Dynamic properties of Petri nets

dr. Tamás Bartha

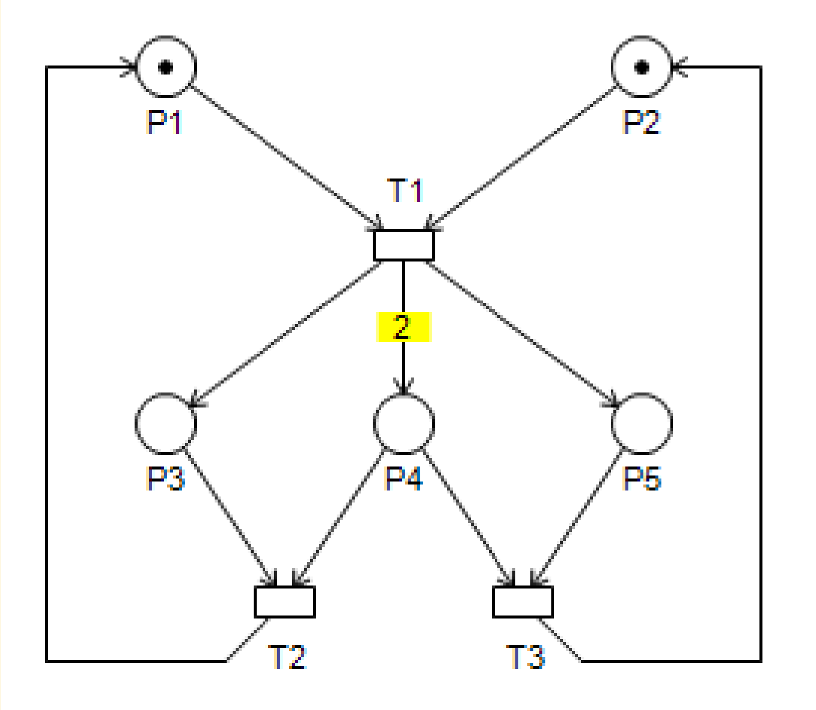
dr. István Majzik

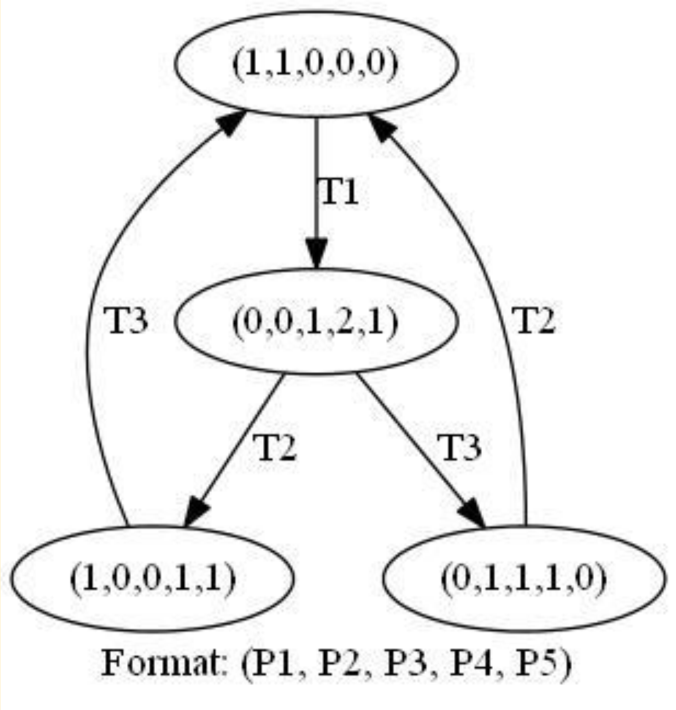
dr. András Pataricza

BME Department of Measurement and Information Systems

Petri net analysis methods:
An overview

Recall: Behavior of Petri nets





Simple Petri net with changing marking
(reachability graph of possible states)

Analysis methods

Depth of the analysis:

- Simulation ⬅ Traverse single trajectories
- Full exploration of state space ⬅ Traverse all trajectories from a given initial state (exhaustive traversal)
  - Analysis of reachability graph: Dynamic (behavioral) properties
  - Model checking
- Analysis of the net structure ⬅ Properties independent from the initial state (hold for every initial state)
  - Static analysis: Structural properties
  - Invariant analysis

if none of the above works

⬇

- Partial decision (e.g. abstraction)

Dynamic and structural properties

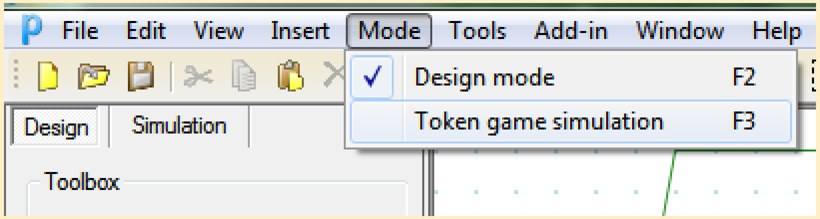
- Dynamic properties based on the reachability graph
  - Depend on the initial marking (not generalizable)
  - Typical properties (see later): Reachability, coverability, liveness, deadlock freedom, boundedness, fairness, reversibility
  - Property preserving reduction techniques can support the analysis
- Structural properties based on the (unmarked) net
  - Independent from the initial marking: hold for each (possible) behavior
  - Typical properties (see later): Structural liveness, structural boundedness, controllability, conservativeness, repetitiveness, consistency
  - Invariants: T-invariants (for transitions),
P-invariants (for places)

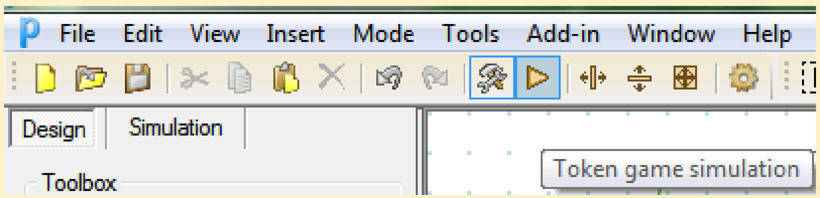
Simulation of Petri net models

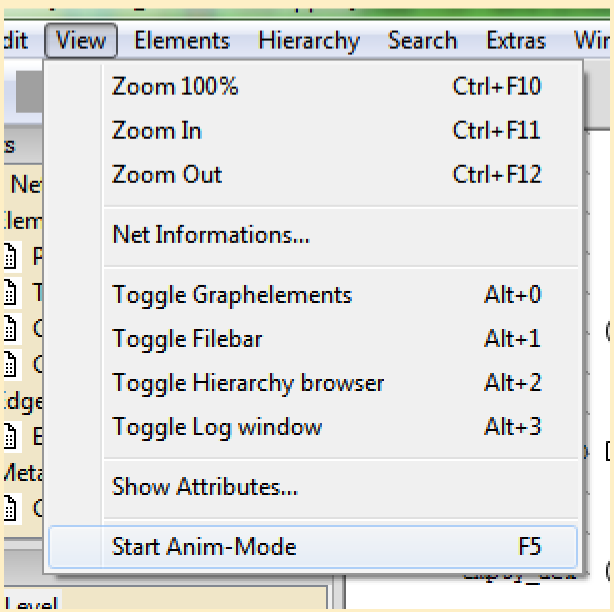
Simulation of discrete systems

- Goal: “realistic” modeling of the examined system
- Simulation for process models
  - Event oriented: Beginning and end of activities
  - Only the time moment of events is recorded
- Simulation of Petri nets
  - Examining the possible trajectories
    - State: token distribution (marking)
    - Change of state (event): firing of a transition
    - Trajectories in the state space: firing sequences
  - Petri nets are non-deterministic
    - (Pseudo) random choice is needed
    - Interactive simulation (token game): User chooses

