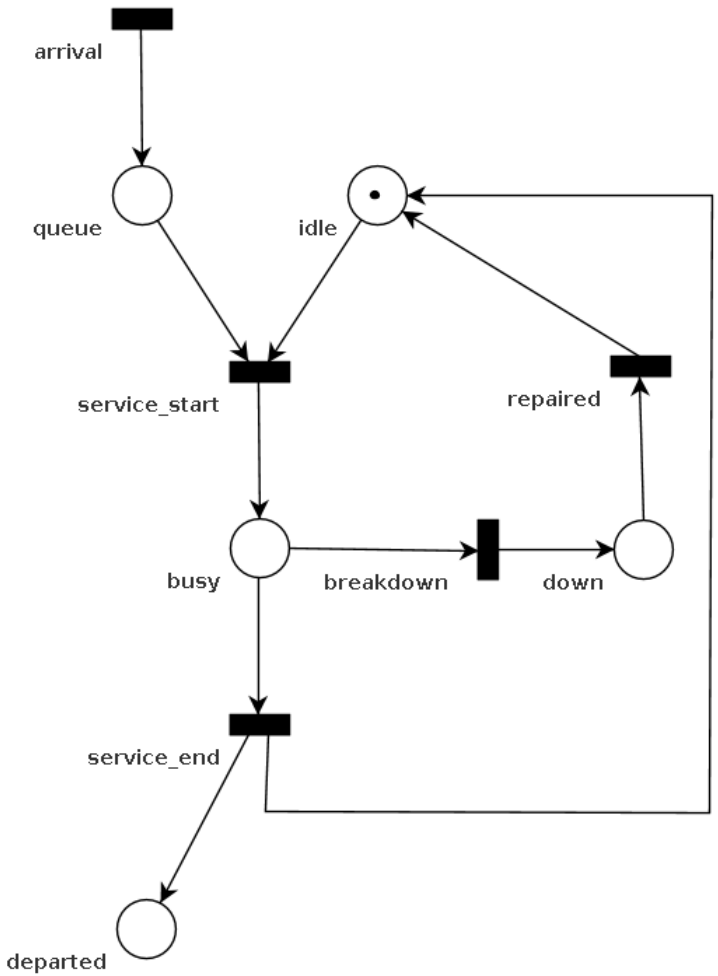
Simple Server/Queue
departure modelled explicitly

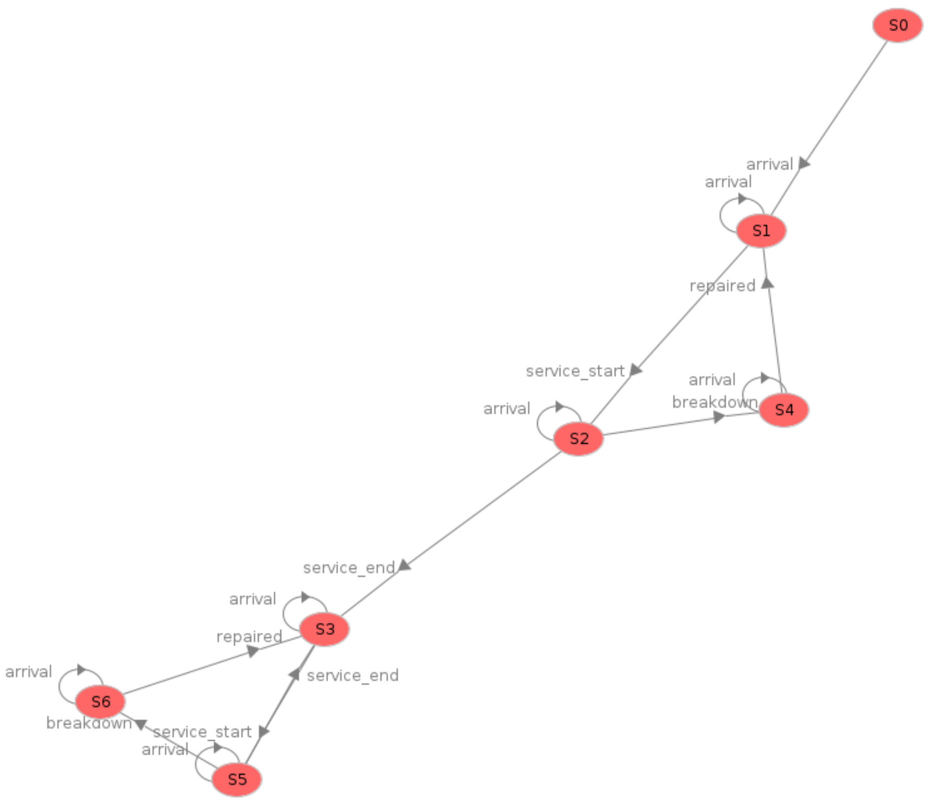


Simple Server/Queue
with server breakdown (and repair)



Simple Server/Queue
with server breakdown (and repair)





Modular Composition: Communication Protocol

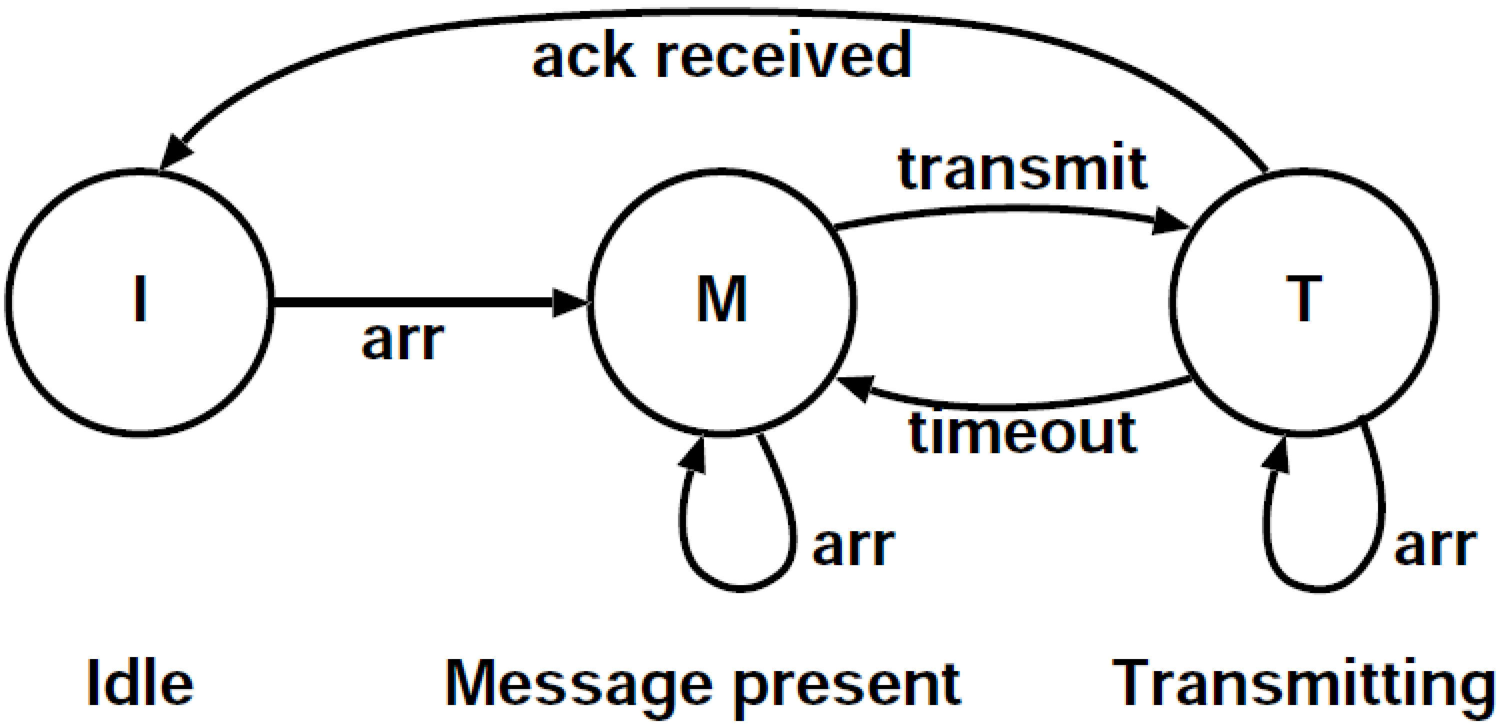
Build incrementally:

