Discrete Event Systems

- Petri Nets –

Lothar Thiele

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Technische Informatik und Kommunikationsnetze
Computer Engineering and Networks

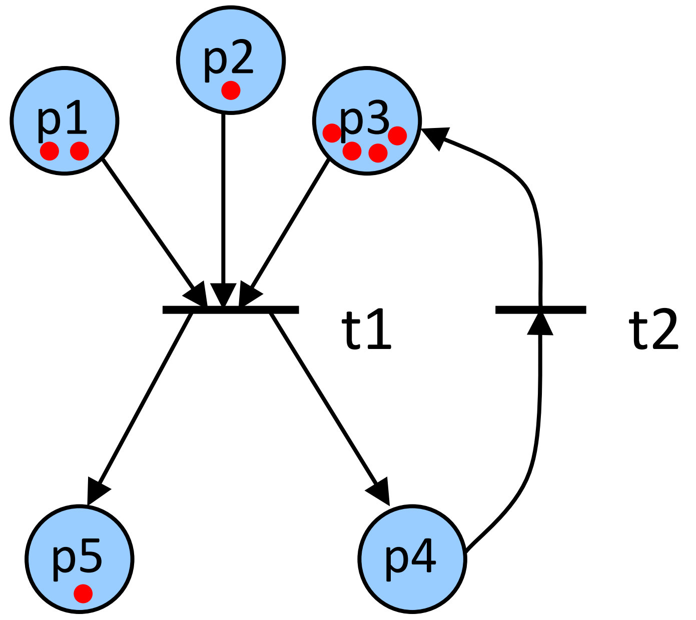
Petri Nets – Motivation

- In contrast to state machines, state transitions in Petri nets are asynchronous. The ordering of transitions is partly uncoordinated; it is specified by a partial order.
- Therefore, Petri nets can be used to model concurrent distributed systems.
- Many flavors of Petri nets are in use, e.g.
  - Activity charts (UML)
  - Data flow graphs and marked graphs
  - GRAFCET (programming language for programming logic controllers)
  - Specialized languages for workflow management and business processes
- Invented by Carl Adam Petri in 1962 in his thesis “*Kommunikation mit Automaten*”

Contents

Petri Net – Definition

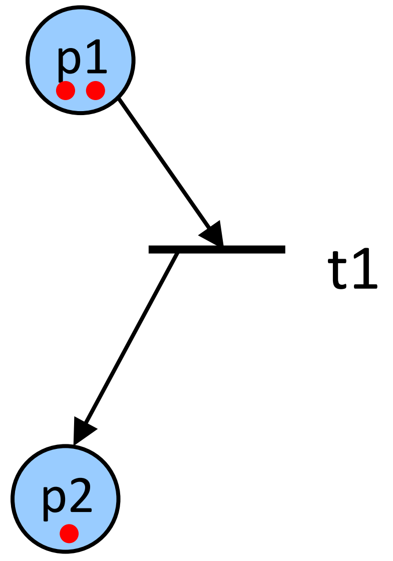
- A **Petri net** is a bipartite, directed graph defined by a 4-tuple (S, T, F, M_0), where:
  - S is a set of places p
  - T is a set of transitions t
  - F is a set of edges (flow relations) f, no parallel edges are allowed
  - $M_0 : S \to \mathbf{N}$; the initial marking



$\{p1, p2, p3, p4, p5\} \in$ S
$\{t1, t2\} \in$ T
$\{(p1, t1), (p2, t1), (t1, p5), \ldots\} \in$ F

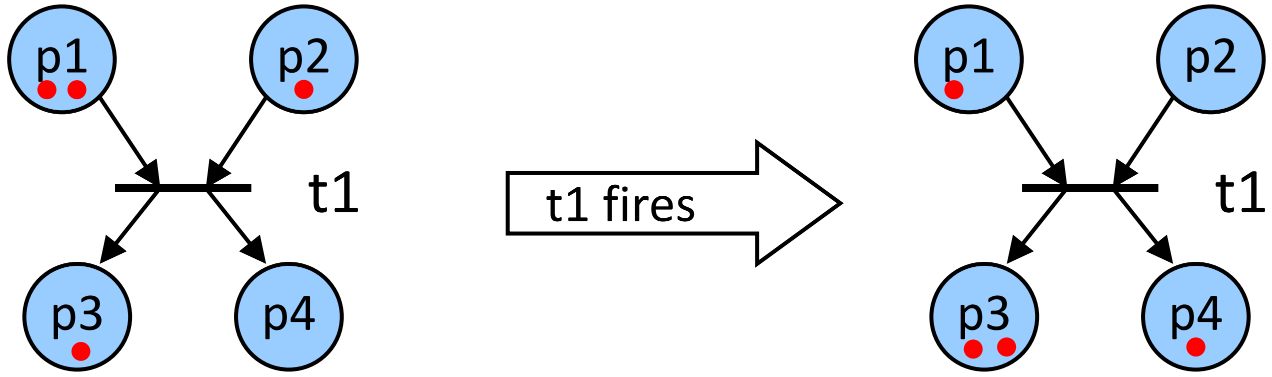
Token Marking

- Each place p_i is marked with a certain number of token.
- The initial distribution of the tokens is given by M_0 .
- M(s) denotes the marking of a place s.
- The distribution of tokens on places defines the state of a Petri net.
- The dynamics of a Petri net is defined by a token game.



Token Game of Petri Nets

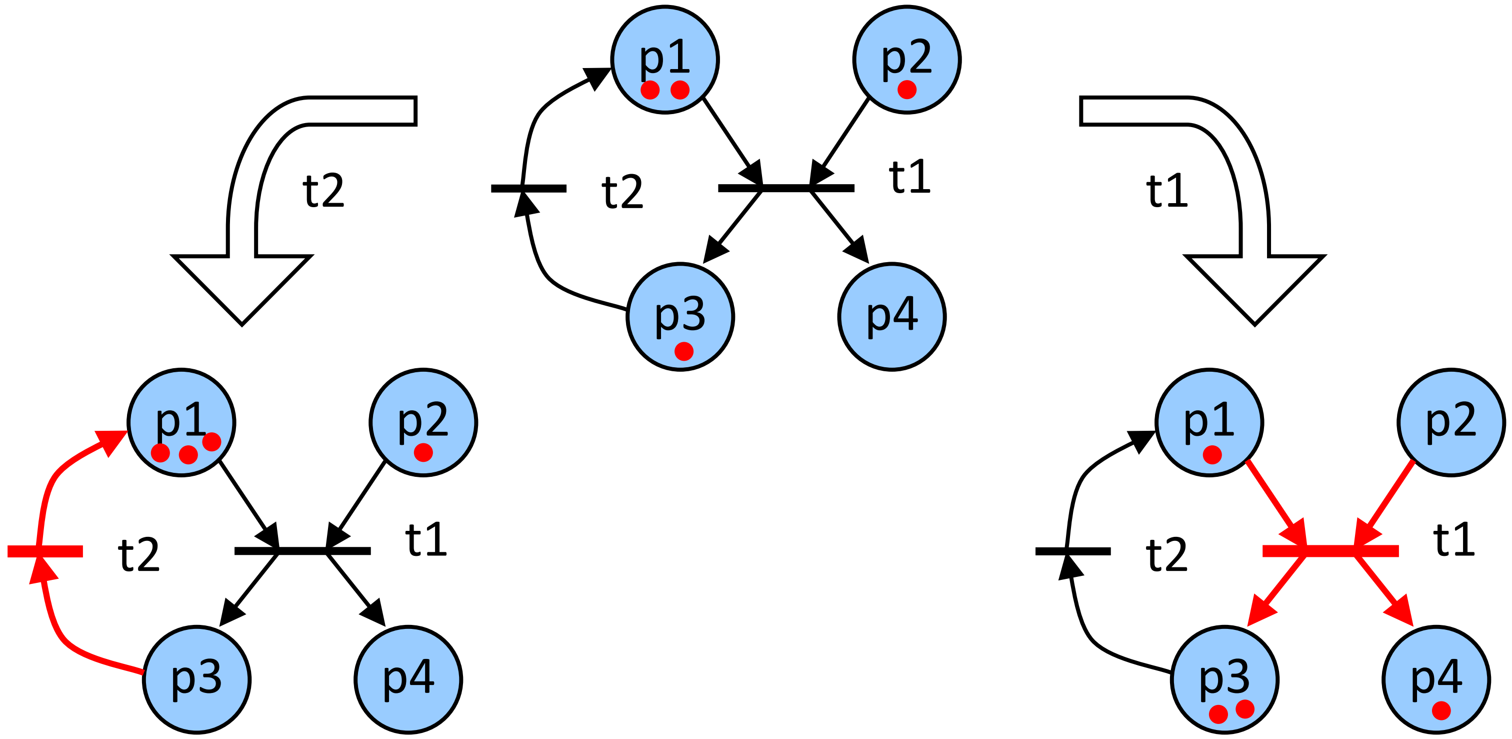
- A marking M activates a transition t ∊ T if each place p connected through an edge f towards t contains at least one token.
- If a transition t is activated by M, a state transition to M’ fires (happens) eventually.
- Only one transition is fired at any time.
- When a transition fires, it
  - consumes a token from each of its input places,
  - adds a token to each of its output places.