Animation (token game)

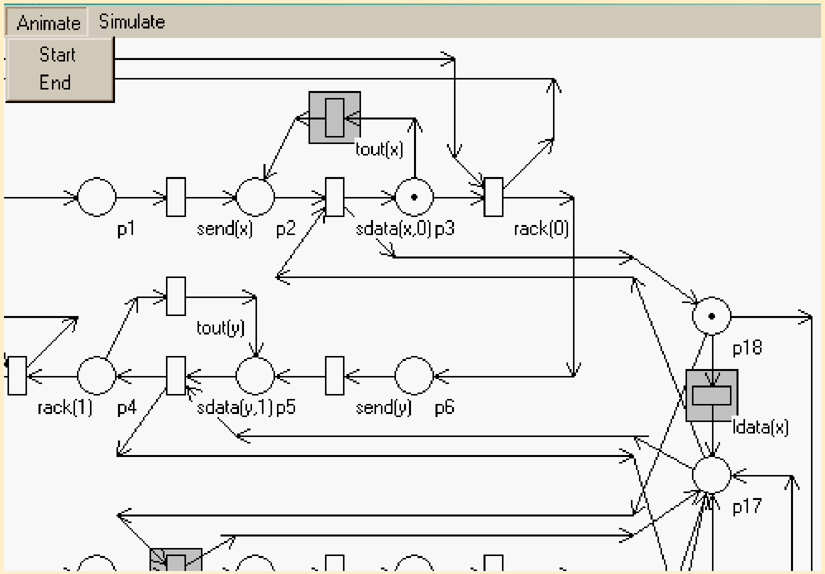


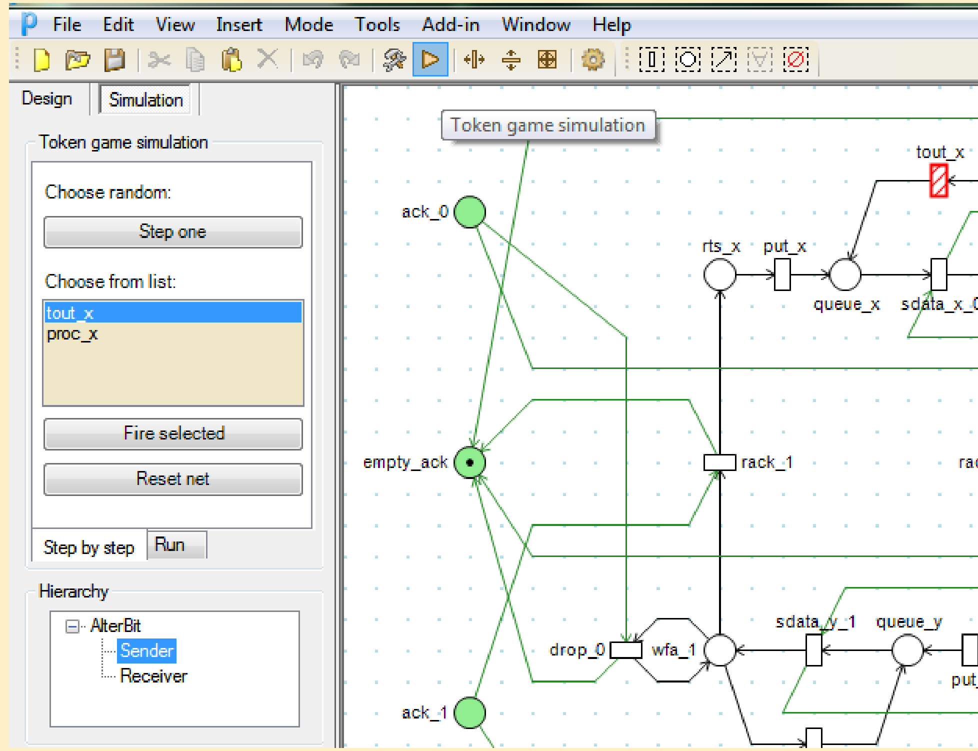


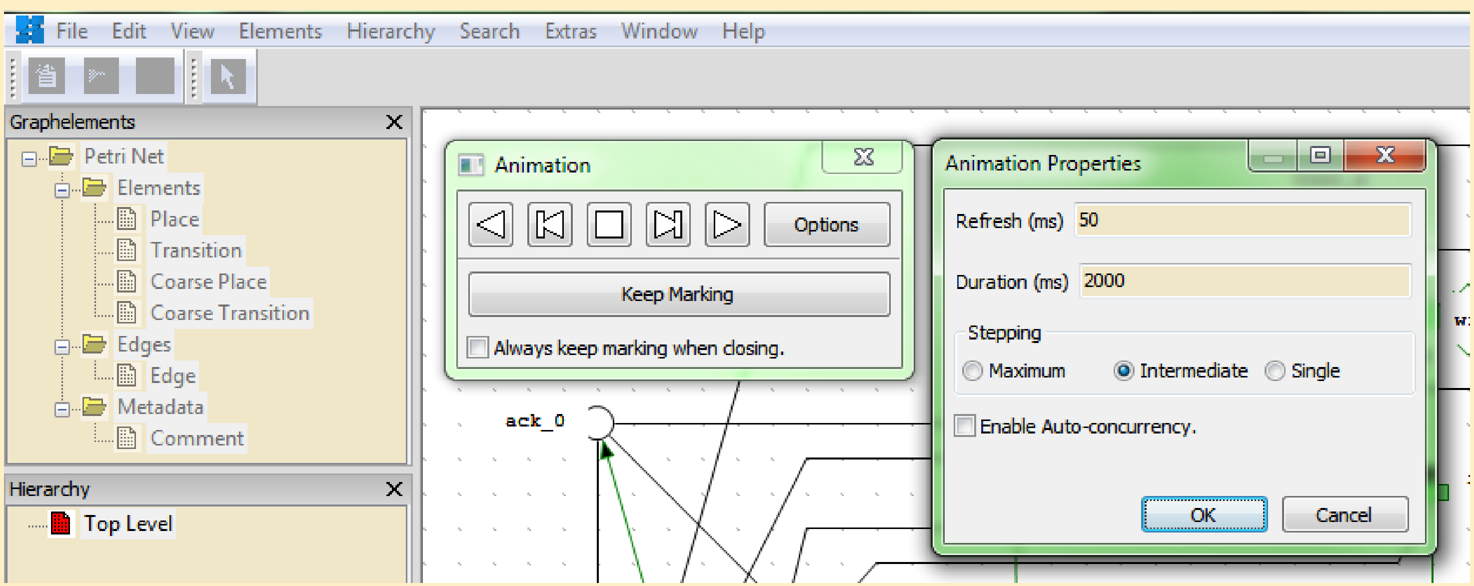


- Interactively examining the model
  - Enabled transitions are highlighted
  - Fire transition by clicking
  - New marking is shown
- Concurrent transitions
  - Manual choice
  - Automatic random choice (e.g. PetriDotNet)
- Original marking is restored in the end

Animation screen

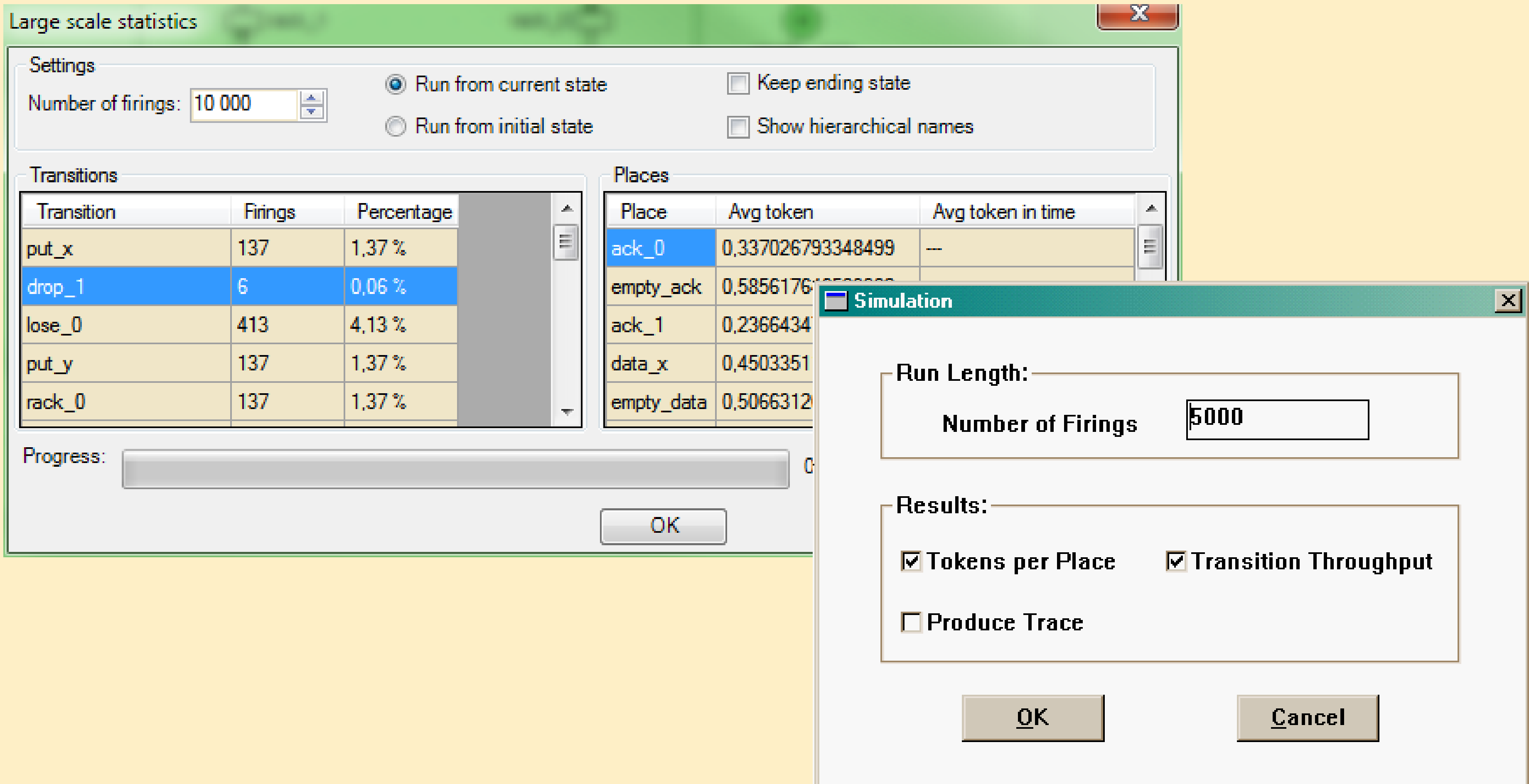






Simulation

- Setting the number of steps (transitions)
- Collecting statistics



Simple simulation algorithm

```
while (true) do
    Collect fireable transitions
    if (There are fireable transitions)
      then Choose a fireable transition (non-deterministic)
      else End simulation
    Fire chosen transition
end while
```

Collecting fireable transitions

function $collect\_fireable\_transitions(M)$

// Set of fireable transitions

$L_{fireable} \leftarrow \varnothing$

for all $t \in T$ **do**

if $enabled(t, M)$ **then** $L_{fireable} \leftarrow L_{fireable} \cup \{t\}$

return $L_{fireable}$

end function

Change of state

If t fires under marking M