1. Single transmitter: FSA vs. Petri net
2. Two transmitters competing for channel

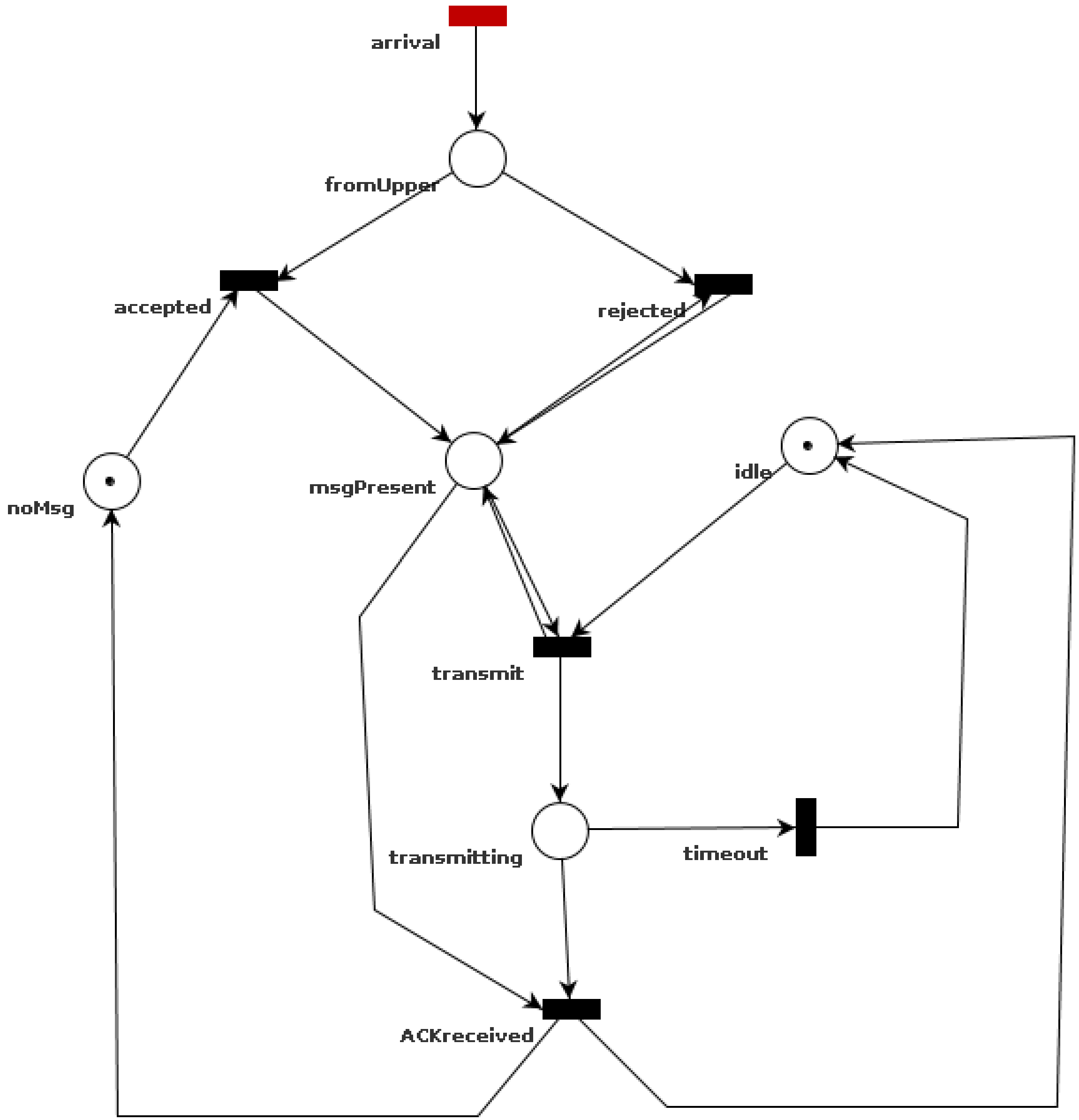
Pros/Cons of Petri net models (depends on goals !):

- Petri net is more complex than FSA for single transmitter
- More insight
- Incremental modelling
- Modular modelling
- Intuitive modelling of concurrency