Non-Deterministic Evolution

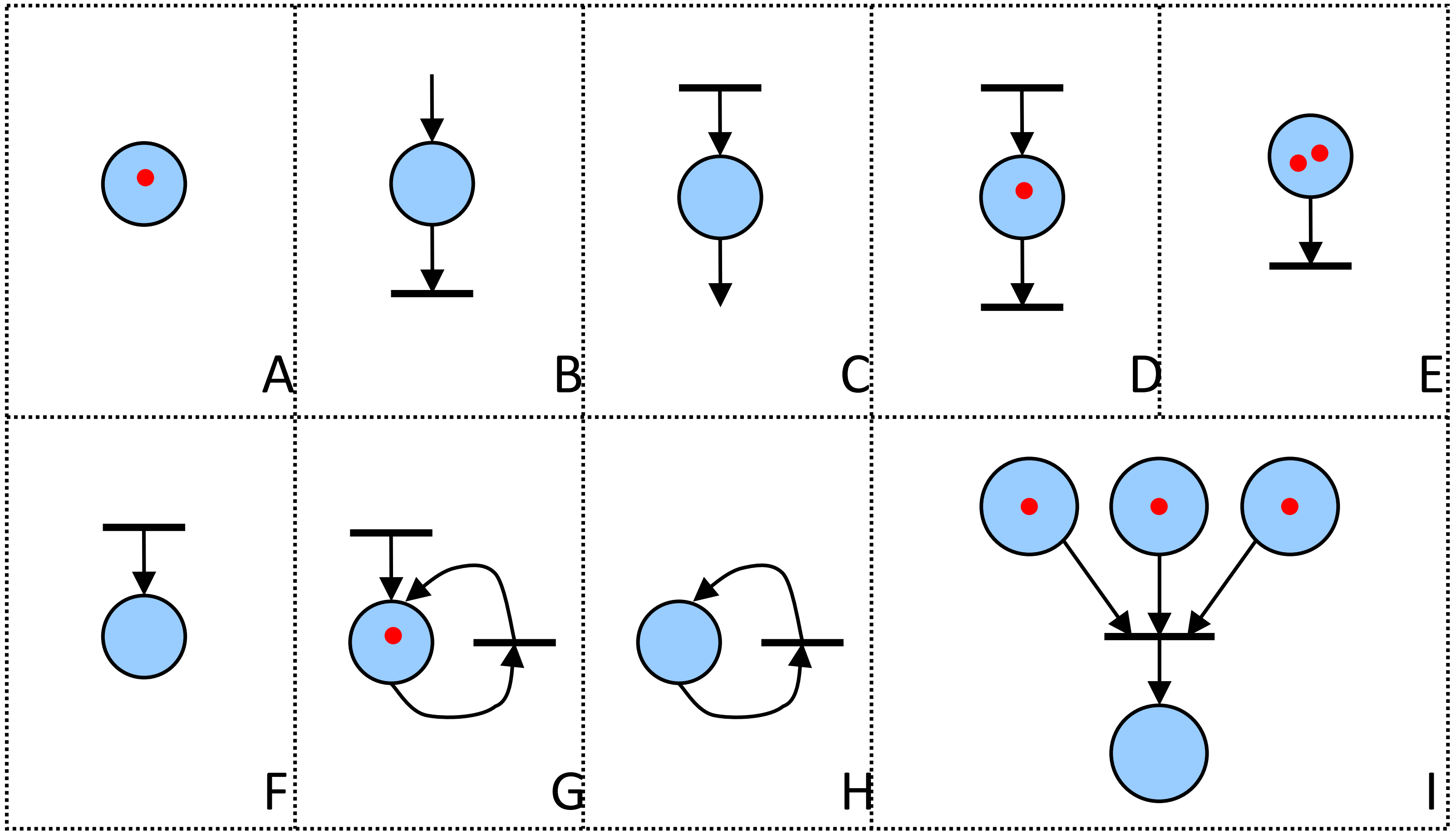
The evolution of Petri nets is not deterministic.

- Any of the activated transactions might fire:



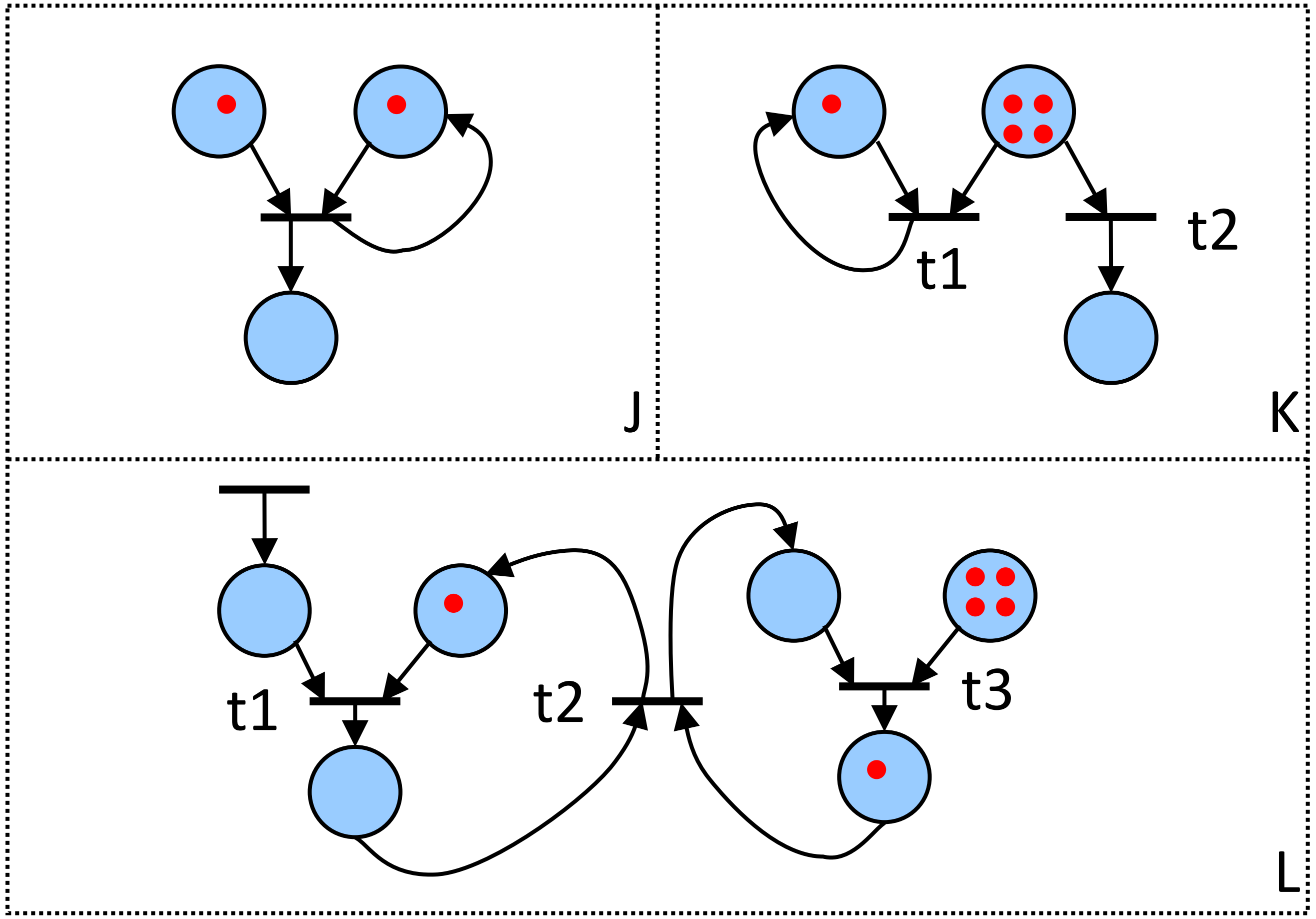
Syntax Exercise (1)

- Is it a valid Petri Net?
- Which transitions are activated?
- What is the marking after firing?



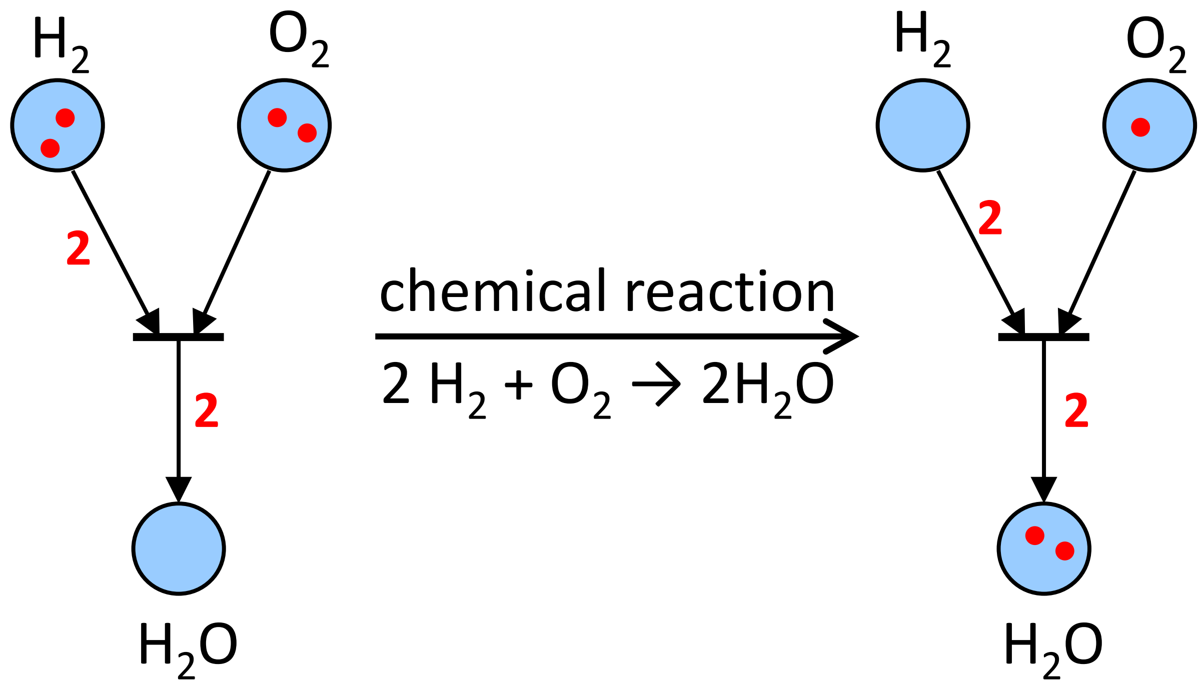
Syntax Exercise (2)

- Is it a valid Petri Net?
- Which transitions are activated?
- What is the marking after firing?



Weighted Edges

- Associating **weights** to edges:
  - Each edge f has an associated weight W(f) (defaults to 1).
  - A transition t is activated if each place p connected through an edge f to t contains at least W(f) token.
  - When transition t fires, then W(f) token are transferred.