- New marking: $M' = M + \mathbf{W}^T \cdot e_t$
  - where e_t is a unit vector corresponding to transition t
- Here **W** is the weighted incidence matrix
  - $\mathbf{W} = [w(t, p)]$ ← change in marking of p if t fires
  - Dimensions: $\tau \times \pi = |T| \times |P|$ ← rows × columns
  - When t fires, the number of tokens in p changes:

$$w(t, p) = \begin{cases} w^+(t, p) - w^-(p, t) & \text{if } (t, p) \in E \text{ or } (p, t) \in E \\ 0 & \text{if } (t, p) \notin E \text{ and } (p, t) \notin E \end{cases}$$

Simulation algorithm

// Initialization

$M \leftarrow \mathrm{M}_0$

$L_{fireable} \leftarrow collect\_fireable\_transitions(M)$

`while` $L_{fireable} \neq \varnothing$ **`do`**

    // Firing

    $t \leftarrow rnd(L_{fireable})$

    $M' \leftarrow M + \mathbf{W}^{\mathbf{T}} \cdot \underline{\mathrm{e}}_t$

    $L_{fireable} \leftarrow collect\_fireable\_transitions(M')$

    $M \leftarrow M'$

`end while`

Idea for improving efficiency

- Why check all transitions ($|T|$ steps), if only the surroundings of the previously fired transition ($\bullet t \cup t\bullet$) changes?
  - Some transitions will be disabled
  - Some transitions will be enabled