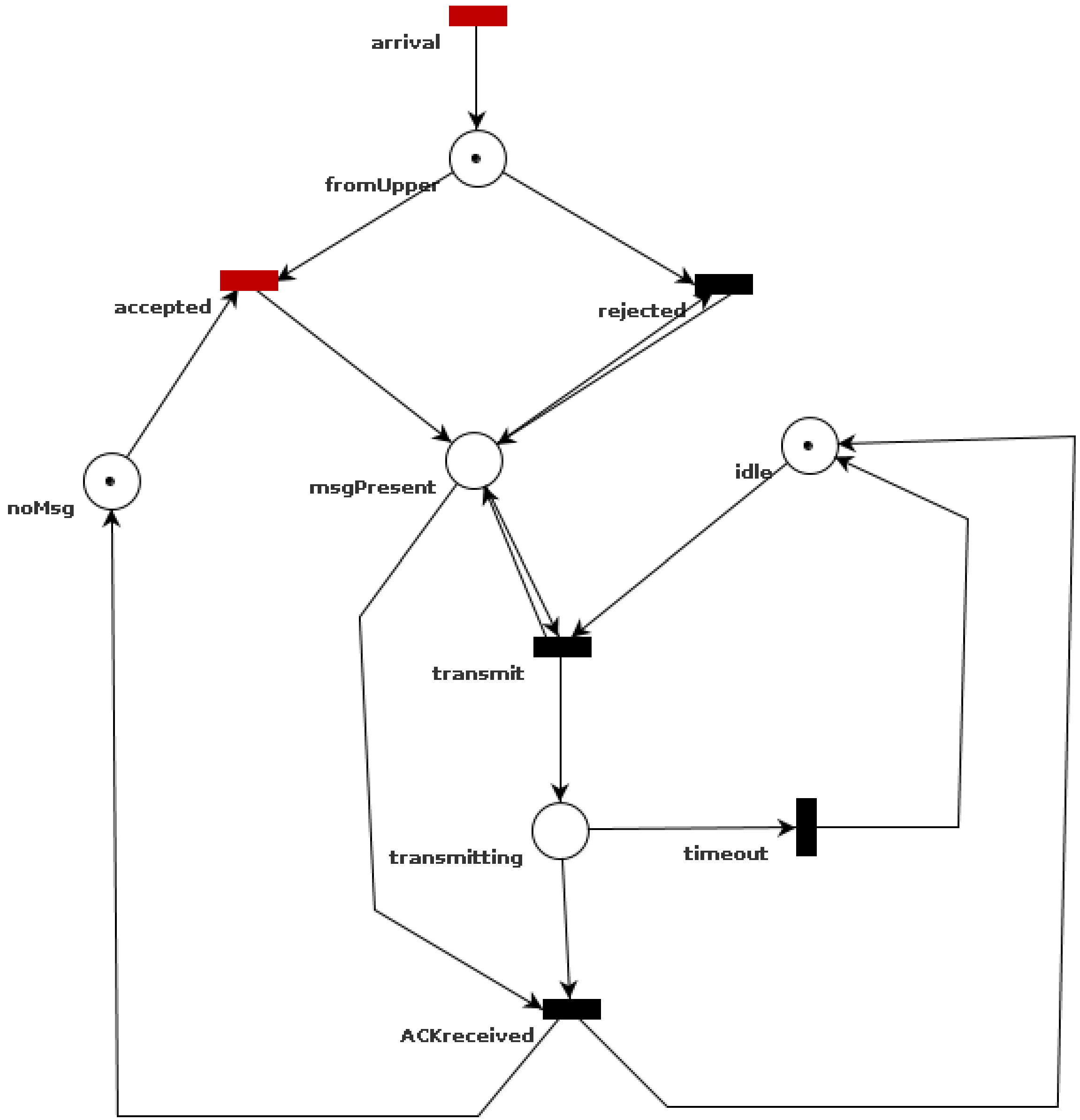
Single Transmitter FSA



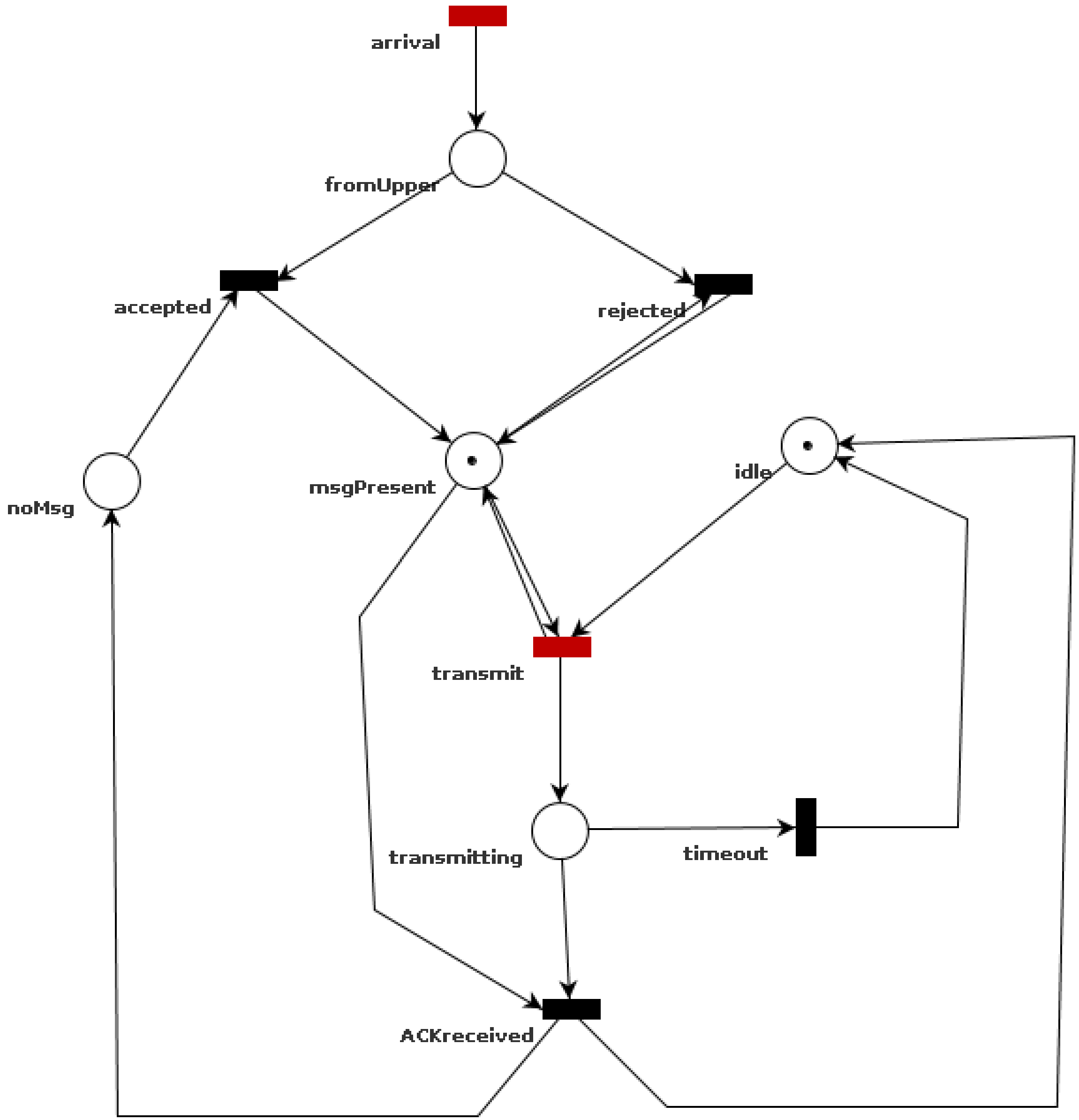
Single transmitter



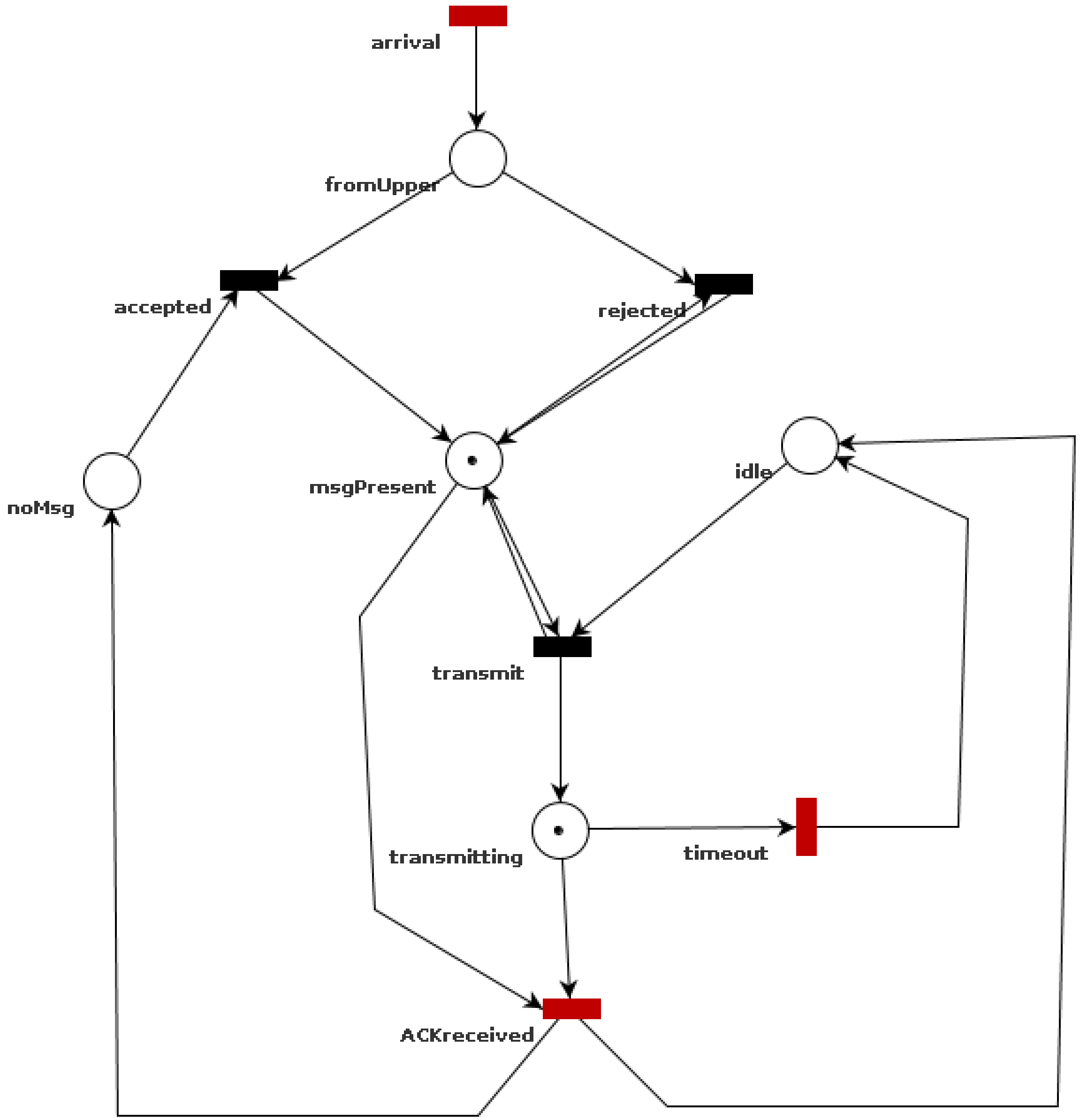
Single transmitter



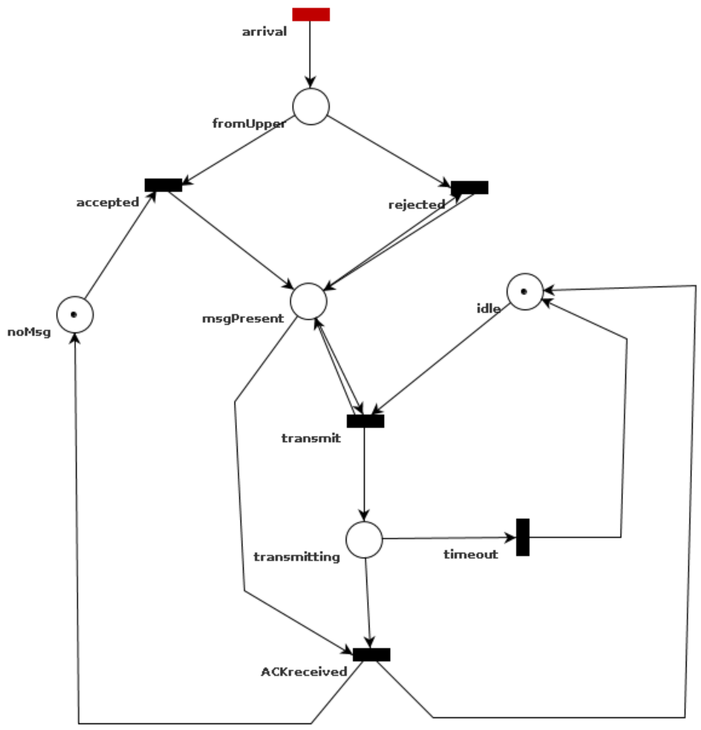
Single transmitter

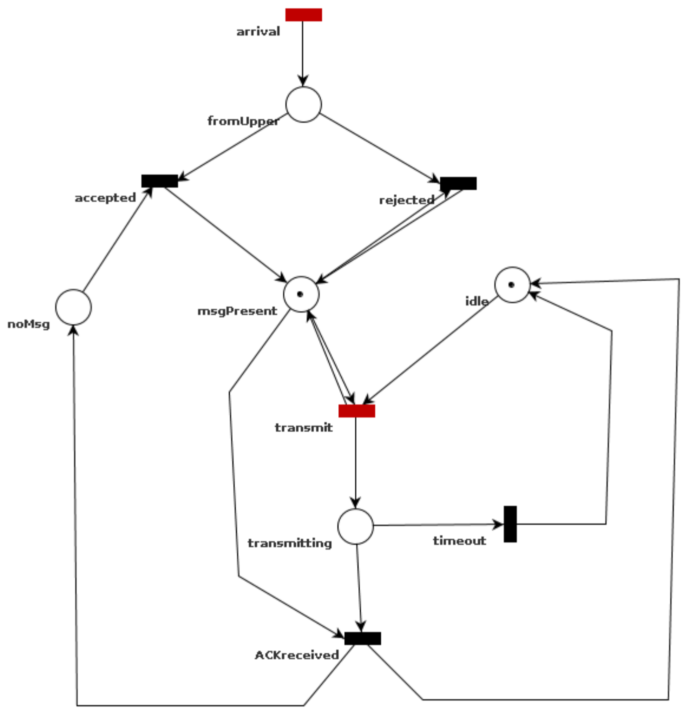


Single transmitter

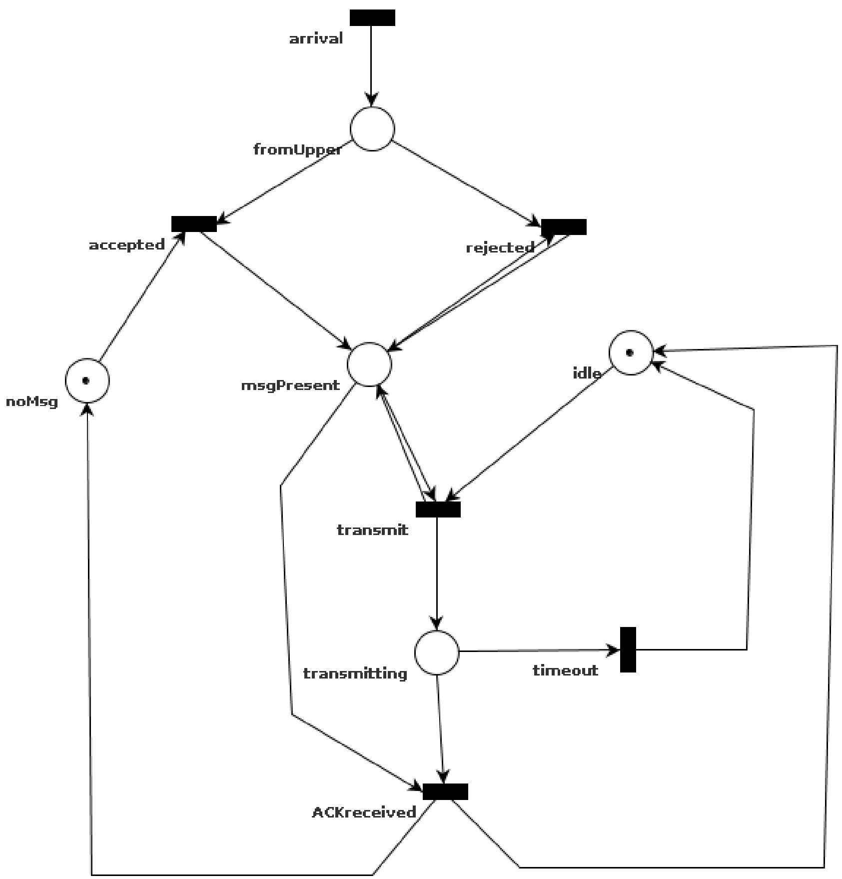


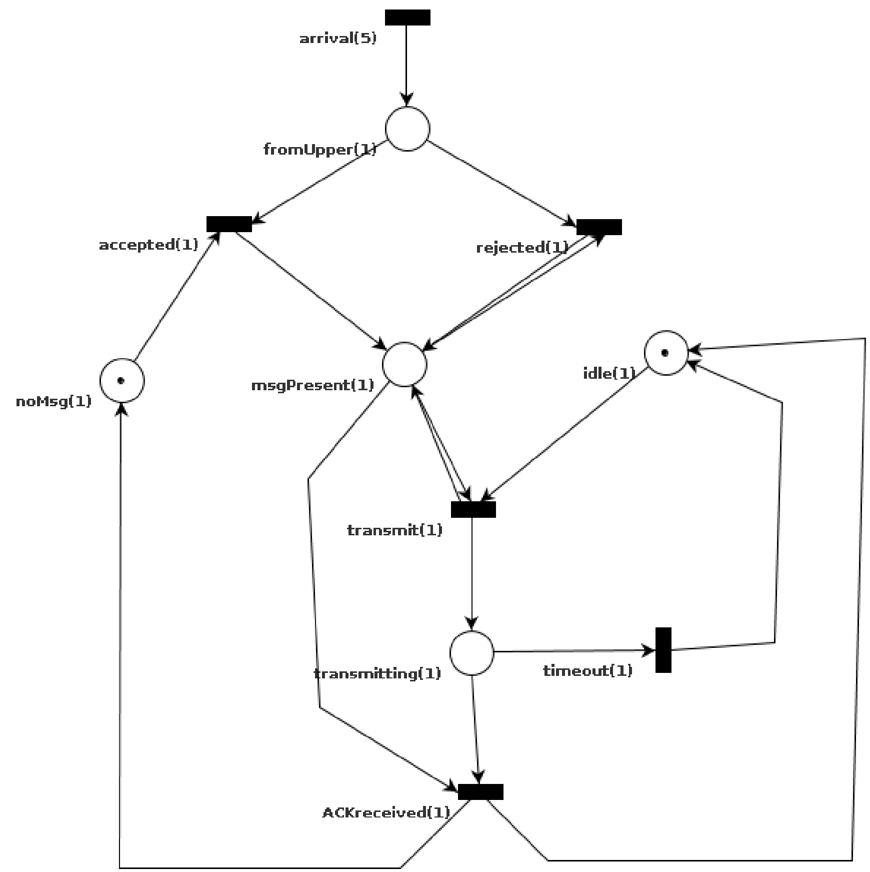
Single transmitter



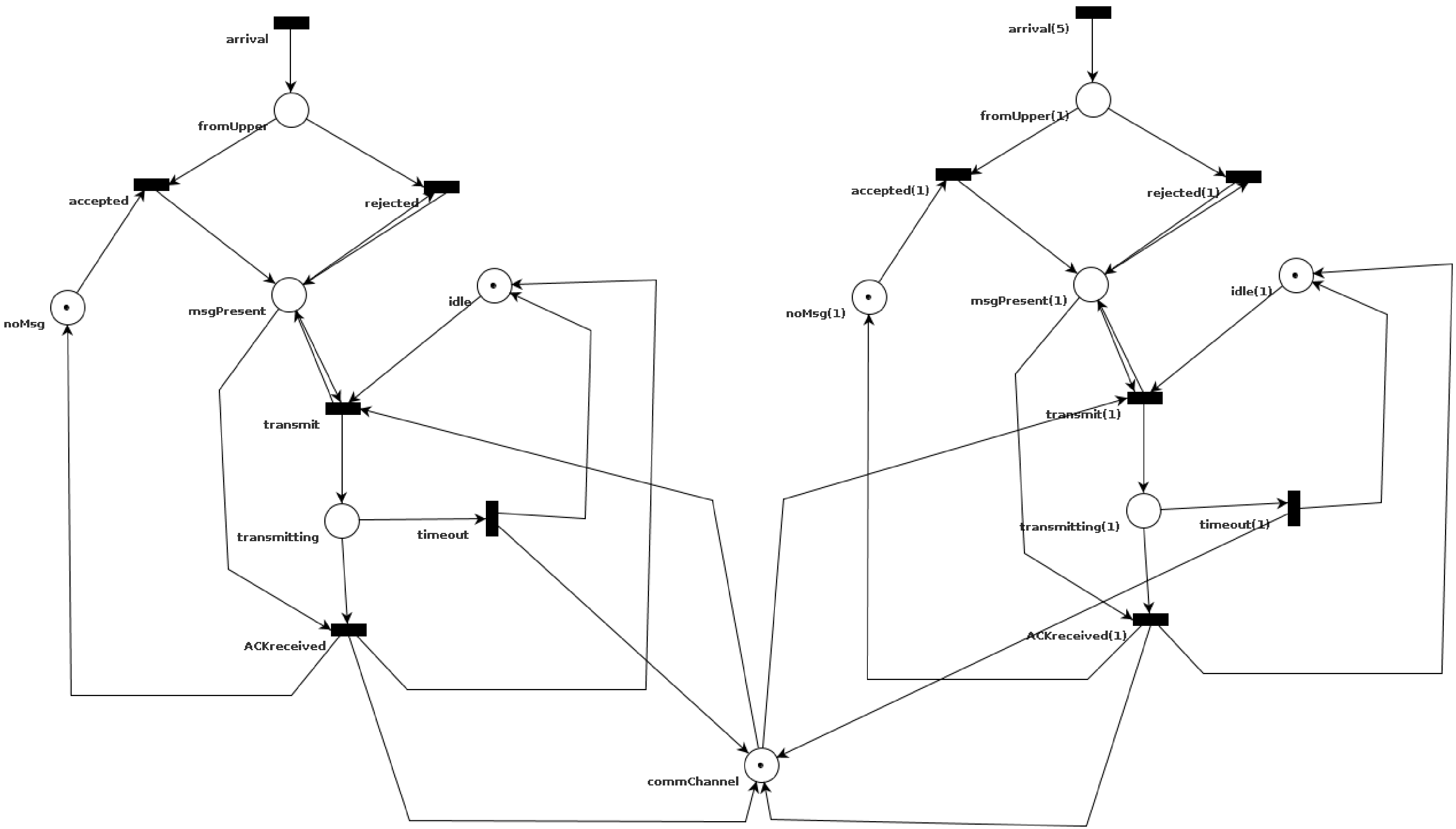


Two independent transmitters





Two transmitters competing for a single communication channel



Analysis of Petri nets

Analysis of *logical* or *qualitative* behaviour.

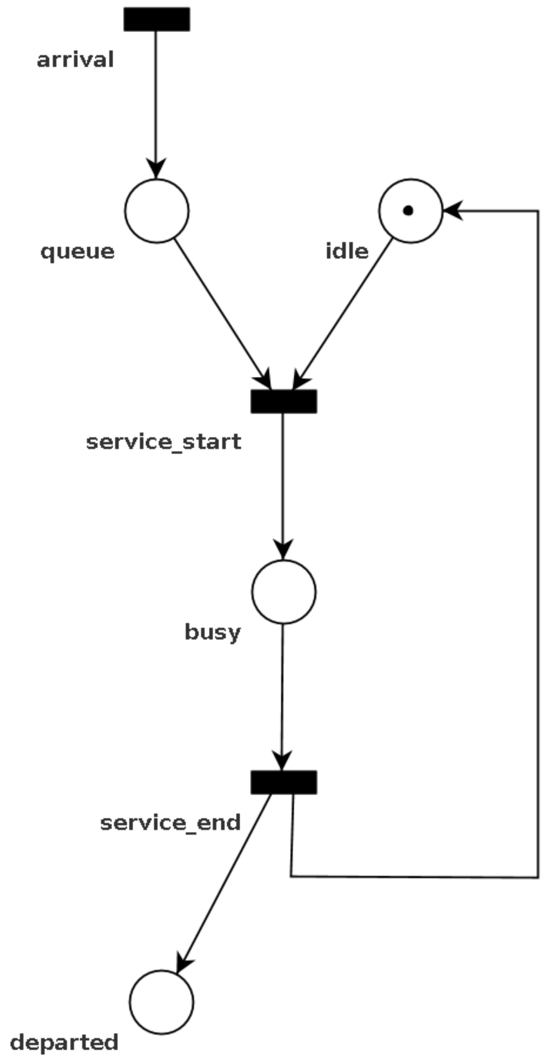