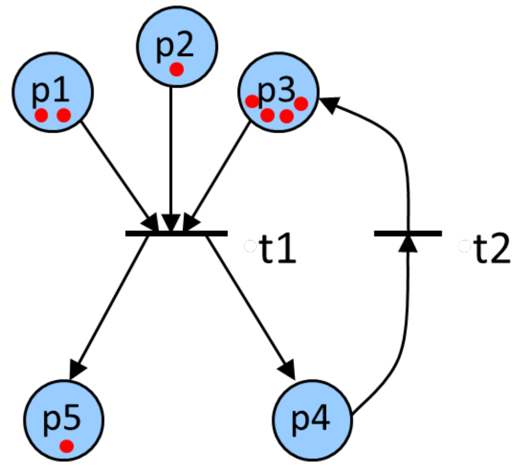


State Transition Function

- Using the previous definitions, we can now define the state transitions function δ of a Petri net:
  - Suppose that in a given Petri net (S, T, F, W, M_0) the transition t is activated. Before firing the marking is M.
  - Then after firing t, the new marking is M' = δ(M, t) with

$$M'(p) = \begin{cases} M(p) - W(p, t) & \text{if } (p, t) \in F \text{ and } (t, p) \notin F \\ M(p) + W(t, p) & \text{if } (t, p) \in F \text{ and } (p, t) \notin F \\ M(p) - W(p, t) + W(t, p) & \text{if } (t, p) \in F \text{ and } (p, t) \in F \\ M(p) & \text{otherwise} \end{cases}$$

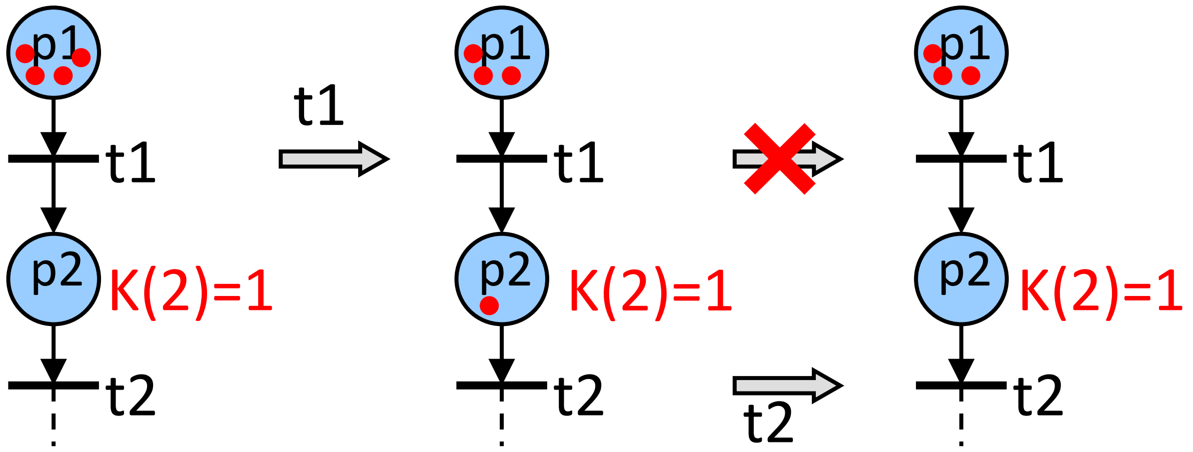
  - We also write sometimes M' = M · t instead of M' = δ(M, t).



$\{p1, p2, p3, p4, p5\} \in S$
$\{t1, t2\} \in T$
$\{(p1, t1), (p2, t1), (t1, p5), \ldots\} \in F$

Finite Capacity Petri Net

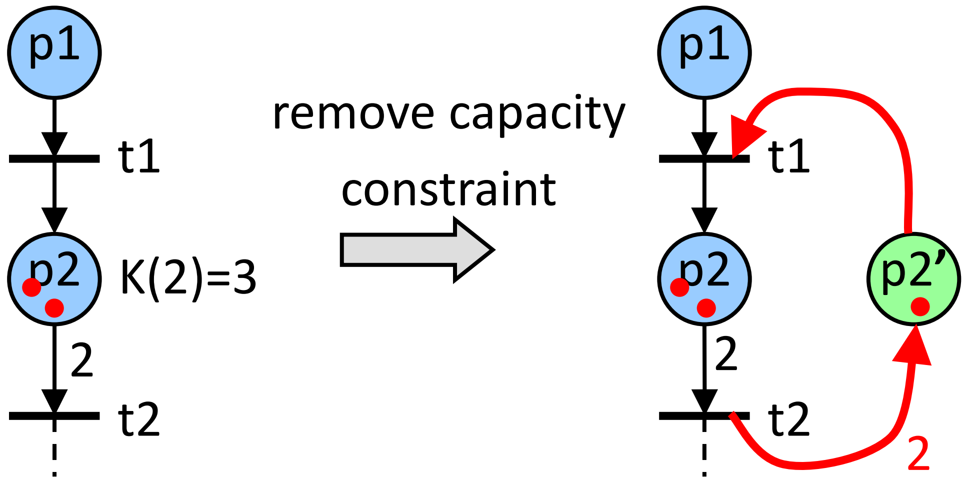
- Each place p can hold **maximally K(p)** token.
- A transition t is only active if all output places p_i of t cannot exceed $K(p_i)$ after firing t.



- Finite capacity Petri Nets can be transformed into equivalent infinite capacity Petri Nets (without capacity restrictions).
- Equivalence: Both nets have the same set of all possible firing sequences

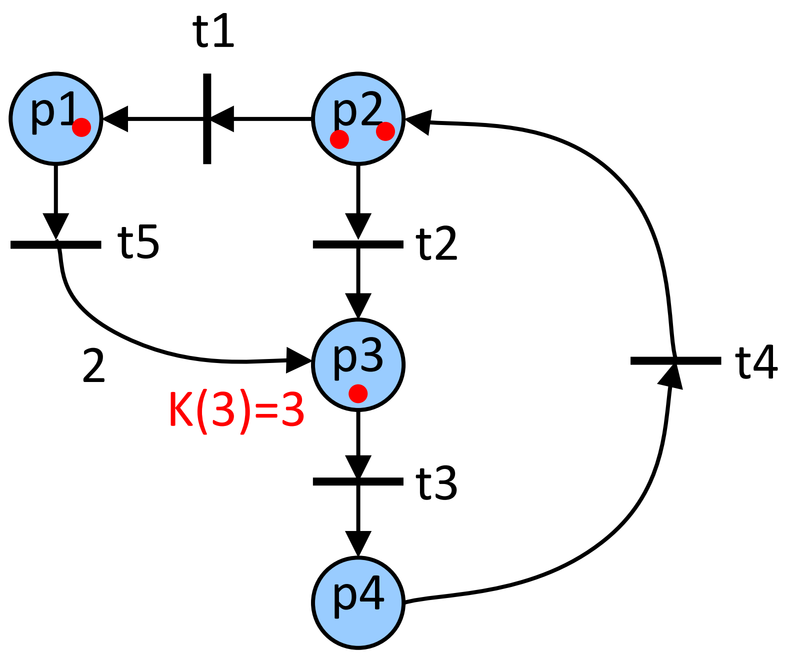
Removing Capacity Constraints

- For each place p with K(p) > 1, add a complementary place p’ with initial marking M_0(p’) = K(p) – M_0(p).
- For each outgoing edge f = (p, t), add an edge f’ from t to p’ with weight W(f).
- For each incoming edge f = (t, p), add an edge f’ from p’ to t with weight W(f).



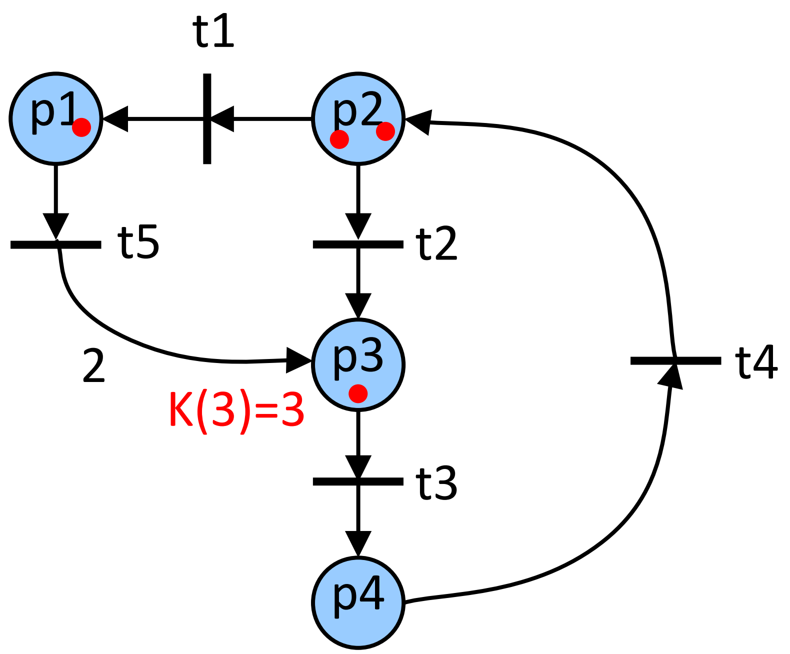
Your turn!

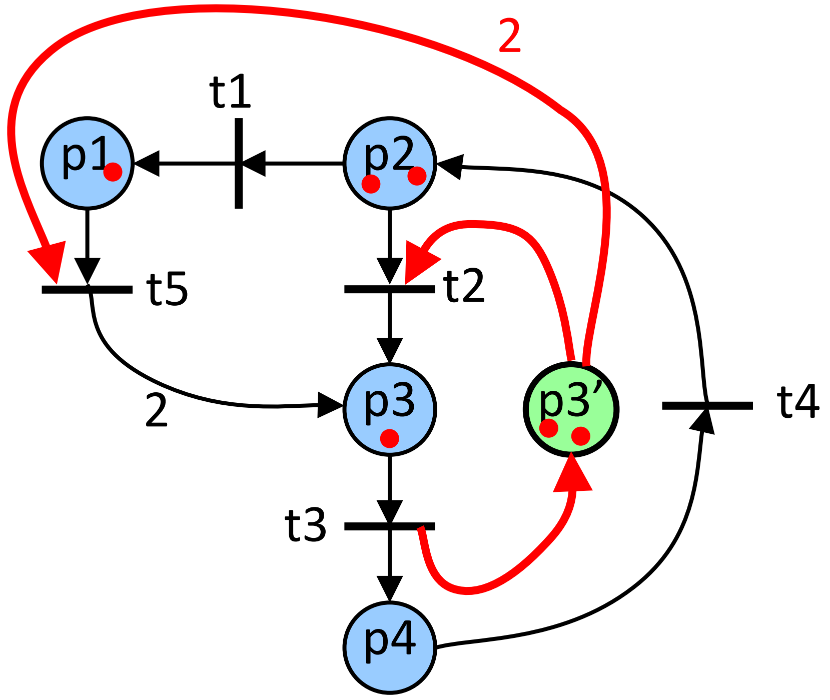
- Remove the capacity constraint from place p3:



Your turn!