Possibly disabled transitions

- After firing t, a transition t' can become disabled
  - By having an input in $\bullet t$, i.e., t "consumes its tokens"
  - By being in conflict with t: $\bullet t' \cap \bullet t \neq \emptyset$
- Calculating numerically
  - Number of consumed tokens: $M^{-} = \mathbf{W}^{-T} \cdot \underline{e}_t$
  - Input places of t: $\bullet t$, i.e., $\{p \in P: M^{-}(p) > 0\}$
  - Possibly disabled by t: $T' = \{(\bullet t)\bullet\}$

Possibly enabled transitions

- After firing t, a transition t' can become enabled
  - By having an input in $t\bullet$, i.e., t "produces tokens"
  - t enables t': $\bullet t' \cap t\bullet \neq \emptyset$
- Calculating numerically
  - Tokens produced: $M^{+} = \mathbf{W}^{+\mathbf{T}} \cdot \underline{e}_t$
  - Output places of t: $t\bullet$, i.e., $\{p \in P: M^{+}(p) > 0\}$
  - Possibly enabled by t: $T'' = \{(t\bullet)\bullet\}$
- It is sufficient to check these transitions only (that can become disabled or enabled)!

Efficient algorithm: Initialization

- Initialization is the same

// Initialization

$M \leftarrow \mathrm{M}_0$

$L_{fireable} \leftarrow \varnothing$

// Set of initially fireable transitions

for all $t \in T$ **do**

  if $enabled(t, \mathrm{M}_0)$ **then** $L_{fireable} \leftarrow L_{fireable} \cup \{t\}$

Efficient algorithm: Firing loop

while $L_{fireable} \neq \emptyset$ **do**

// Firing

$t \leftarrow rnd(L_{fireable})$

$M' \leftarrow M + \mathbf{W}^{T} \cdot \underline{e}_t$

// Remove newly disabled transitions

for all $t' \in \{(\bullet t)\bullet\}$ **do**

if $not(enabled(t', M'))$ **then** $L_{fireable} \leftarrow L_{fireable} \setminus \{t'\}$

// Add newly enabled transitions

for all $t'' \in \{(t\bullet)\bullet\}$ **do**

if $enabled(t'', M')$ **then** $L_{fireable} \leftarrow L_{fireable} \cup \{t''\}$

$M \leftarrow M'$

end while

Priority

- Extended firing rule: a transition t can fire iff
  - It is enabled and
  - No transition is enabled with higher priority than $\pi(t)$
- Consequence:
  - $L_{fireable}$ is not a set, but a vector $L_{fireable}[\pi]$ of sets ordered by priority levels $\pi \in \Pi$
  - A transition is chosen non-deterministically from the highest priority non-empty set of $L_{fireable}[\pi]$

Algorithm with priorities: Initialization

// Initialization

$M \leftarrow \mathrm{M}_0$

`for all` $\pi \in \Pi$ **`do`**

    $L_{fireable}[\pi] \leftarrow \varnothing$

// Set of initially fireable transitions

`for all` $t \in T$ **`do`**

    `if` $enabled(t, \mathrm{M}_0)$ **`then`** $L_{fireable}[\pi(t)] \leftarrow L_{fireable}[\pi(t)] \cup \{t\}$

Algorithm with priorities: Firing loop

```
while ⋃_{π∈Π} L_fireable[π] ≠ ∅ do
    for π = π_max to π_min step -1 do      // Firing (with priority)
        if L_fireable[π] ≠ ∅ then
            t ← rnd(L_fireable[π])
            M' ← M + W^T·e_t
            exit for
        end if

    for all π ∈ Π do                          // Enabled/disabled transitions
        for all t' ∈ {(•t)•} do
            if not(enabled(t', M') then L_fireable[π(t')] ← L_fireable[π(t')] \ {t'}
        for all t'' ∈ {(t•)•} do
            if enabled(t'', M') then L_fireable[π(t'')] ← L_fireable[π(t'')] ∪ {t''}
    end for

    M ← M'
end while
```

Reachability analysis

Reachability

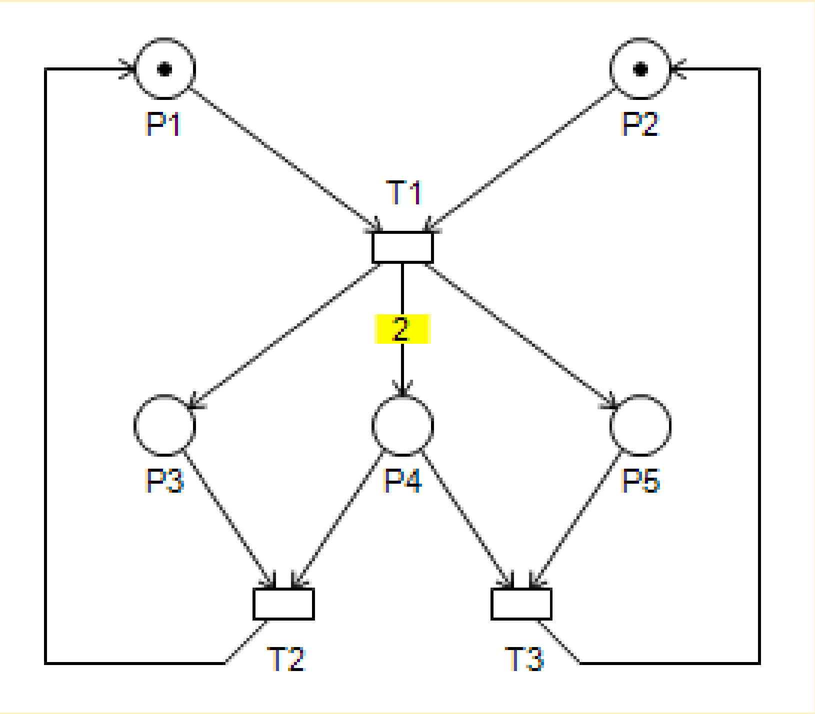
- Reachability analysis
  - Dynamic behavior depending on the initial marking
    - Marking = state