Resource sharing $\Rightarrow$ *fair* usage of resources:

- Boundedness
- Conservation
- Liveness and Deadlock
- State Reachability
- State Coverability
- Persistence
- Language Recognition

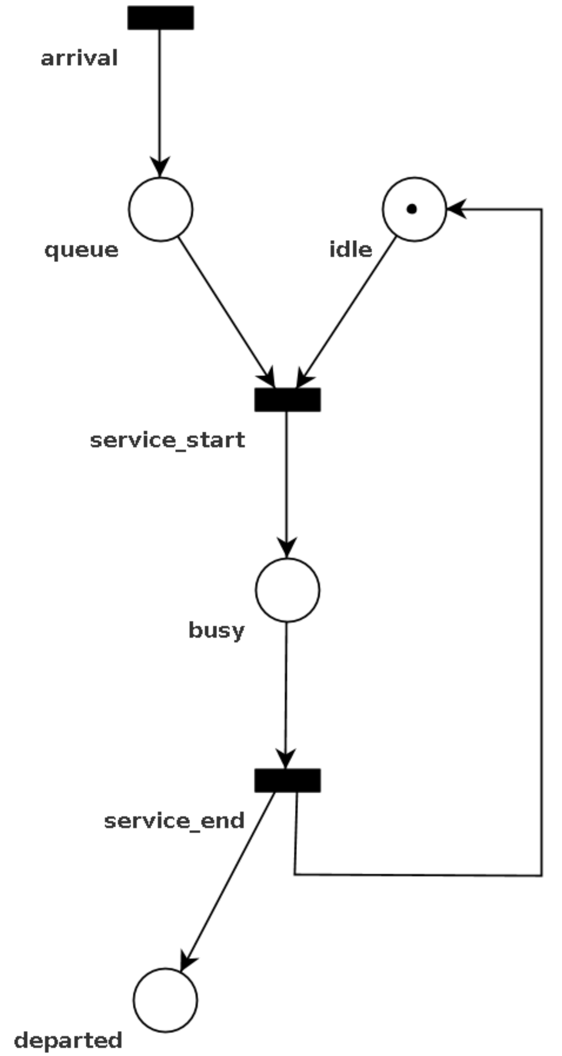
Boundedness

- Example: upper bound on number of customers in queue.
- Definition: A place $p_i \in P$ in a Petri net with initial state $\mathbf{x}_0$ is $k-$*bounded* or $k-$*safe* if
  $x(p_i) \leq k$ for all states in all possible sample paths.
- A $1-$bounded place is called *safe*.
- If a place is $k-$bounded for some k, the place is *bounded*.
- If all places are bounded, the Petri net is *bounded*.