- Remove the capacity constraint from place p3:

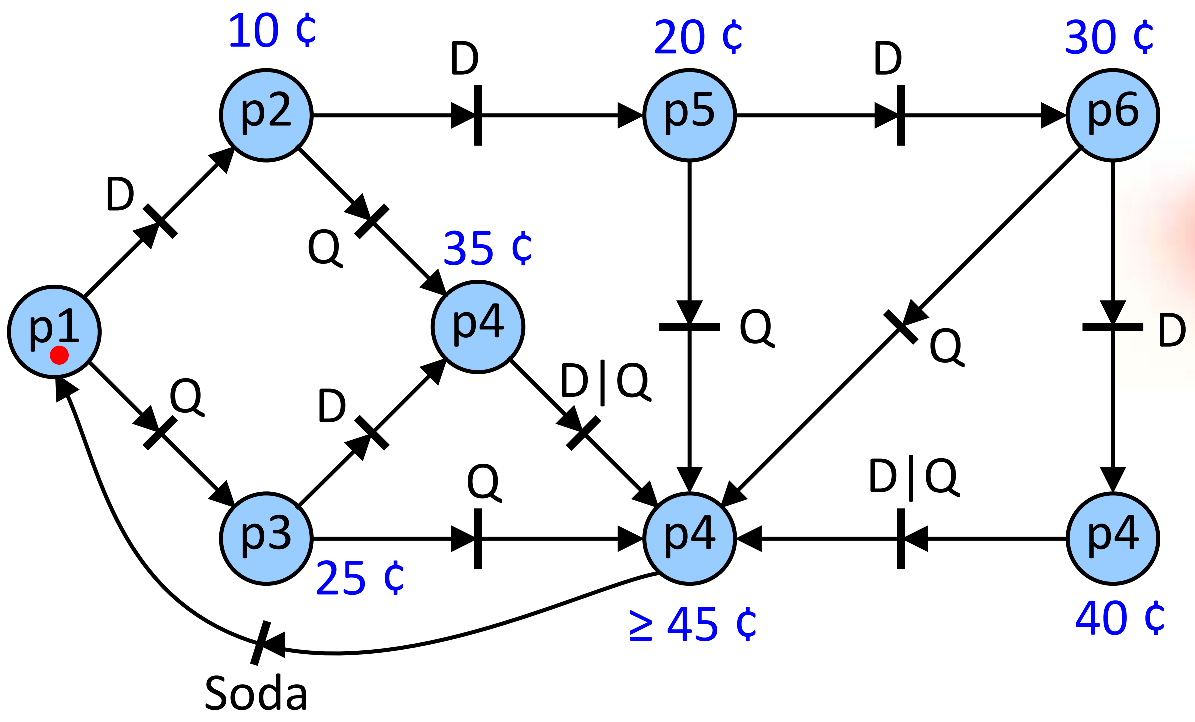




Modeling Finite Automata

- Finite automata can be represented by a subclass of Petri nets, where each transition has exactly one incoming edge and one outgoing edge.
- Such Petri nets are also called state machines.
- Coke vending machine revisited:

  - Coke costs 45 ¢.
  - Customer pays with Dime (10 ¢) or Quarter (25 ¢).
  - Overpaid money is lost.

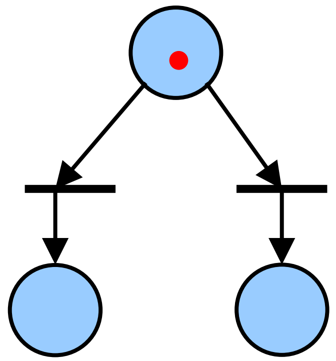


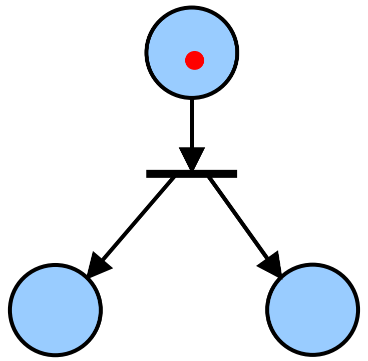
Concurrent Activities

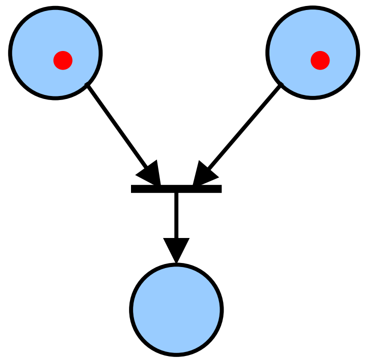
- Finite Automata allow the representation of decisions, but no concurrency.
- General Petri nets support concurrency with intuitive notation:

decision

concurrency

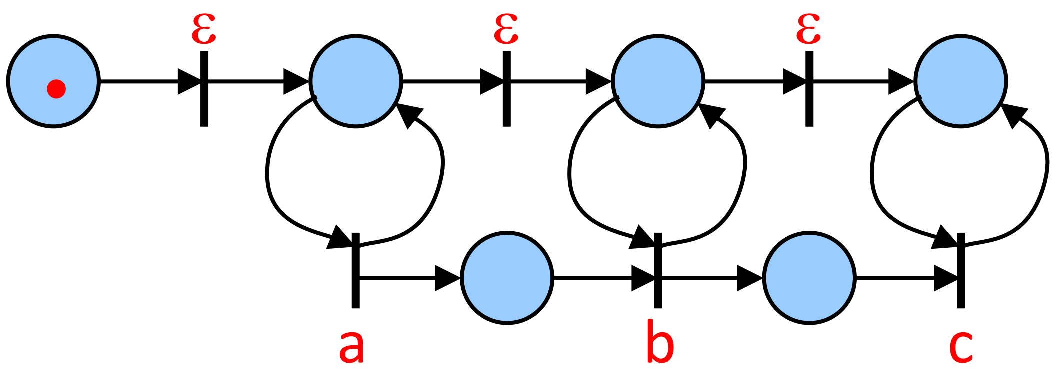

decision / conflict


fork


join / synchronization

Petri Net Languages

- Transitions are labeled with (not necessarily distinct) symbols.
- Final state is reached if no transition is activated.
- Any sequence of firing generates a string of symbols, i.e. a word of the language.



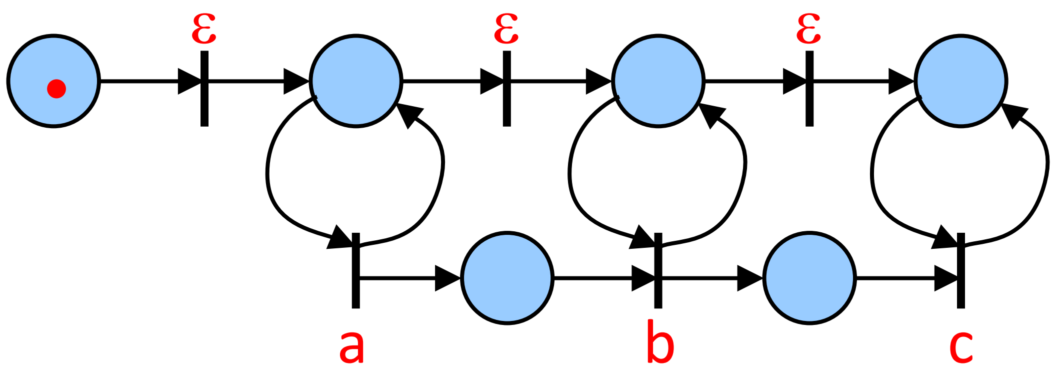
$L(M_0) = ??$

- Every finite-state machine can be modeled by a Petri net.

Every regular language is a Petri net language.
Not every Petri net language is regular.

Petri Net Languages

- Transitions are labeled with (not necessarily distinct) symbols.
- Final state is reached if no transition is activated.
- Any sequence of firing generates a string of symbols, i.e. a word of the language.



$L(M_0) = \{a^n b^m c^m \mid n \geq m \geq 0\}$

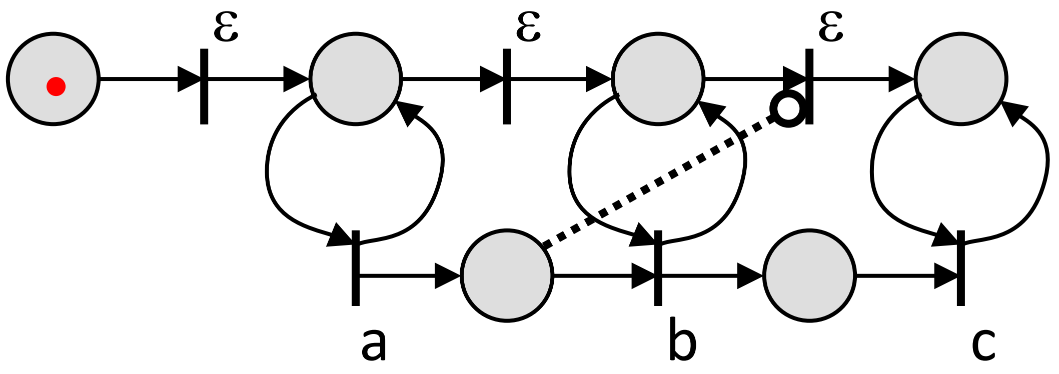
- Every finite-state machine can be modeled by a Petri net.

Every regular language is a Petri net language.
Not every Petri net language is regular.

Common Extensions

- **Colored Petri nets**: Tokens carry values (colors).

  Any Petri net with finite number of colors can be transformed into a regular Petri net.
- **Continuous Petri nets**: The number of tokens can be a real number (not only an integer).
  - Cannot be transformed into a regular Petri net.
- **Inhibitor Arcs**: Enable a transition if a place contains **no** tokens.
  - Cannot be transformed to a regular Petri net



$L(M_0) = \{a^n\, b^n\, c^n \mid n \geq 0\}$

- **Timed Petri nets**: See later …

Contents

Behavioral Properties (1)

Reachability

A marking M_n is *reachable* from M_0 iff there exists a sequence of firings $\{t_1, t_2, \dots t_n\}$ such that $M_n = M_0 \cdot t_1 \cdot t_2 \cdot \dots \cdot t_n$

K-Boundedness

A Petri net is *K-bounded if* the number of tokens in every place never exceeds K. The number of states is **finite** in this case.

Safety

1-Boundedness: Every node holds at most 1 token at any time.

Behavioral Properties (2)

Liveness

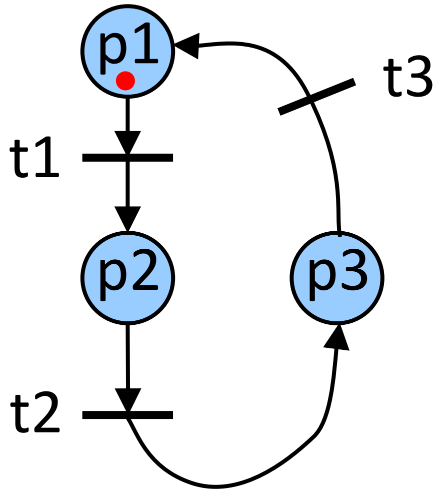
A transition t in a Petri net is

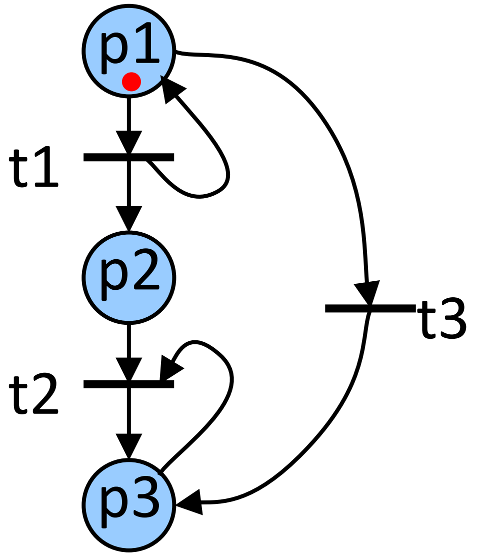
dead	iff t cannot be fired in any firing sequence,
L_1-live	iff t can be fired at least once in some firing sequence,
L_2-live	iff, $\forall\ k \in \mathbf{N}^+$, t can be fired at least *k* times in some firing sequence,
L_3-live	iff t appears infinitely often in some infinite firing sequence,
L_4-live	(live) iff t is L_1-live for every marking that is reachable from M_0 .