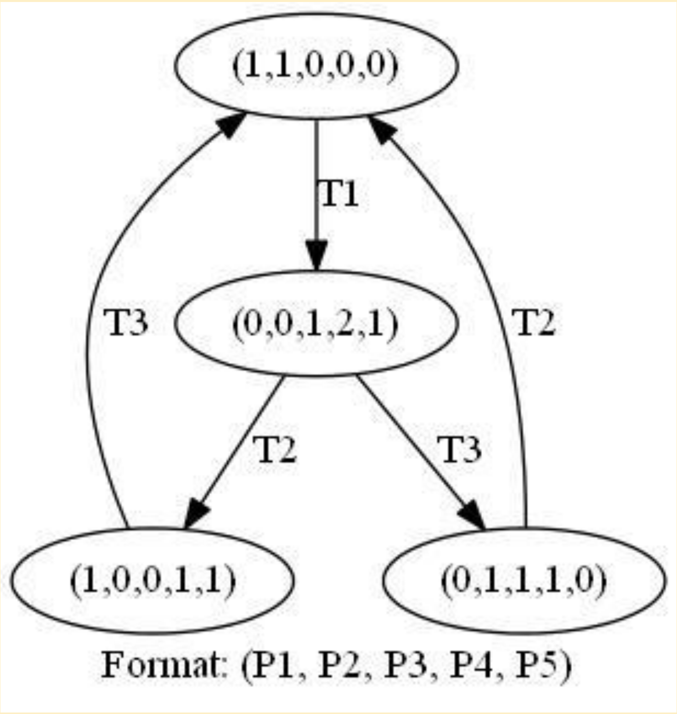
      Token distribution = value of state variable
    - Firing = transition
    - Sequence of states $M_0, M_1, \ldots, M_n$ for a firing sequence
  - State sequence: trajectory in the state space
  - A state M_n is *reachable* from initial state M_0 if

$$\exists\vec{\sigma} : M_0[\sigma > M_n$$

  - Reachability graph: graphical representation of the state space

Example: Reachability graph





A simple Petri net with its reachability graph (exported from PetriDotNet tool)

Reachability analysis

- From the initial state M_0 of a Petri net N
  - Reachable states are:

    $$R(N, M_0) = \{\, M \mid \exists \vec{\sigma} : M_0 \, [\vec{\sigma} > M \,\}$$

    Can answer state-based queries

  - Executable firing sequences are:

    $$L(N, M_0) = \{\, \vec{\sigma} \mid \exists M : M_0 \, [\vec{\sigma} > M \,\}$$

    Can answer transition-based (event-based) queries

Reachability problem

- Reachability problem of Petri nets:
  - Is the marking M_n reachable from an initial marking M_0

$$M_n \overset{?}{\in} R(N, M_0)$$

- Submarking reachability problem:
  - Restricting the question to a subset $P' \subset P$ of the places, i.e., whether M_n with a token distribution for the given subset of places is reachable:

$$\overset{?}{\exists} M \in R(N, M_0) : \forall p \in P' : M(p) = M_n(p)$$

Decidability of the reachability problem

- The reachability problem is decidable
  - But has exponential (space) complexity in general

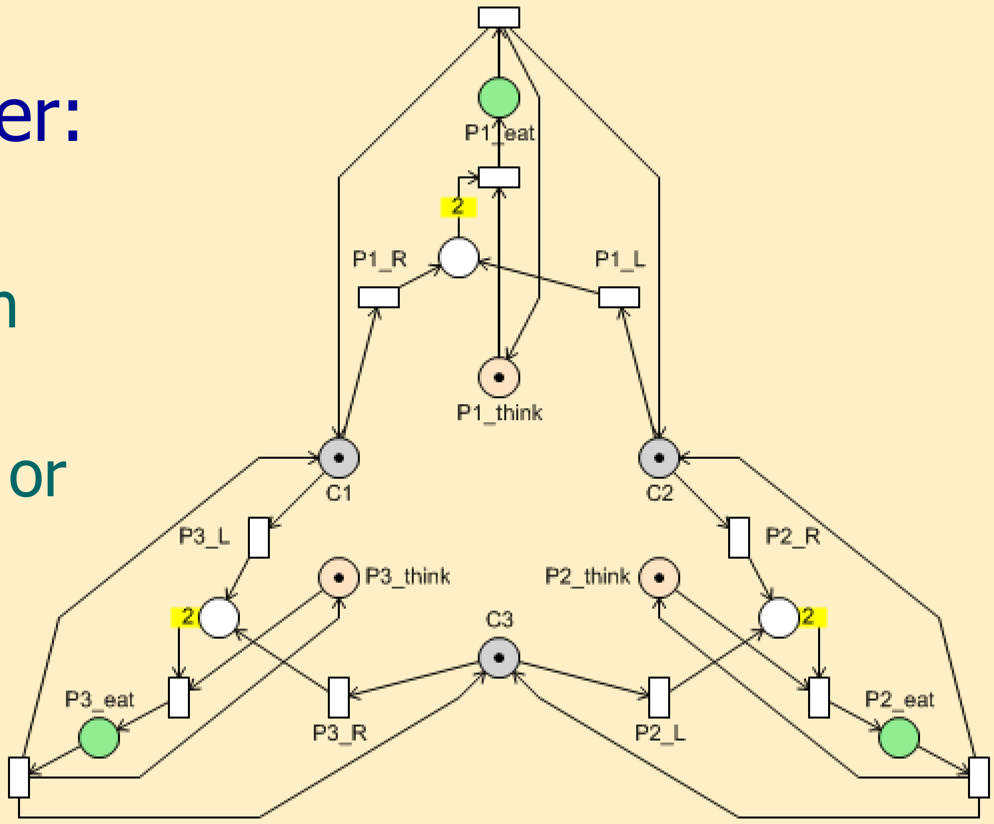
- In contrast the equality problem is not decidable in general
  - Task: checking the equivalence of the possible firing sequences of two Petri nets (N, N')

$$L(N, M_0) \stackrel{?}{=} L(N', M'_0)$$

  - Exponential algorithm for 1-bounded (safe) Petri nets
    - Bisimulation: can simulate each other

Model checking Petri nets

- Dining philosophers
- For a single philosopher:
  - Can eat at least once?
  - Will eat at least once in any case?
  - Will always eat sooner or later?
- For the whole model:
  - Deadlock freedom?



Dynamic (behavioral) properties of Petri nets

Dynamic properties

- Reachability-based properties
  - Depend on the initial marking (state)

    (Cf.: structural properties are independent from the initial marking!)
  - Can be determined not only with reachability analysis
- Dynamic properties (overview):

1. Boundedness
2. Liveness
   - Deadlock freedom
3. Reversibility
4. Home state
5. Coverability
6. Persistency
7. Fairness
   - Bounded fairness
   - Global fairness