Bounded vs. Unbounded

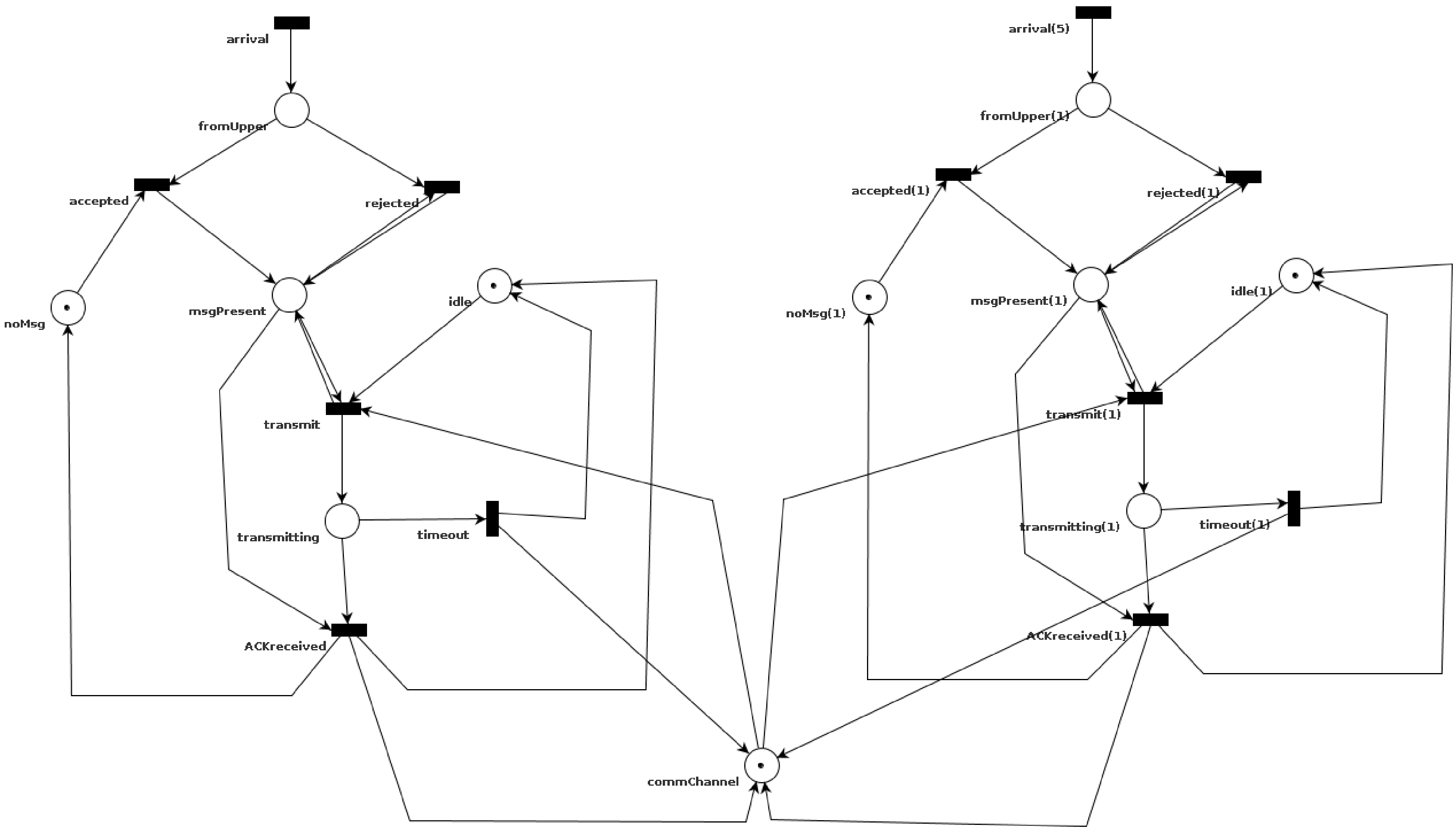


Conservation (invariants)



Sum of `busy` and `idle` marking is *constant* across *all* sample paths

Conservation (invariants): weighted sum



2 x transmitting + 1 x idle + 1x commChannel = 2

Conservation

A Petri net with initial state x_0 is *conservative with respect to* $\gamma = [\gamma_1, \gamma_2, \ldots, \gamma_n]$ if

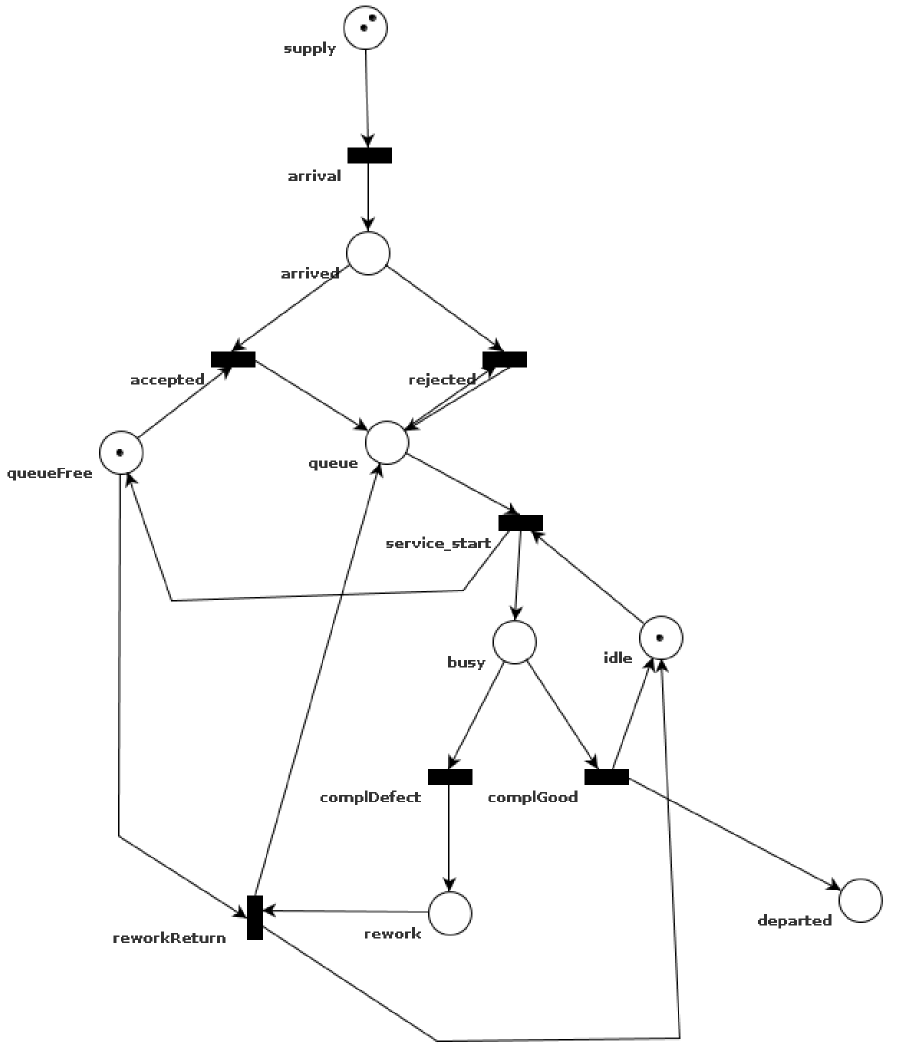
$$\Sigma_{i=1}^{n} \gamma_i x(p_i) = constant$$

for all states in all possible sample paths.

Liveness and Deadlock

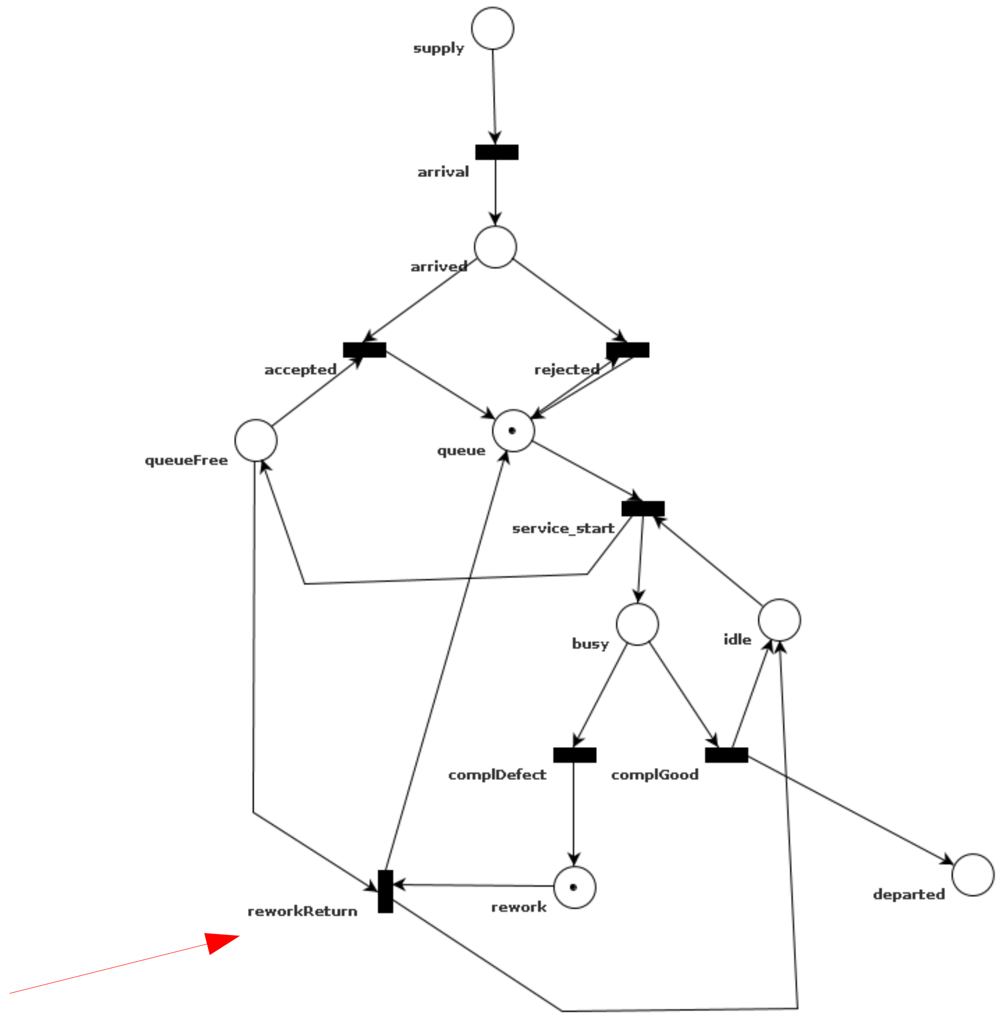
- Cyclic dependency $\Rightarrow$ wait indefinitely
- Deadlock
- Deadlock avoidance: avoid certain states in sample paths