L_{j+1}-liveness implies L_j-liveness.

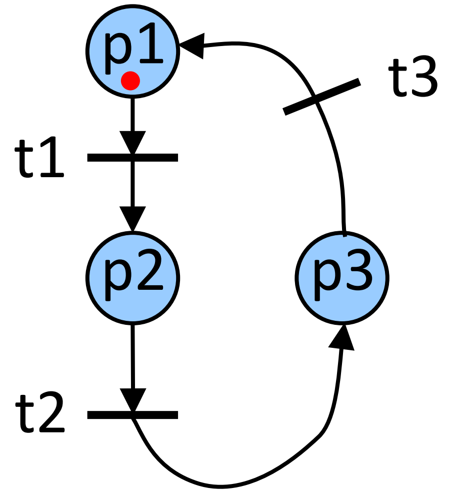
A Petri net is free of **deadlocks** iff there is no reachable marking from M_0 in which all transitions are dead.

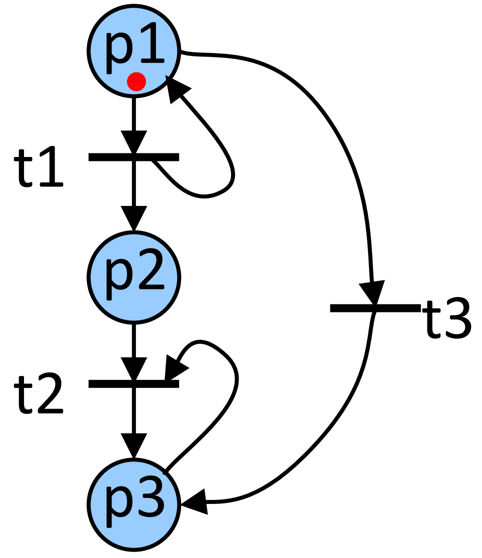
Liveness Examples





Liveness Examples





Every transition is L_4-live.

The Petri net is free of deadlocks.

t1 is L_3-live.

t2 is L_2-live.

t3 is L_1-live.

The Petri net is not free of deadlocks.

Contents

Analysis Methods

Coverability tree

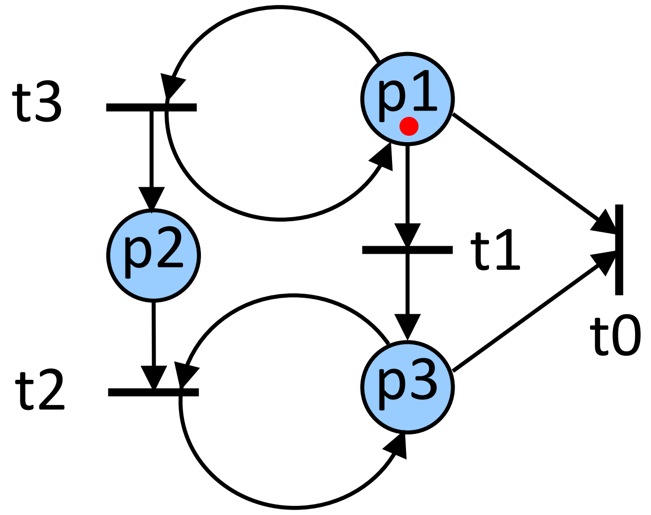
Enumeration of all reachable markings, limited to small nets if done by explicit enumeration. Reachability analysis similar to that of finite automata can be done if the net is bounded.

Incidence Matrix

Describes the token-flow and state evolution by a set of linear equations. This method allows to derive necessary but not sufficient conditions for reachability.

Coverability Tree

- **Question:** What token distributions are reachable?
- **Problem:** There might be infinitely many reachable markings, but we must avoid an infinite tree.
- **Solution:** Introduce a special symbol ω to denote an arbitrary number of tokens:



$$\begin{tikzcd}
& M_0 = [1\ 0\ 0] \arrow[dl, "t1"'] \arrow[dr, "t3"] & & \\
M_1 = [0\ 0\ 1] & & M_3 = [1\ \omega\ 0] \arrow[dl, "t1"'] \arrow[dr, "t3"] & \\
\textit{deadlock} & M_4 = [0\ \omega\ 1] \arrow[d, "t2"] & & M_6 = [1\ \omega\ 0] \\
& M_5 = [0\ \omega\ 1] & & \textit{old} \\
& \textit{old} & &
\end{tikzcd}$$

Coverability Tree – Algorithm

Special symbol ω, similar to ∞: $\forall n \in \mathbf{N}$: $\omega > n$; $\omega = \omega \pm n$; $\omega \geq \omega$

- Label initial marking M_0 as root and tag it as *new*
- **while** *new* markings exist, pick one, say M
    - Remove marking from M;
    - If M is identical to an already existing marking, mark it as *old*; **continue**;
    - If no transitions are enabled at M, tag it as *deadlock*; **continue**;
    - For each enabled transition t at M do
        - Obtain marking $M' = M \cdot t$
        - If there exists a marking M'' on the way from the root to M s.t. $M'(p) \geq M''(p)$ for each place p and $M' \neq M''$, replace $M'(p)$ with ω for *p* where $M'(p) > M''(p)$.
        - Introduce M' as a node, draw an arc with label t from M to M' and tag M' *new*.

Results from the Coverability Tree T

- The net is **bounded** iff ω does not appear in any node label of T
- The net is **safe** iff only ‘0’ and ‘1’ appear in the node labels of T
- A transition t is **dead** iff it does not appear as an arc in T
- If M is **reachable** from M_0, then there exists a node M' s.t. $M \leq M'$. This is a necessary, but not sufficient condition for reachability.

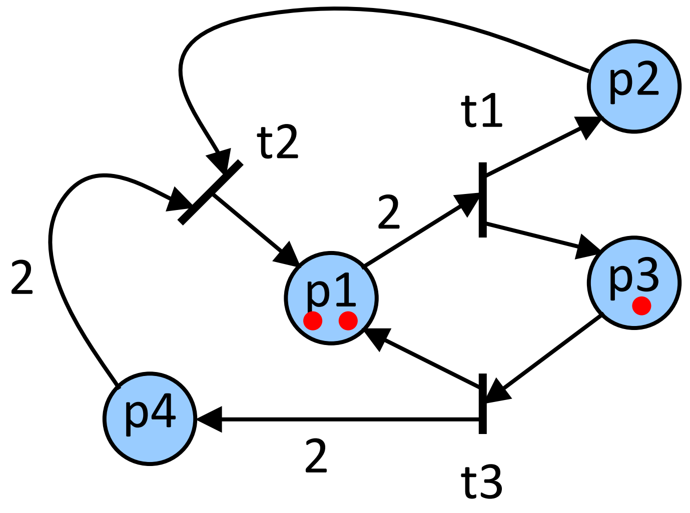
- For *bounded* Petri nets, the coverability tree T does not contain ω and is also called **reachability tree**, as all reachable markings are contained in it.

3/3

Contents

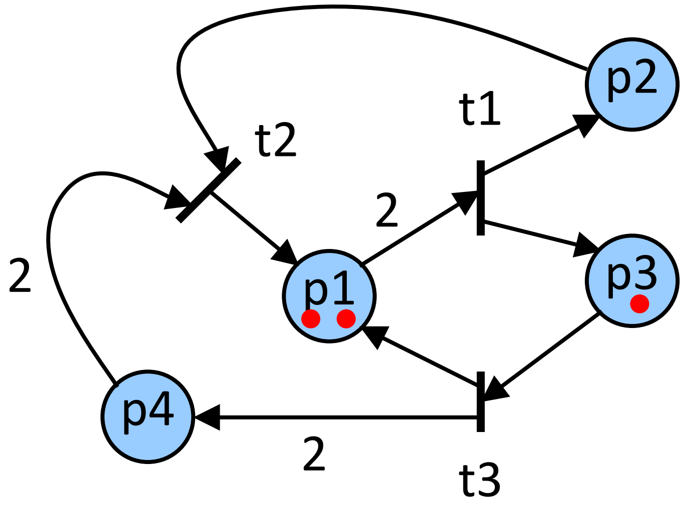
Incidence Matrix

- **Method:** Describe a Petri net through a set of linear equations.
- The ($m \times n$) **incidence matrix A** describes the token-flow for a Petri net with n transitions and m places.
  - The matrix element A_{ij} corresponds to the "gain" of tokens at place p_i when transition $t_j \leq$ fires. In other words, $A_{ij} = W(t_j , p_i) - W(p_i , t_j)$. Here, we set W(p,t) = 0 or W(t, p)=0 when the corresponding edges do not exist.
- A marking M is written as a $m \times 1$ column vector:



Incidence Matrix

- **Method:** Describe a Petri net through a set of linear equations.
- The ($m \times n$) **incidence matrix A** describes the token-flow for a Petri net with n transitions and m places.
  - The matrix element A_{ij} corresponds to the "gain" of tokens at place p_i when transition $t_j \leq$ fires. In other words, $A_{ij} = W(t_j , p_i) - W(p_i , t_j)$. Here, we set W(p,t) = 0 or W(t, p)=0 when the corresponding edges do not exist.
- A marking M is written as a $m \times 1$ column vector:

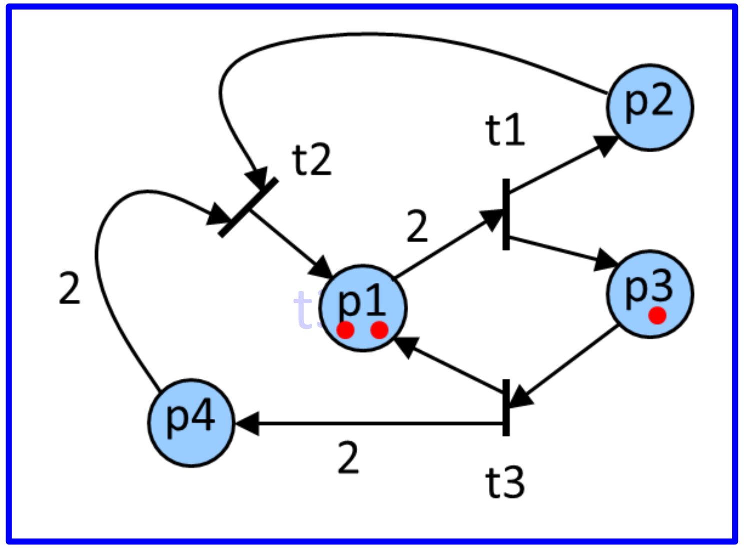


$$M_0 = \begin{bmatrix} 2 \\ 0 \\ 1 \\ 0 \end{bmatrix} \qquad A = \begin{bmatrix} -2 & 1 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \\ 0 & -2 & 2 \end{bmatrix}$$