1. Boundedness

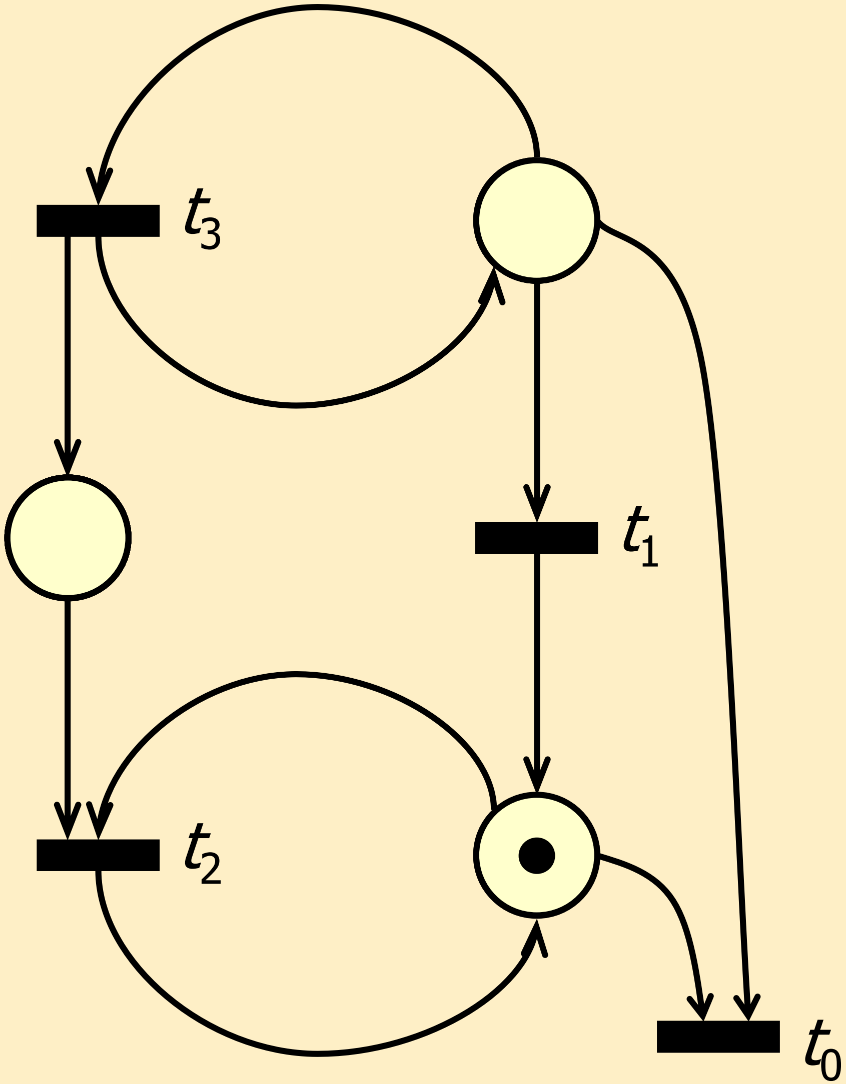
- *k*–boundedness (boundedness)
  - In each state each place contains maximum *k* tokens (depends from the initial marking M_0!)
  - Safe Petri net: special case of boundedness ($k = 1$)
  - Modeling "finiteness"
    - Boundedness ⇔ finite state space
- Practical queries that can be answered
  - Do tasks accumulate in a system?
  - Are messages processed periodically?

2. Liveness for transitions

- Deadlock freedom of a net
  - There is at least one enabled transition in each state
- Liveness property: More general
  - Can a transition fire once/many times/infinite times?
  - Weak liveness properties for a transition t:
    - L0-live (dead): t can never fire in a
    - L1-live: t can fire at least once in some
    - L2-live: for each finite integer $k > 1$, t can fire at least k times in some
    - L3-live: t can fire infinitely many times in some

    } trajectory $\vec{\sigma} \in L(N, M_0)$
  - L4-live: t is L1-live in each $M_n \in R(N, M_0)$ marking

Liveness: Example



- Transition t_0: L0-live (dead)
- Transition t_1: L1-live
- Transition t_2: L2-live
- Transition t_3: L3-live

Liveness for Petri nets

- A Petri net (PN, M_o) is Lx-live
  - If every transition $t \in T$ is Lx-live
  - Liveness properties contain each other from L4 to L1
- A Petri net (PN, M_o) is live
  - If it is L4-live, i.e., every transition $t \in T$ is L4-live
    - L4-live: L1-live (can fire at least once in some trajectory) in every reachable state
  - Deadlock freedom guaranteed independently from trajectory
    - Each transition can be fired again, independently from the intermediate states
    - Deadlock freedom $\Leftarrow$ liveness
  - Can be proven expensively
    - In lucky cases it is not expensive (see invariants later!)

3. Reversibility

- Reversibility
  - Initial state can be reached from every reachable state

$$\forall M \in R(N, M_0) : M_0 \in R(N, M)$$

- Practical examples:
  - Cyclical behavior of network through initial state
  - The system can be “resetted” to initial state
  - The safe initial state can be reached from anywhere

4. Home state

- Home state
  - A reachable state that can be reached from every state reachable from it

$$\exists M_n \in R(N, M_0) : \forall M \in R(N, M_n) : M_n \in R(N, M)$$

- Practical examples:
  - Cyclical behavior after initialization period
  - A safe state can be reached anytime after initialization

5. Coverability

- Coverability
  - Can a state covering previous behavior occur?
  - State M′ covers state M if: $M' \in R(N, M_0) \wedge M' \geq M$
    - Reverse: State M can be covered with state M'
    - Meaning of $M' \geq M$: $\forall p \in P : \mathrm{m}'(p) \geq \mathrm{m}(p)$
      - Weak coverability: cover identical states
      - Strong coverability: $\exists p \in P : \mathrm{m}''(p) > \mathrm{m}(p)$
- Relationship with liveness
  - If μ is the minimal marking enabling transition t
    - t is not L1-live if and only if, μ cannot be covered
    - reverse: coverability of μ guarantees t to be L1-live (can fire)

6. Persistence

- Persistence for transitions
  - A transition is persistent if after becoming enabled it remains enabled until it fires
  - I.e., no other transition can disable the transition by firing
- Persistence for Petri nets
  - A Petri net (PN, M_o) is persistent, if any two transitions $t_1, t_2 \in T$ are persistent in every possible firing sequence
- Practical examples:
  - Is the functional decomposition of a system working properly?
  - Do parallel behaviors interfere?

7. Fairness: Bounded fairness

- Two definitions for fairness
  - Bounded fairness (B-fairness)
  - Global fairness (unbounded fairness)

- Bounded fairness
  - A firing sequence is a bounded fair (B-fair) sequence
    - if any transition can fire only a bounded number of times without a different enabled transition being fired
  - A Petri net is a bounded fair (B-fair) net
    - if every possible firing sequence is bounded fair

Fairness: Global fairness

- Global fairness
  - A firing sequence is globally (unbounded) fair, if
    - it is finite, or
    - Contains every transition infinitely many times
  - A Petri net is a globally (unbounded) fair net
    - If all possible firing sequences of the net are globally fair
- Practical examples:
  - Do parallel processes block each other?
  - Do all processes (eventually) proceed?
  - Will a request eventually be served?