Deadlock in queueing system with rework



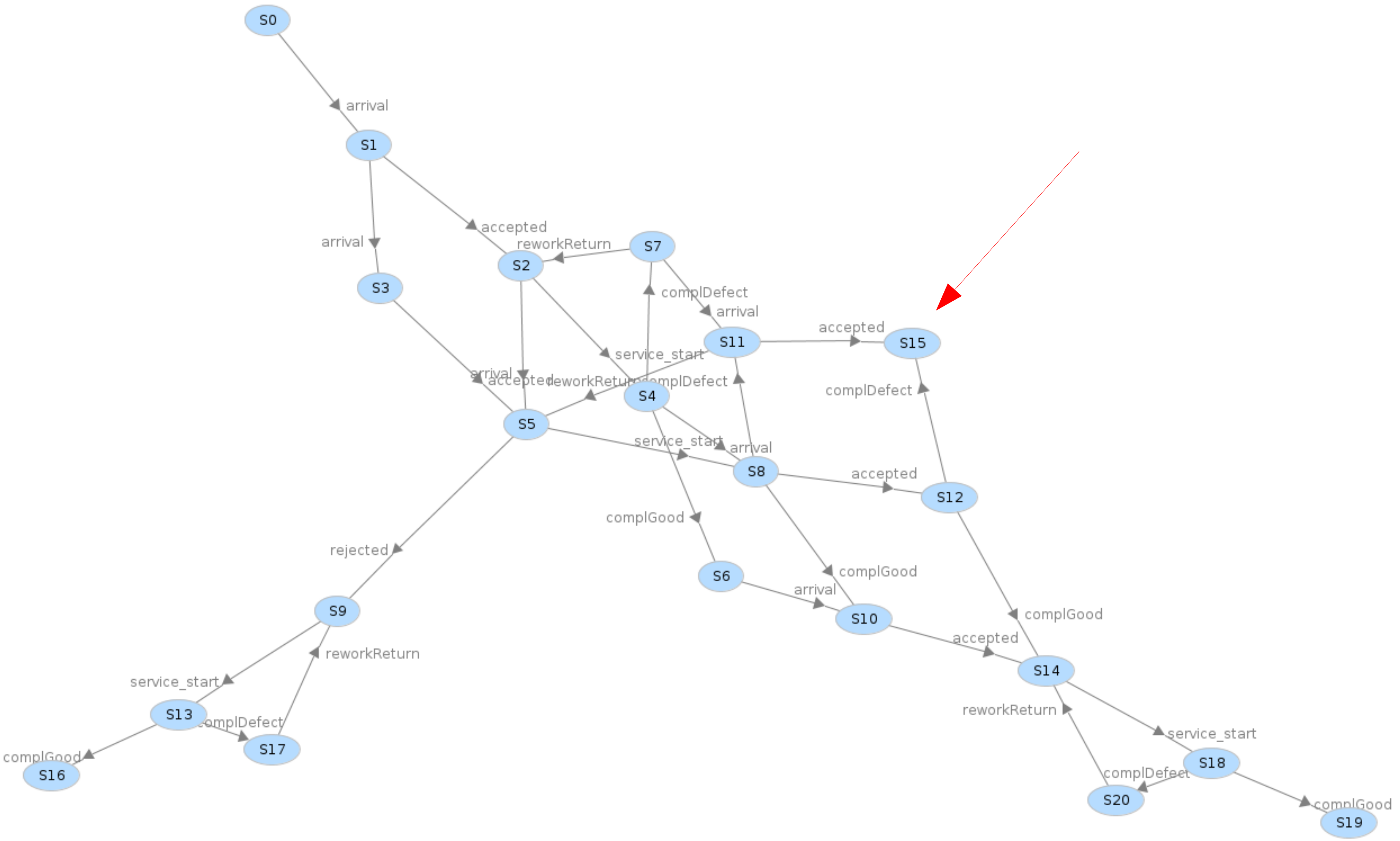
[queueFree, queue, rework] = [0, 1, 1] → deadlock

Deadlock in queueing system with rework



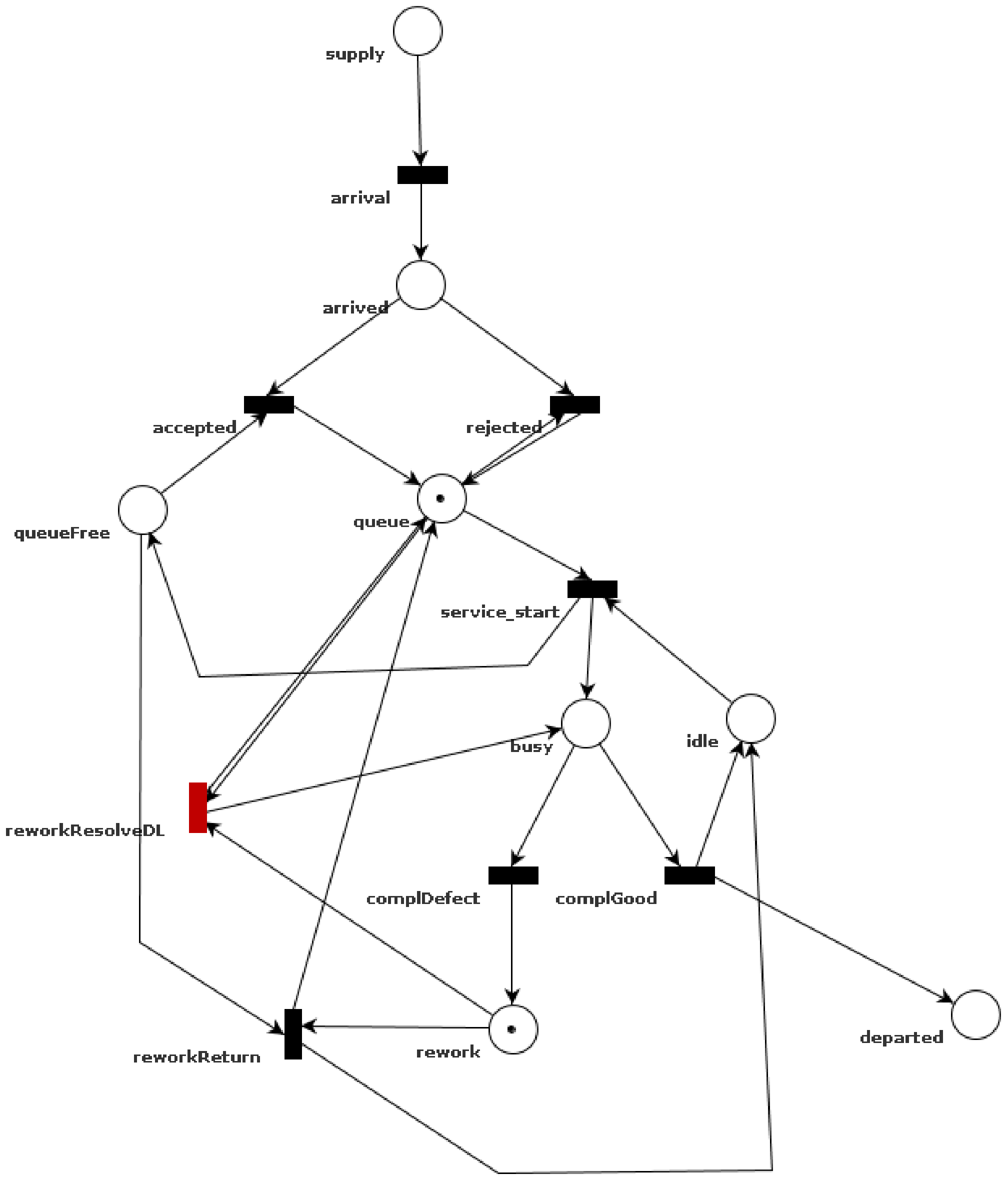
[queueFree, queue, rework] = [0, 1, 1] → deadlock