State Equation

- The firing vector u describes the firing of a transition t. If transition t_i fires, then u_i consists of all '0', except for the *i*-th row, where it has a '1':

$$u_1 = \begin{bmatrix}1\\0\\0\end{bmatrix} \quad u_2 = \begin{bmatrix}0\\1\\0\end{bmatrix} \quad u_3 = \begin{bmatrix}0\\0\\1\end{bmatrix}$$



- A state transition from M to M' due to firing t_i is written as

$$\boxed{M' = \delta(M, t_i) = M + A \cdot u_i}$$

- For example, M_1 is obtained from M_0 by firing t3:

$$\begin{bmatrix}3\\0\\0\\2\end{bmatrix} = \begin{bmatrix}2\\0\\1\\0\end{bmatrix} + \begin{bmatrix}-2 & 1 & 1\\1 & -1 & 0\\1 & 0 & -1\\0 & -2 & 2\end{bmatrix}\begin{bmatrix}0\\0\\1\end{bmatrix}$$

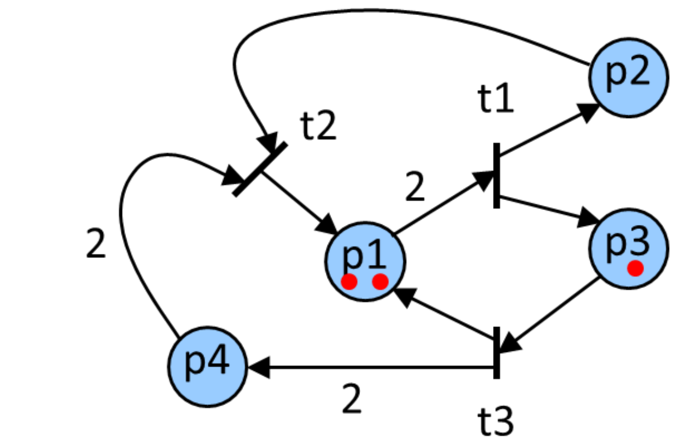
State Equation: Reachability

- A marking M_k is **reachable** from M_0 if there is a sequence σ of k transitions $\{t_{\sigma[1]}, t_{\sigma[2]}, \ldots, t_{\sigma[k]}\}$ such that $M_k = M_0 \cdot t_{\sigma[1]} \cdot t_{\sigma[2]} \cdot \ldots \cdot t_{\sigma[k]}$.
- Expressed with the incidence matrix:

$$M_k = M_0 + A \sum_{i=1}^{k} u_{\sigma[i]} \qquad (1)$$

which can be rewritten as

$$M_k - M_0 = \Delta M = Ax \qquad (2)$$



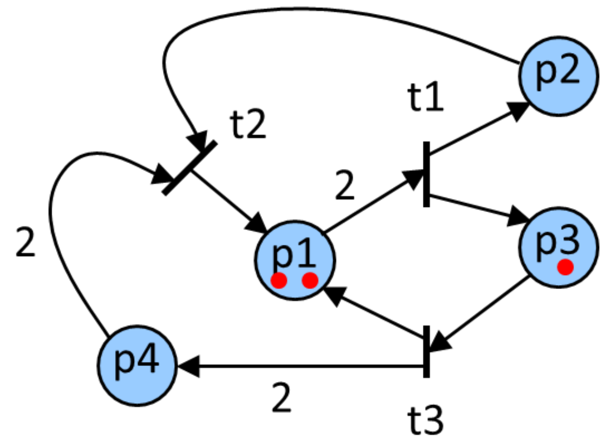
$$M_0 = \begin{bmatrix} 2 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad A = \begin{bmatrix} -2 & 1 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \\ 0 & -2 & 2 \end{bmatrix}$$

If M_k is reachable from M_0, equation (2) must have a solution where all components of *x* are non-negative integers. This is a necessary but not sufficient condition for reachability.

Reachability - Example

- Is $M_k = \begin{bmatrix} 3 \\ 0 \\ 0 \\ 2 \end{bmatrix}$ reachable?

- Is $M_k = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 2 \end{bmatrix}$ reachable?



$$M_0 = \begin{bmatrix} 2 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad A = \begin{bmatrix} -2 & 1 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \\ 0 & -2 & 2 \end{bmatrix}$$

Reachability - Example

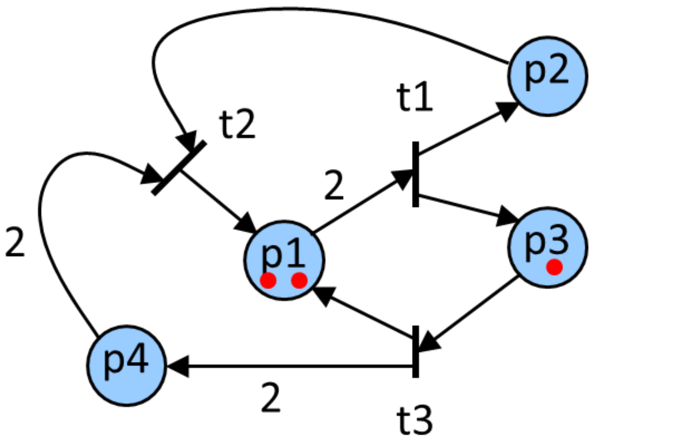
- Is $M_k = \begin{bmatrix} 3 \\ 0 \\ 0 \\ 2 \end{bmatrix}$ reachable? Possibly yes as $\Delta M = \begin{bmatrix} 1 \\ 0 \\ -1 \\ 2 \end{bmatrix}$

  and $M_k - M_0 = \Delta M = Ax$ with $x = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}$.

  It is reachable as the sequence {t_1, t_3, t_3, t_2} reaches M_k .

- Is $M_k = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 2 \end{bmatrix}$ reachable? No, as there is no solution to

  $M_k - M_0 = \Delta M = Ax$ with $\Delta M = \begin{bmatrix} -1 \\ 0 \\ -1 \\ 2 \end{bmatrix}$.



$$M_0 = \begin{bmatrix} 2 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad A = \begin{bmatrix} -2 & 1 & 1 \\ 1 & -1 & 0 \\ 1 & 0 & -1 \\ 0 & -2 & 2 \end{bmatrix}$$

Contents

Discrete Event Models with Time

- In most of the discrete event systems, time is an important factor, for example queuing systems, computer systems, digital circuits, workflow management, business processes.
- Based on a **timed discrete event model** we would like to determine properties like delay, throughput, execution rate, resource load and buffer sizes.
- There are many ways of adding the concept of time to finite automata and Petri nets. In the following, one specific model is used.

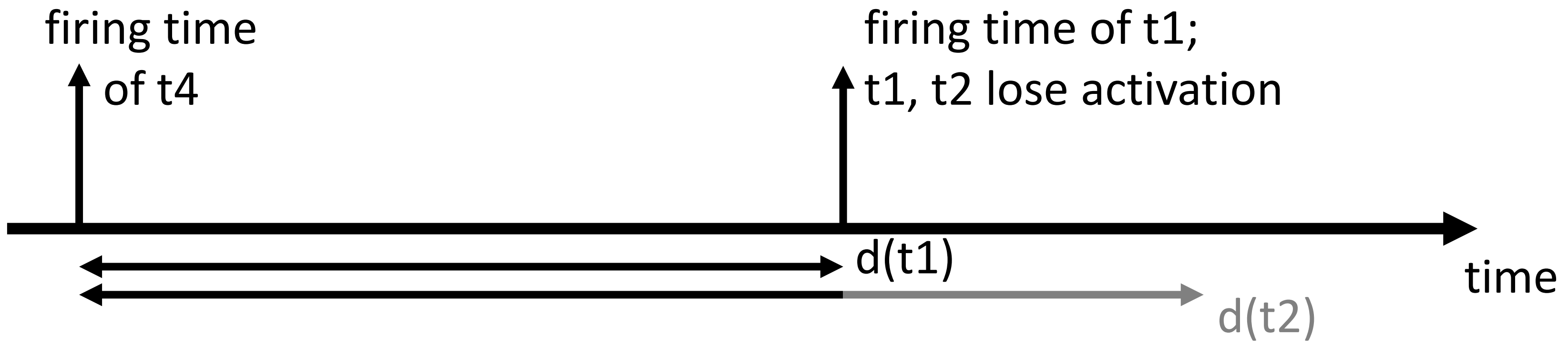
- What can you do with it?
  - **Verifying** timed properties (How long does it take at most until a certain event happens? What is the minimum time between two events?).
  - **Simulate** a timed discrete event model (Given a specific input, how does the system state evolve over time? Is the resulting trace of execution what we had in mind?).