Dynamic properties (summary)

- Boundedness
- Deadlock freedom
- Liveness
  - L0 live (dead)
  - L1 live (can fire once)
  - L2 live (can fire k times)
  - L3 live (can fire ∞ times)
  - L4 live (L1 in every state)
- Reversibility
- Home state
- Coverability
  - Weak coverability
  - Strong coverability
- Persistence
- Fairness
  - Bounded fairness
  - Global fairness

State space representations:
Reachability and coverability graphs

State space representations: Reachability graph

- Reachability graph
  - State graph starting from initial marking M_0
    - Nodes: markings; labels: token distributions
    - Transitions: directed arcs; labels: firings
    - A node has as many successors (outgoing arcs) as the number of enabled transitions
      - Or less, if the net has priorities
    - Node with no outgoing arcs: deadlock
  - Unbounded Petri net $\rightarrow$ infinitely many states
    - Boundedness $\Leftrightarrow$ finite state space
  - Analysis: Breadth-first search from a state through transitions
    - Depth-first search is a bad idea in an infinite state space…

State space representations: Coverability graph

- Infinite state graph: token "overgrowth"
  - Where and "how" it becomes infinite?
  - What are the analysis possibilities?
- Coverability graph: works for infinite state space
  - Similar structure: initial marking M_0, arcs: firings
  - Trajectory: M_0 … M''… M'
    when $M'' \leq M'$, i.e., M'' is covered, i.e.,
    $p \in P : \mathrm{m}'(p) > \mathrm{m}''(p)$ are covered places (strong cov.)
  - Special symbol for covered places:
    ω, expressing infinity

Coverability tree generating algorithm

Building with graph nodes:

$L_{to\_be\_examined} \leftarrow \{ M_0 \}$

```
MAIN: if L_to_be_examined ≠ ∅
    Remove the next node M ∈ L_to_be_examined
    if M already occurred on the path from the root node
        then mark M as "old node"
        goto MAIN    // loop
    if no transition is enabled under M
        then mark M as "final node"
        goto MAIN    // loop
```

(continued on next page)

Coverability tree generating algorithm (cont.)

else // (there are enabled transitions under M)

for all enabled transition *t*:

Determine successor node *M′*: $M\ [\ \vec{e}_t > M'$

if an *M″* exists on path from M_0 to *M*, which is covered by *M′*

$$M' \neq M'' \wedge \forall p \in P : \mathrm{m}'(p) \geq \mathrm{m}''(p) \wedge \exists p \in P : \mathrm{m}'(p) > \mathrm{m}''(p)$$

then *M″* is a covered node:

markings of strongly covered places are replaced with ω in the token distribution of node *M′*

$$\forall p \in P : \mathrm{m}'(p) > \mathrm{m}''(p) \rightarrow \mathrm{m}'(p) = \omega$$

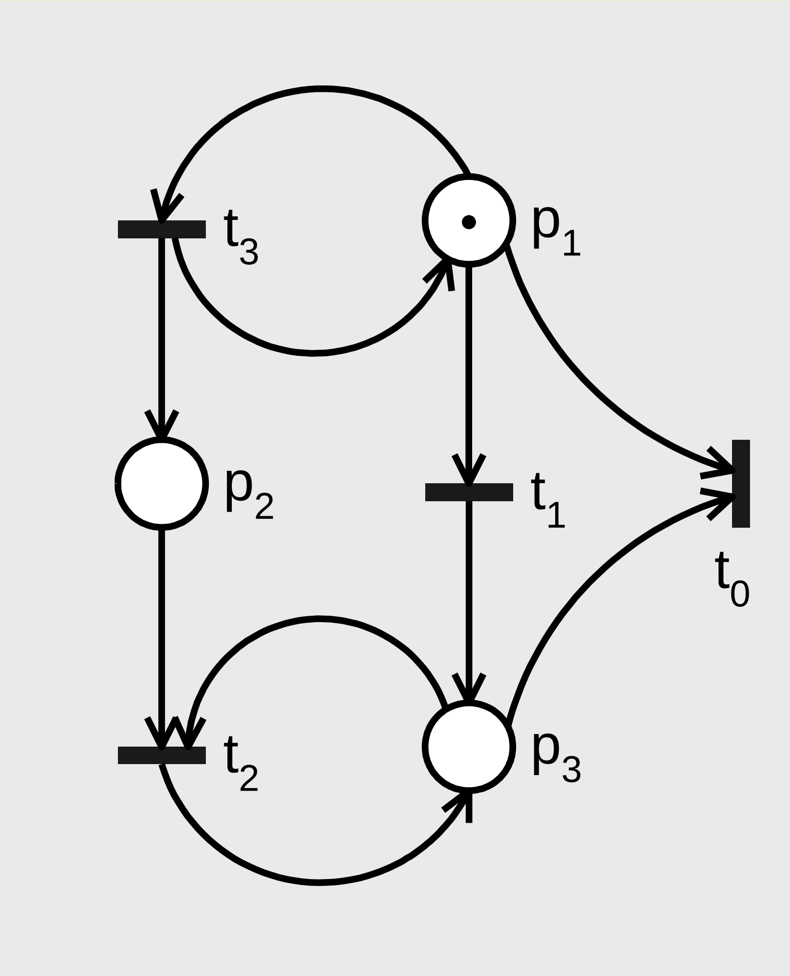
Add *M′*, to be examined: $L_{to\_be\_examined} \leftarrow L_{to\_be\_examined} \cup M'$

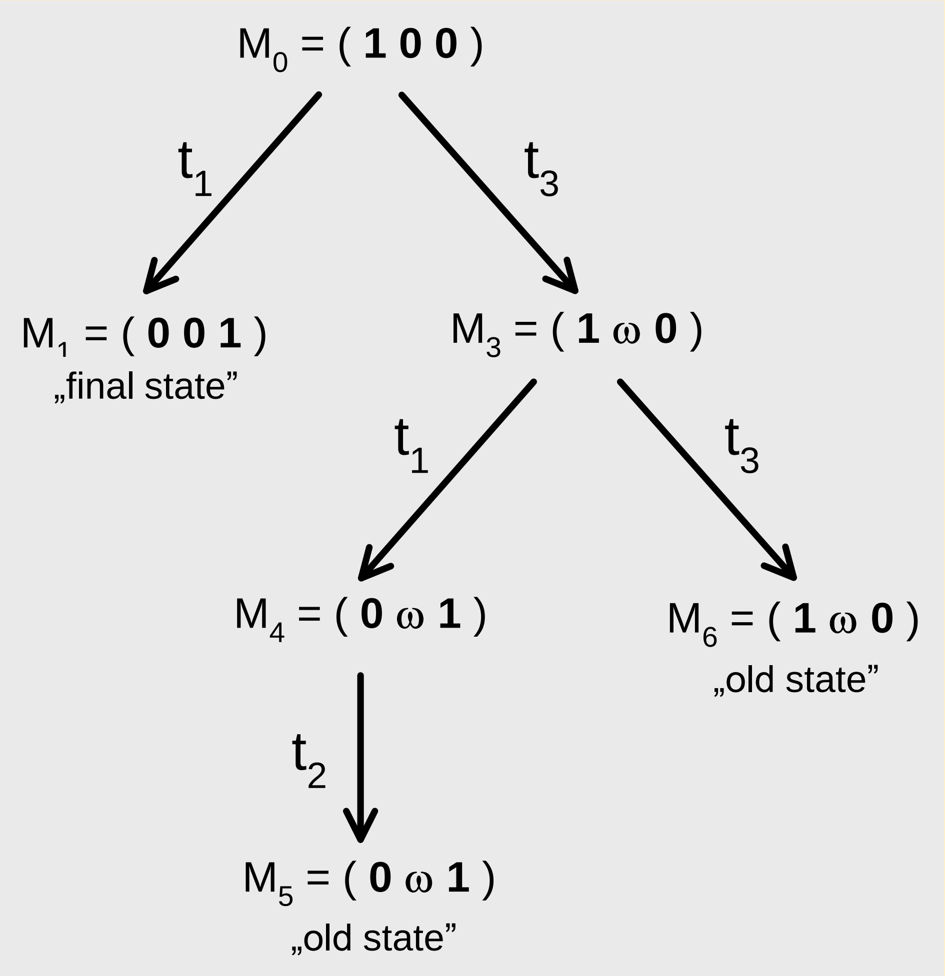
Draw an arc from *M* to *M′* marked with *t*