Deadlock in queueing system with rework

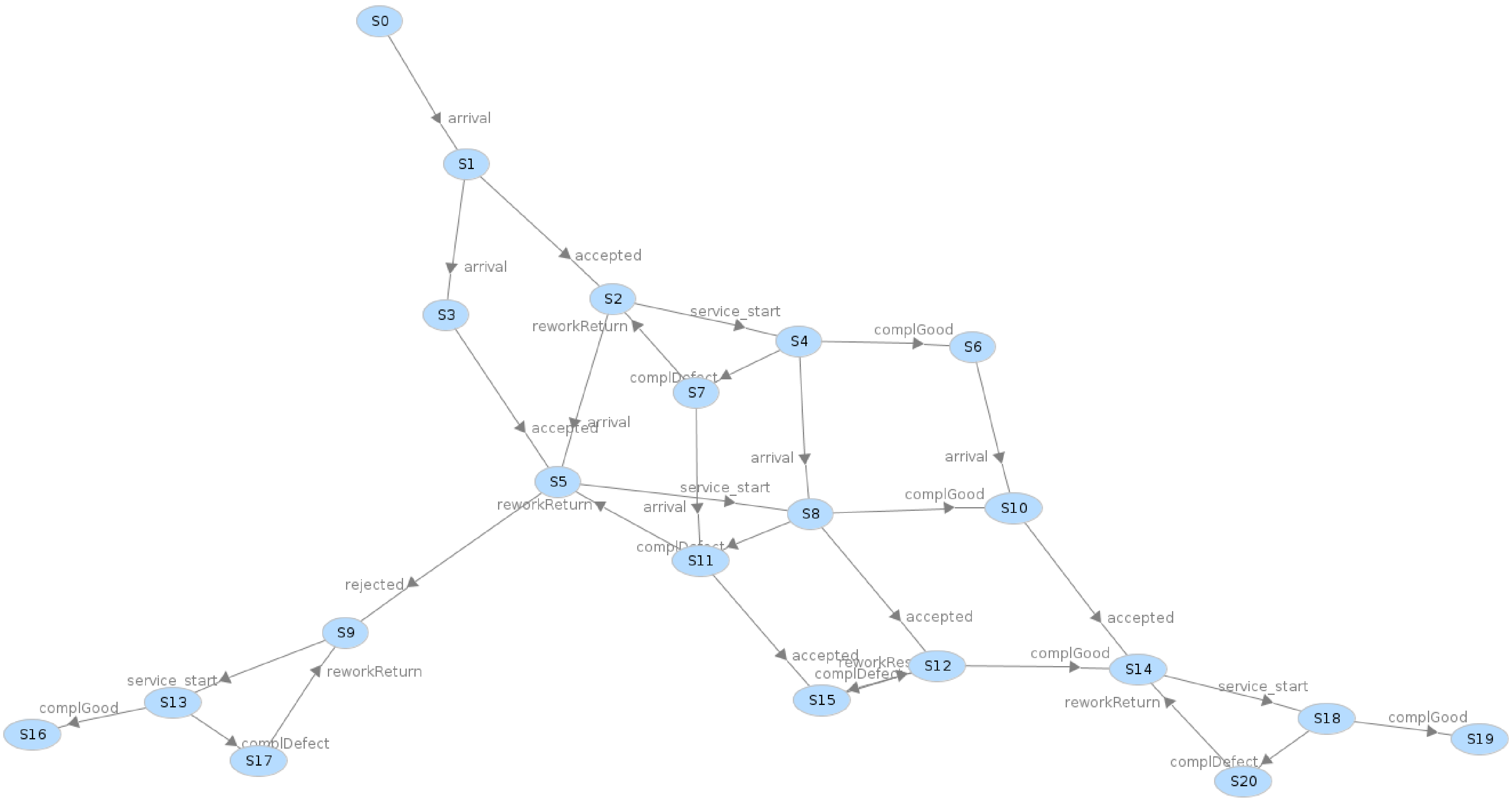


[queueFree, queue, rework] = [0, 1, 1] → deadlock

Deadlock resolved (avoided)



Deadlock resolved (avoided)

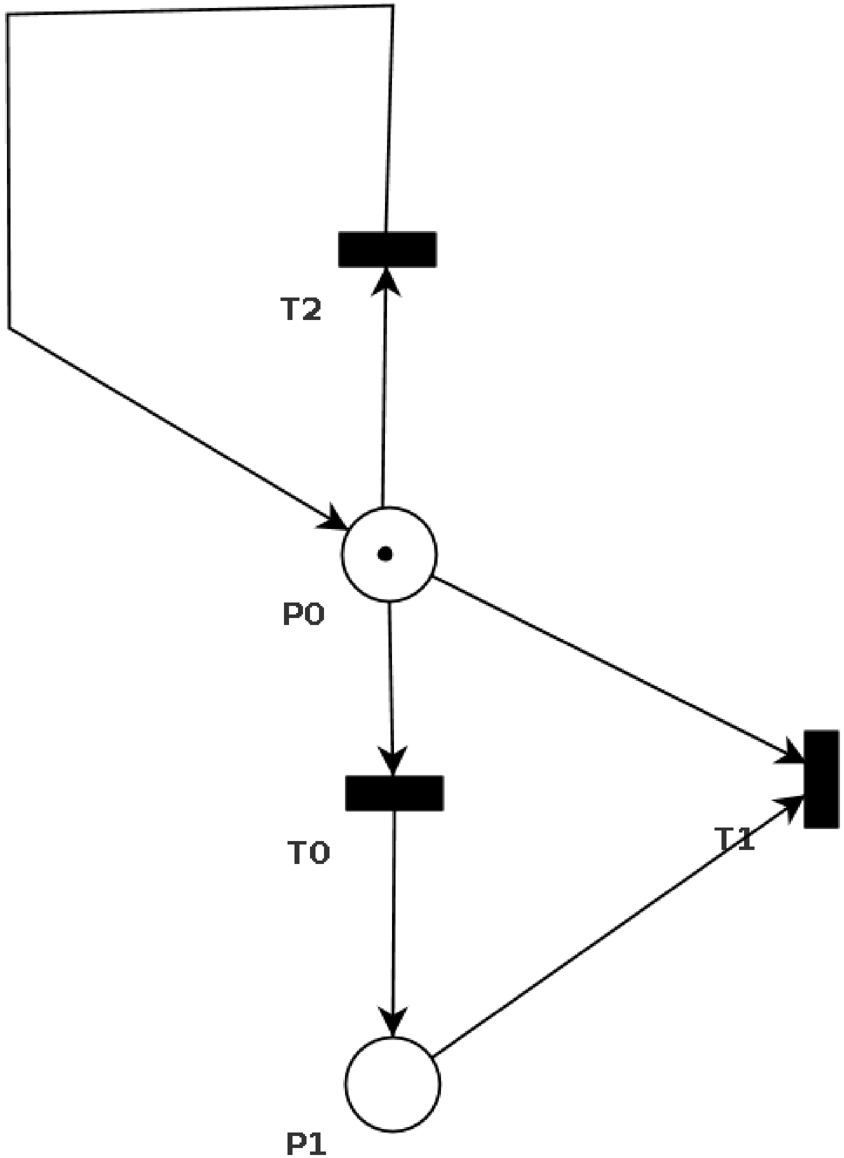


Liveness

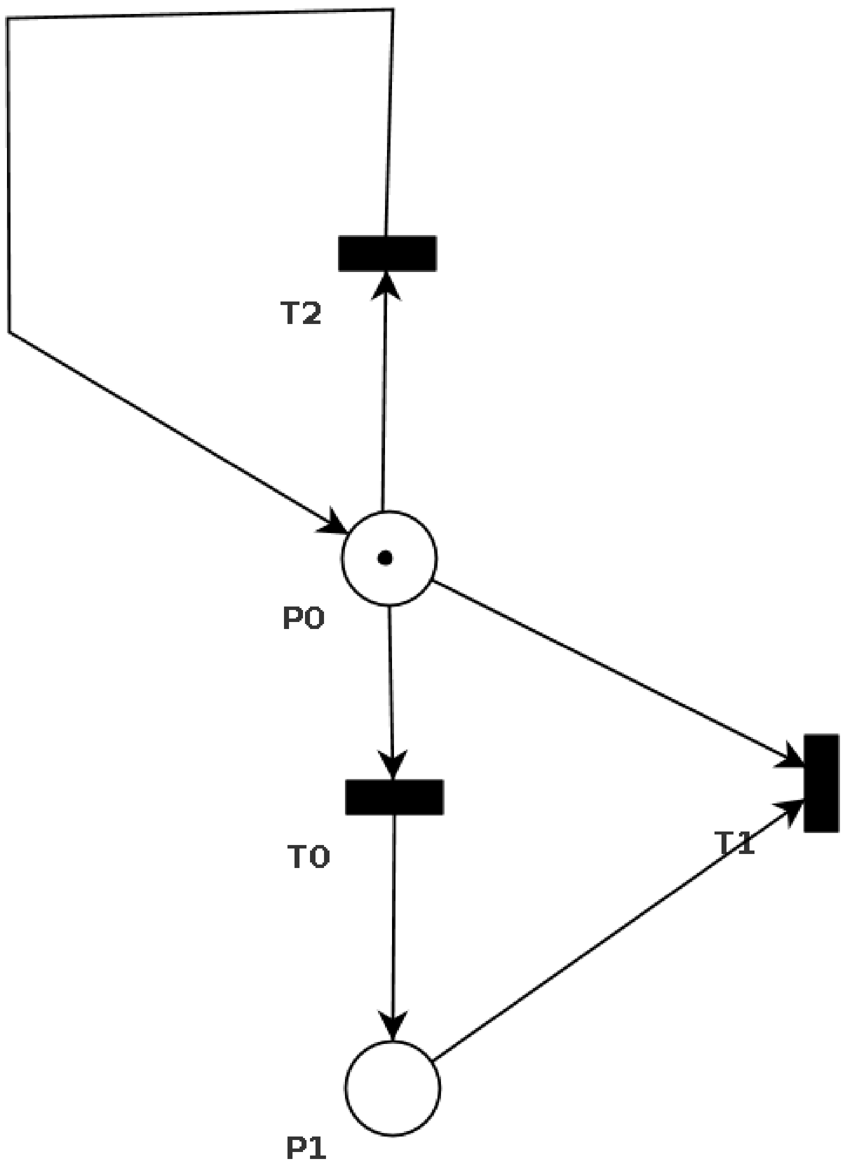
Given initial state $\mathbf{x}_0$, a transition in a Petri net is:

- L0-live (dead): if the transition can never fire.
- L1-live: if there is some firing sequence from $\mathbf{x}_0$ such that the transition can fire at least once.
- L2-live: if the transition can fire at least k times for some given positive integer k.
- L3-live: if there exists some infinite firing sequence in which the transition appears infinitely often.
- L4-live: if the transition is L1-live for every possible state reached from $\mathbf{x}_0$.