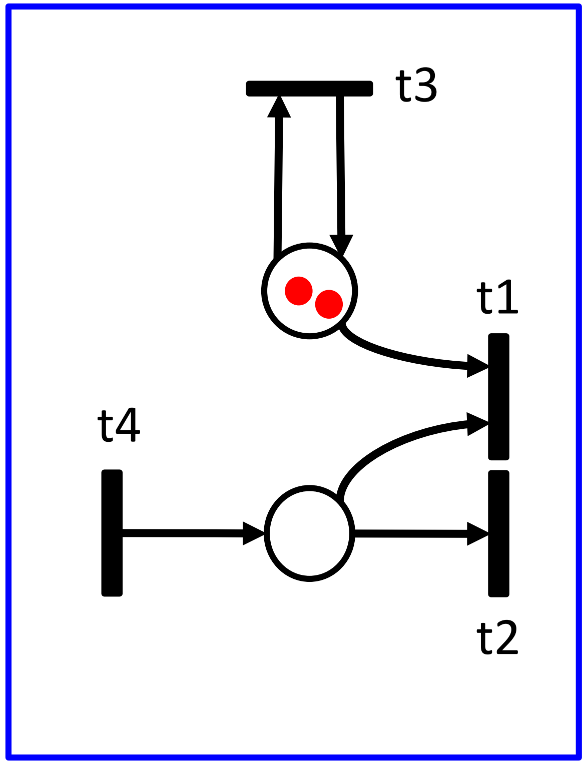
Timed Petri Net

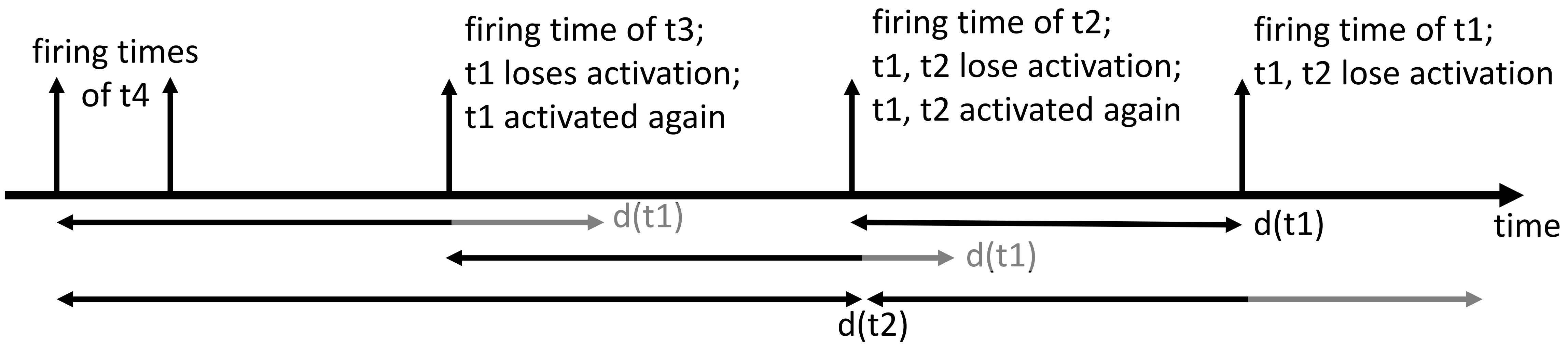
- We define a delay function d: T → **R** that determines for each transition t a **delay** between its activation and firing.
  - If called, the function d(t) returns a delay for the current activation. Repeated calls may lead to the same value (constant delay) or to different delays, e.g. by returning the value of a random variable.
  - The function is called for every new activation of transition t and determines the time until the transition fires. There is a new activation whenever a token is removed from some input place of t.
  - If the transition t loses its activation, then at the next activation d(t) is called again.
  - Only one transition fires at a time; in case of two activations with the same firing time, one of them is chosen non-deterministically to fire first.

Timed Petri Net

Example traces:

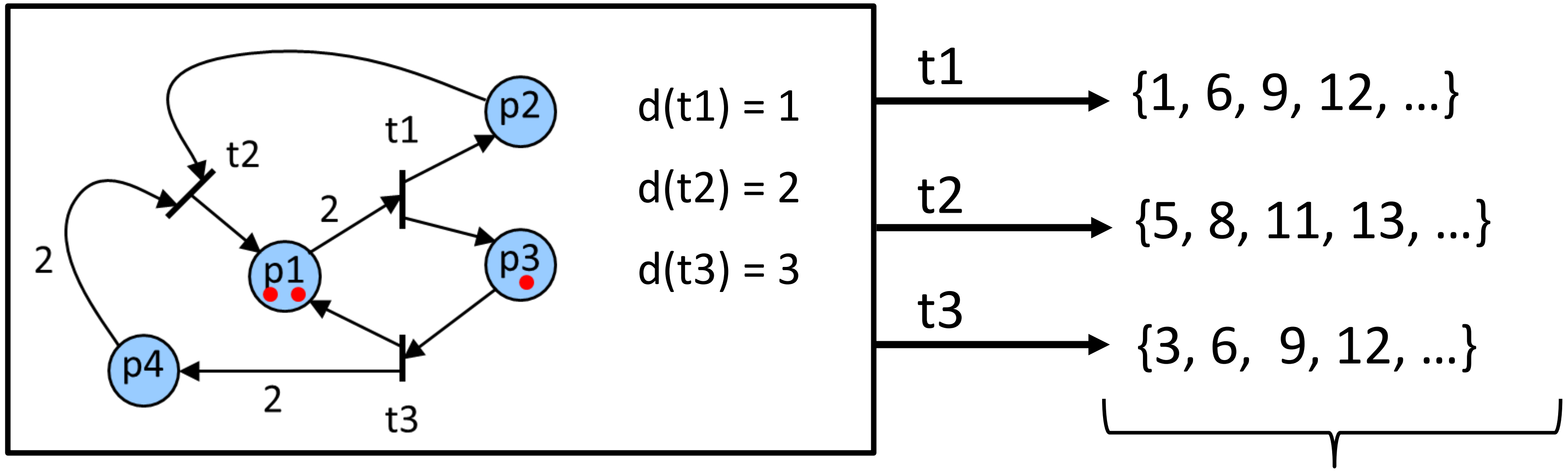






Time Petri Net

- The time when a transition t fires is called its **firing time**.
- A timed Petri net can be regarded as a **generator for firing times** of its transitions.



initialization time 0

firing time sequences for transitions t1, t2 and t3

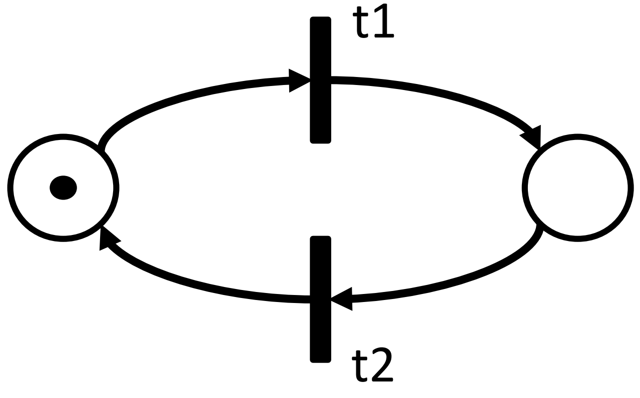
- How do we get the firing times? By simulation!

Time Petri Net

Example Continuous Time Markov Chain:

$$\begin{tikzcd}
1 \arrow[r, bend left, "\lambda"] & 0 \arrow[l, bend left, "\mu"]
\end{tikzcd}$$

Figure 6.1: A CTMC modeling an unreliable system. In state 1 the system is working, in state 0 the system is faulty. The *failure rate*, i.e., the time until the system fails, is exponentially distributed with parameter λ. After a failure, the repair takes some time, exponentially distributed with parameter μ.



d(t1) returns a sample of an exponentially distributed random variable with parameter λ

d(t2) returns a sample of an exponentially distributed random variable with parameter μ

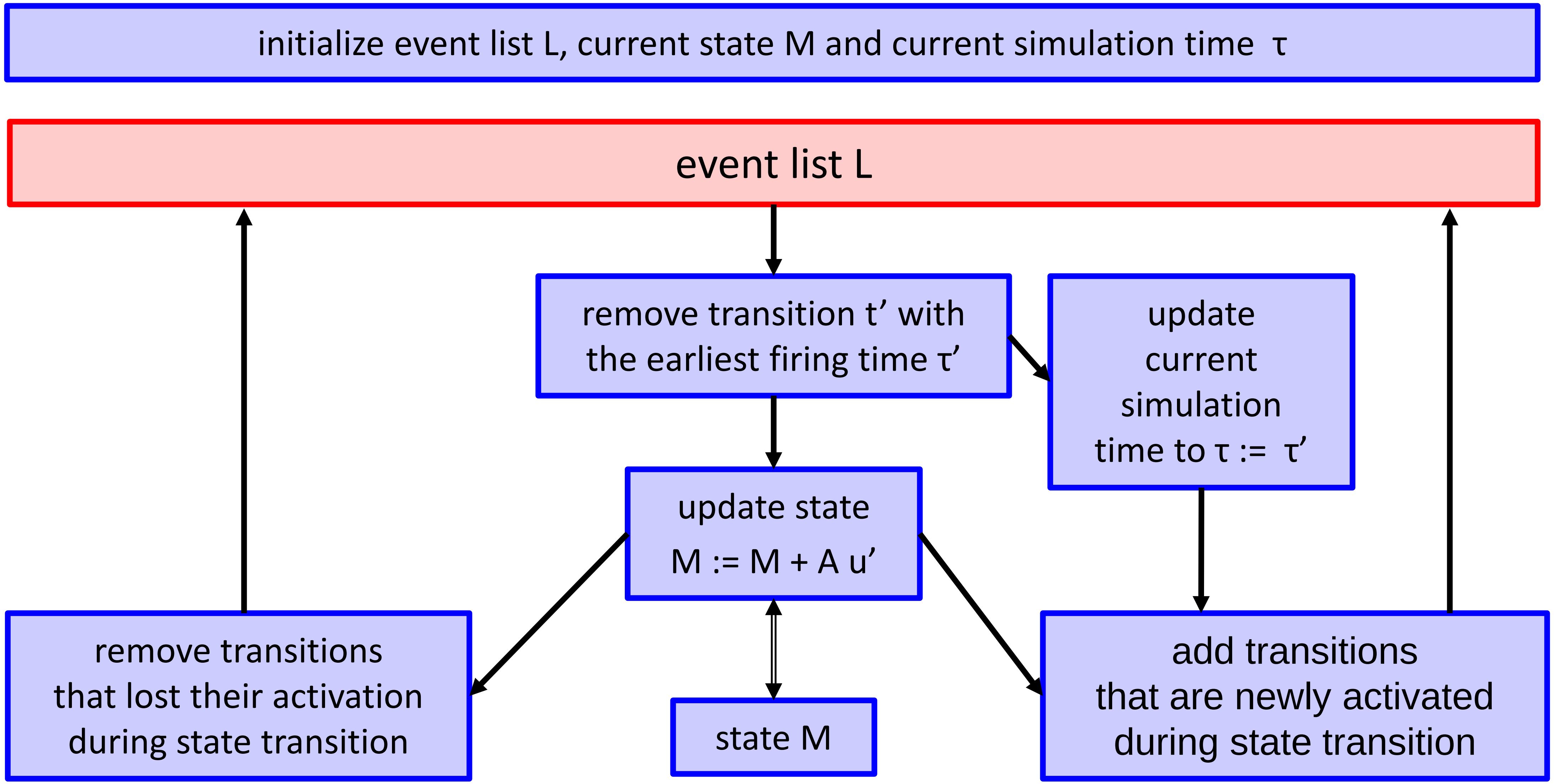
Contents

Timed Petri Net – Simulation Principle

- The simulation is based on the following basic principles.
    1. The simulator maintains a set L of currently activated transitions and their firing times. (We call L **the event list** from now on.)
    2. A transition with the earliest firing time is selected and fired. The **state** of the Petri net as well as the current **simulation time** is **update**d accordingly.
    3. All transitions that lost their activation during the state transition are **remove**d from the event list L.
    4. Afterwards, all transitions that are newly activated are **add**ed in the event list L together with their firing times.
    5. Then we continue with 2. unless the event list L is empty.