goto `MAIN` // loop

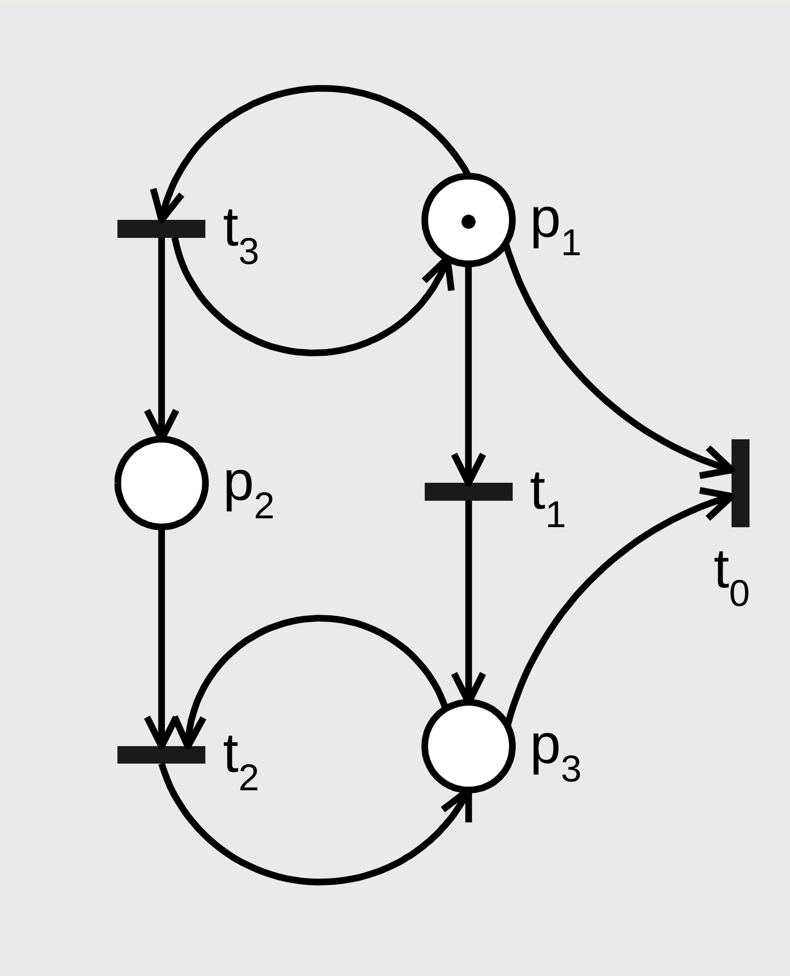
Coverability graph: join nodes denoting the same marking

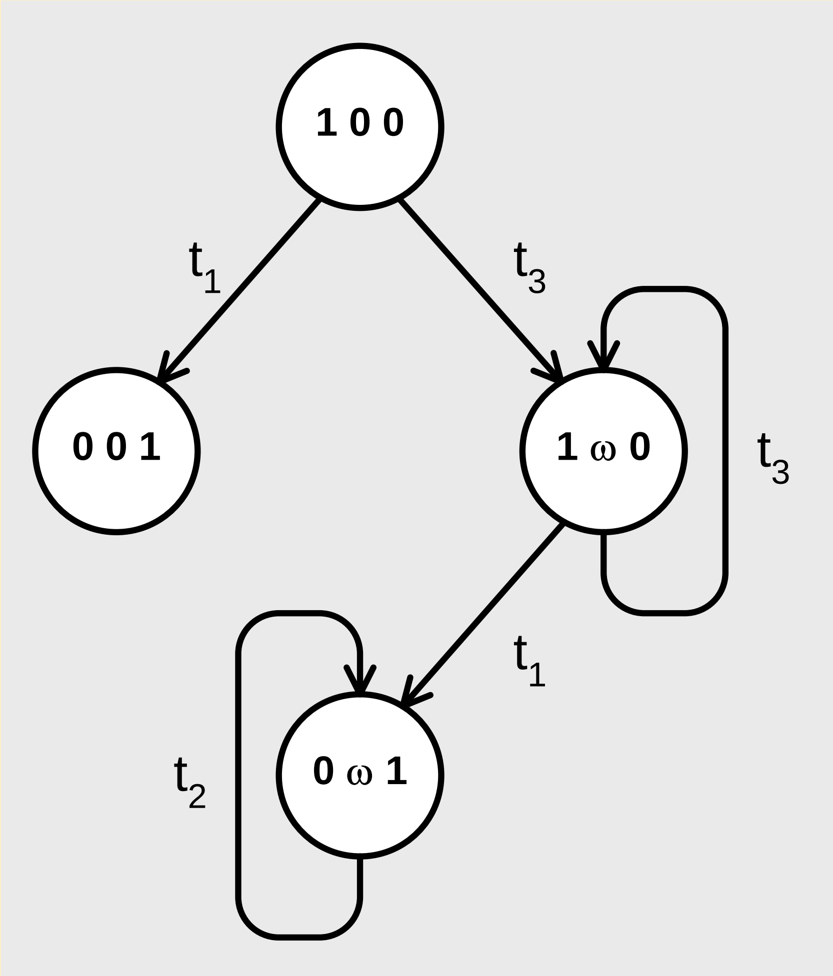
An example with coverability tree





An example with coverability graph





Analysis of the coverability graph

Observable properties:

- Bounded Petri net ⇔ Reachability graph $R(N, M_0)$ is finite ⇔ ω does not appear as a label in the coverability graph
- Safe Petri net ⇔ Only 0 and 1 appears as a label in the coverability graph
- A transition is dead ⇔ firing of the transition does not appear as an arc label in the coverability tree