Liveness example

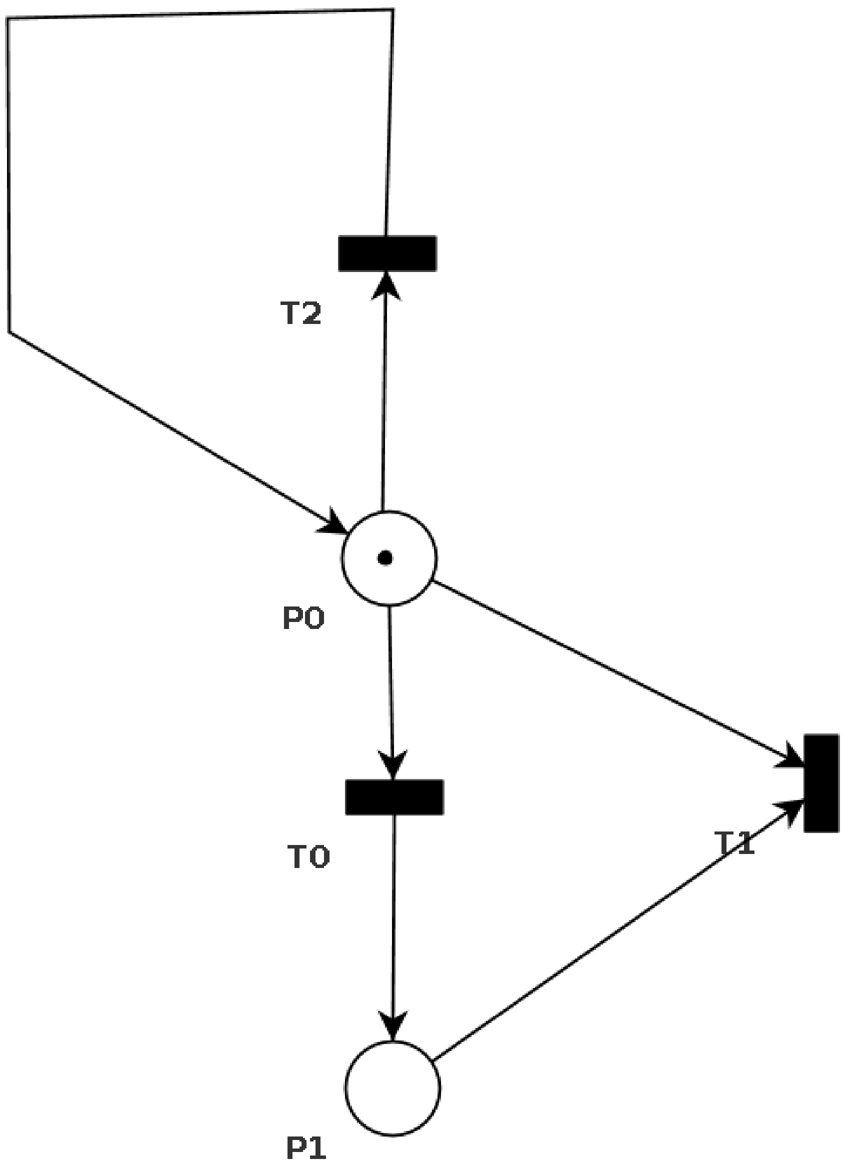


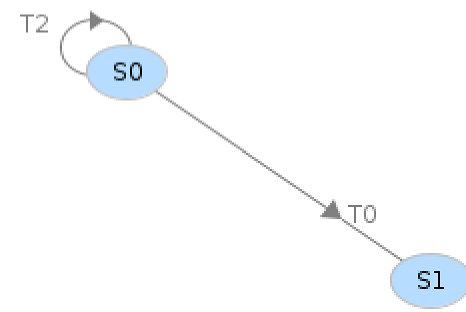
Liveness example



T1 is L1-live
T2 is dead
T3 is L3-live, not L4-live

Liveness example





S0 = [1, 0]
S1 = [0, 1]

T1 is L1-live
T2 is dead
T3 is L3-live, not L4-live

State Reachability

- A state $\mathbf{x}$ in a Petri net is *reachable* from a state $\mathbf{x}_0$ if there exists a sequence of transitions starting at $\mathbf{x}_0$ such that the state eventually becomes $\mathbf{x}$.
- Build/use reachability graph.
- Deadlock avoidance is a special case of reachability.

State Coverability

- In a Petri net with initial state $\mathbf{x}_0$, a state $\mathbf{y}$ is *coverable* if there exists a sequence of transitions starting at $\mathbf{x}_0$ such that the state eventually becomes $\mathbf{x}$ and $x(p_i) \geq y(p_i)$.
- Related to L1-liveness: *minimum number of tokens required* to enable a transition.

Persistence

- More than one transition enabled by the same set of conditions (choice, undeterminism).
- If one fires, does the other remain enabled ?
- A Petri net is *persistent* if, for any two enabled transitions, the firing of one cannot disable the other.
- Non-interruptedness (of multiple processes).

Language Recognition

Language defined by Petri net

$\equiv$

set of transition sequences which can fire

Fairness

Time

Colour

Coverability Notation

- Root node
- Terminal node
- Duplicate node

Coverability Notation

- Node *dominance*

$$\mathbf{x} = [x(p_1), x(p_2), \ldots, x(p_n)]$$

$$\mathbf{y} = [y(p_1), y(p_2), \ldots, y(p_n)]$$

$\mathbf{x} >_d \mathbf{y}$ ($\mathbf{x}$ dominates $\mathbf{y}$)if

1. $x(p_i) \geq y(p_i), \forall i \in \{1, \ldots, n\}$
2. $x(p_i) > y(p_i)$ for at least some $i \in \{1, \ldots, n\}$

- The symbol ω represents *infinity*

$$\mathbf{x} >_d \mathbf{y}$$

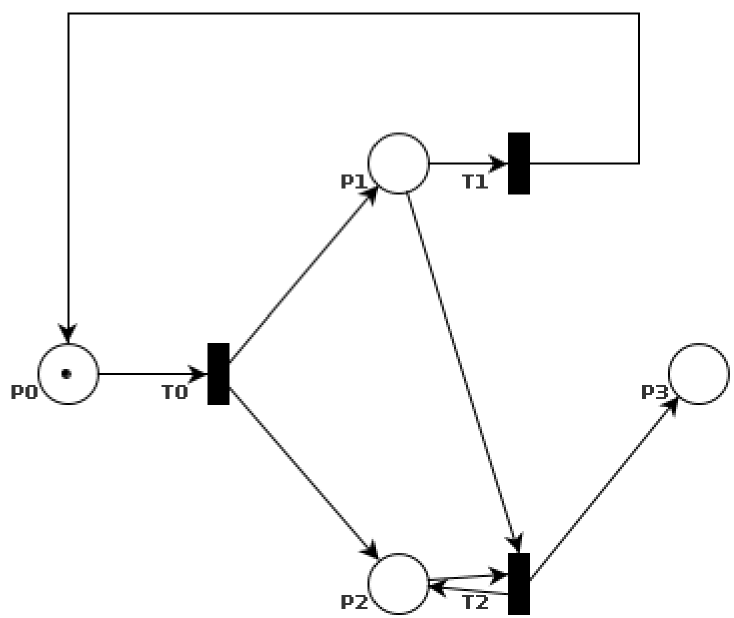
For all i such that $x(p_i) > y(p_i)$, replace $x(p_i)$ by ω

$$\omega + k = \omega = \omega - k$$

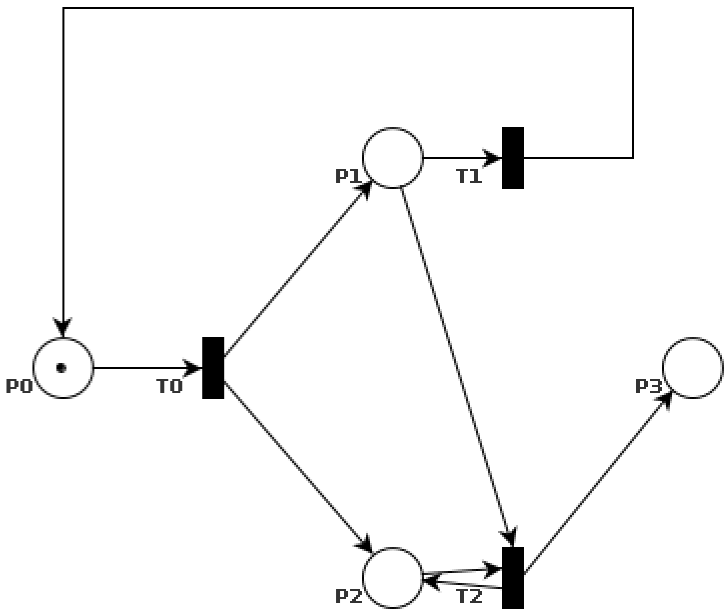
Coverability Tree Construction

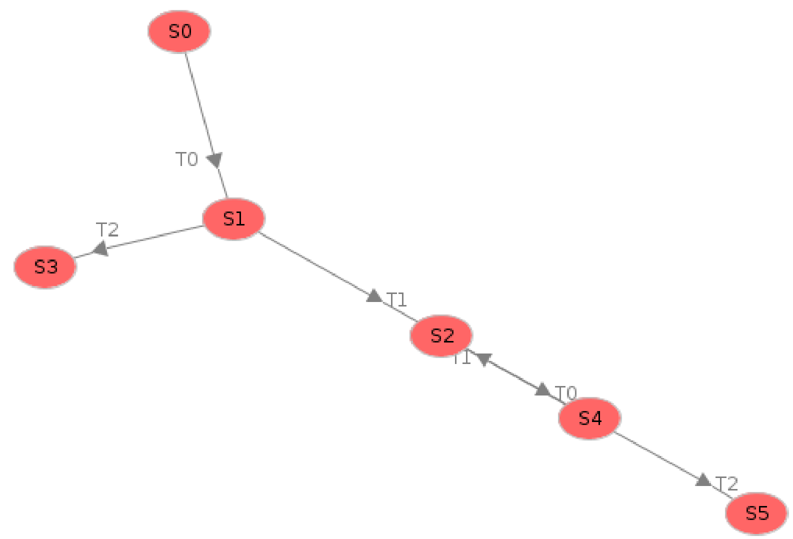
1. Initialize $\mathbf{x} = \mathbf{x}_0$ (initial state)
2. Fore each new node $\mathbf{x}$,
   evaluate the transition function $f(\mathbf{x}, t_i)$ for all $t_j \in T$:
   (a) if $f(\mathbf{x}, t_j)$ is undefined for all $t_j \in T$, then $\mathbf{x}$ is a terminal node.
   (b) if $f(\mathbf{x}, t_j)$ is defined for some $t_j \in T$,
   create a new node $\mathbf{x}' = f(\mathbf{x}, t_j)$.
      i. if $x(p_i) = \omega$ for some p_i, set $x'(p_i) = \omega$.
      ii. If there exists a node $\mathbf{y}$ in the path from root node $\mathbf{x}_0$ (included) to $\mathbf{x}$ such that $\mathbf{x}' >_d \mathbf{y}$, set $x'(p_i) = \omega$ for all p_i such that $x'(p_i) > y(p_i)$
      iii. Otherwise, set $\mathbf{x}' = f(\mathbf{x}, t_j)$.
3. Stop if all new nodes are either *terminal* or *duplicate*

Coverability Example



Coverability Example





Applications of the Coverability Tree

- Boundedness: ω does not appear in coverability tree
- Bounded Petri net $\Rightarrow$ reachability graph
- Conservation: $\gamma_i = 0$ for ω positions
- Inverse problem: what are γ and C ?
- Coverability: inspect coverability tree
- Limitations: deadlock detection

Path Conditions: LTL and CTL