- This simulation principle holds in one form or the other for any simulator of timed discrete event models.

Timed Petri Net – Simulation Principle


initialize event list L, current state M and current simulation time τ
event list L
remove transition t' with the earliest firing time τ'
update current simulation time to τ := τ'
update state M := M + A u'
state M
remove transitions that lost their activation during state transition
add transitions that are newly activated during state transition

Timed Petri Net – Simulation Steps

- **Initialization**:
  - set the initial simulation time τ := 0
  - set the current state to M := M_0
  - for each activated transition t, add the event (t, τ + d(t)) to the event list L

- Determine and remove **current event**:
  - determine a firing event (t', τ') with the earliest firing time:

$$\forall 1 \leq i \leq N \ : \ \tau' \leq \tau_i \quad \text{where} \quad L = \{(t_1, \tau_1), (t_2, \tau_2), \cdots, (t_N, \tau_N)\}$$

  - remove event (t', τ') from the event list L:

$$L := L \setminus \{(t', \tau')\}$$

Timed Petri Net – Simulation Steps

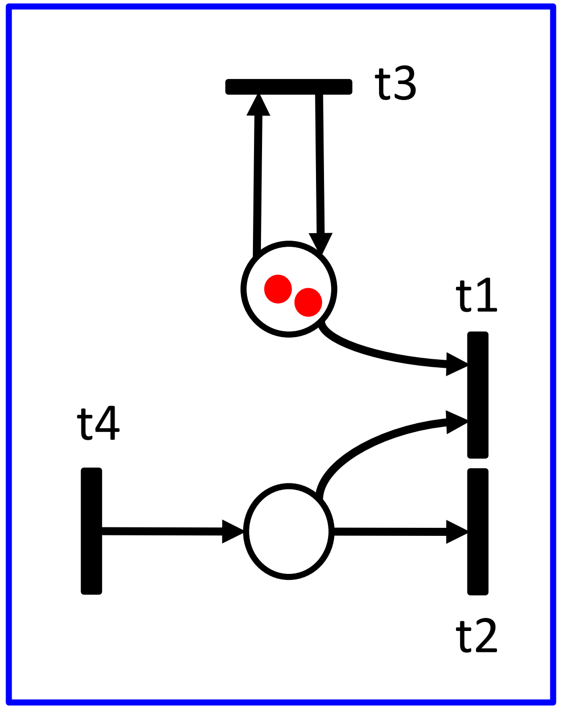
- **Update** current **simulation time**:
    - set current simulation time $\tau := \tau'$

- **Update token distribution** M
    - suppose that the firing transition has index j, i.e. $t_j = t'$. Then, the firing vector is

$$u' = [\; 0 \quad \cdots \quad 0 \quad \underset{j}{1} \quad 0 \quad \cdots \quad 0 \;]^t$$

- update current state $M := M + A\,u'$

Timed Petri Net – Simulation Steps



- **Remove transitions** from L that lost activation:
  - determine the set of places S' from which at least one token was removed during the state transition caused by t':

$$S' = \{p \,|\, (p, t') \in F\}$$

  - remove from event list L all transitions in T' that lost their activation due to this token removal:

$$T' = \{t \,|\, (p, t) \in F \wedge p \in S'\}$$

- **Add** all **transitions** to event list L that are activated but not in L yet:
  - if some transition t with $M(p) \geq W(p, t)$ for all $(p, t) \in F$ is not in event list L, then add $(t, d(t))$ to the event list L:

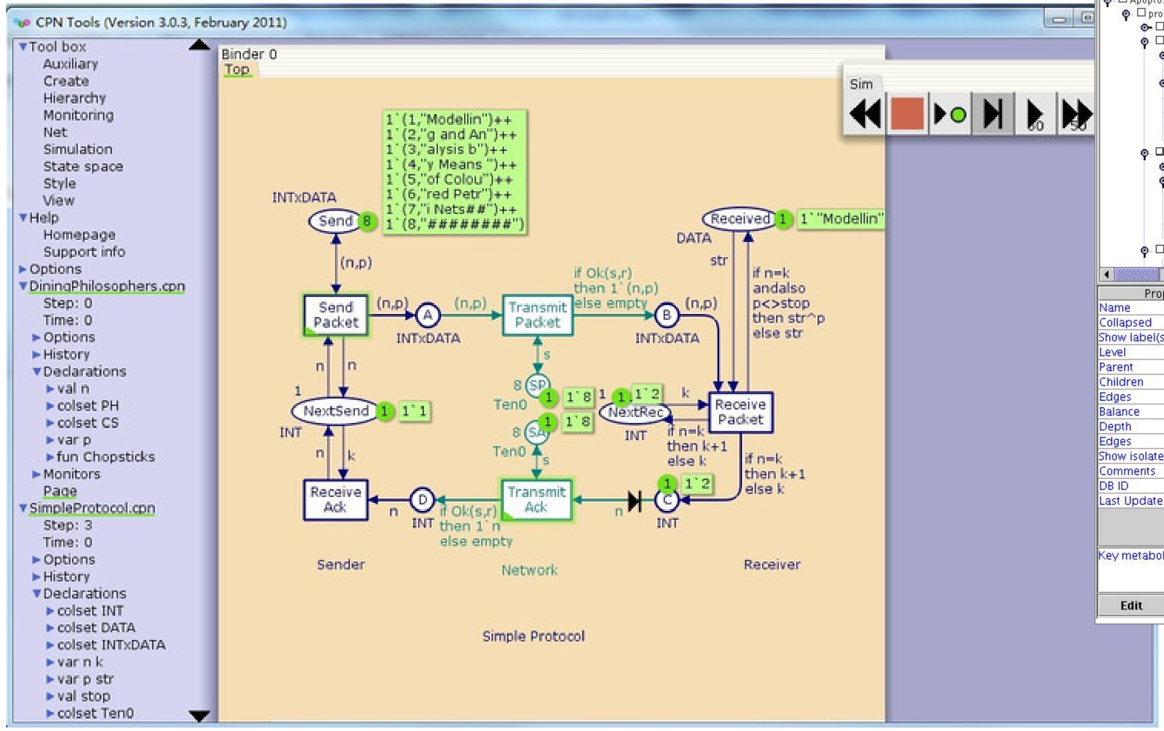
$$L := L \cup \{(t, d(t))\}$$

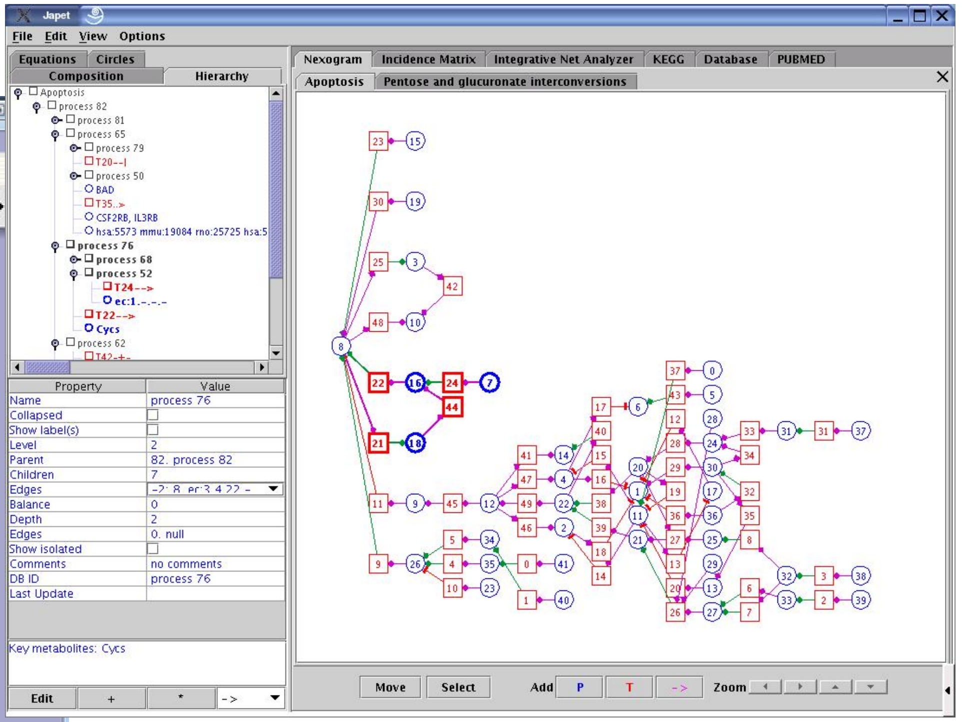
Petri Net Simulators

- There are many simulators available, see e.g.

  https://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/quick.html

  http://cpntools.org/


CPN Tools (Version 3.0.3, February 2011)
Tool box
Auxiliary
Create
Hierarchy
Monitoring
Net
Simulation
State space
Style
View
Help
Homepage
Support info
Options
DiningPhilosophers.cpn
Step: 0
Time: 0
Options
History
Declarations
val n
colset PH
colset CS
var p
fun Chopsticks
Monitors
Page
SimpleProtocol.cpn
Step: 3
Time: 0
Options
History
Declarations
colset INT
colset DATA
colset INTxDATA
var n k
var p str
val stop
colset Ten0
Binder 0
Top
Sim
1`(1,"Modellin")++
1`(2,"g and An")++
1`(3,"alysis b")++
1`(4,"y Means ")++
1`(5,"of Colou")++
1`(6,"red Petr")++
1`(7,"i Nets##")++
1`(8,"########")
INTxDATA
Send
Send Packet
(n,p)
A
INTxDATA
Transmit Packet
if Ok(s,r) then 1`(n,p) else empty
B
Received
1`"Modellin"
DATA
str
if n=k andalso p<>stop then str^p else str
NextSend
1`1
INT
Ten0
SP
SA
1`8
1`2
NextRec
Receive Packet
if n=k then k+1 else k
Receive Ack
D
INT
if Ok(s,r) then 1`n else empty
Transmit Ack
C
Sender
Network
Receiver
Simple Protocol


Japet
File
Edit
View
Options
Equations
Circles
Composition
Hierarchy
Nexogram
Incidence Matrix
Integrative Net Analyzer
KEGG
Database
PUBMED
Apoptosis
Pentose and glucuronate interconversions
Apoptosis
process 82
process 81
process 65
process 79
T20--|
process 50
BAD
T35..>
CSF2RB, IL3RB
process 76
process 68
process 52
T24-->
T22-->
Cycs
process 62
Property
Value
Name
process 76
Collapsed
Show label(s)
Level
2
Parent
82. process 82
Children
7
Edges
Balance
0
Depth
2
Edges
0. null
Show isolated
Comments
no comments
DB ID
process 76
Last Update
Key metabolites: Cycs
Edit
Move
Select
Add
P
T
Zoom