Discrete and continuous dynamic systems

Petri Nets
Definition and operation

Katalin Hangos, Anna Ibolya Pózna

University of Pannonia
Faculty of Information Technology
Department of Electrical Engineering and Information Systems

hangos.katalin@virt.uni-pannon.hu, pozna.anna@virt.uni-pannon.hu

Apr 2018

Lecture overview

1. Previous notions
   - Discrete event systems
   - Automata models
   - Simple examples
2. Petri net models
   - Description forms
   - Operation (dynamics) of Petri nets
   - Parallel and conflicting execution steps
3. Solution of Petri net models
   - The reachability graph

Discrete event systems

Characteristic properties:

- the *range space* of the signals (input, output, state) is **discrete**: $x(t) \in \mathbf{X} = \{x_0, x_1, ..., x_n\}$
- *event*: the occurrence of change in a discrete value
- *time is also* ***discrete***: $T = \{t_0, t_1, ..., t_n\} = \{0, 1, ..., n\}$

Only the **order of the events** is considered

- description of sequential and parallel events
- **application area**: scheduling, operational procedures, resource management

Discrete event systems (DES) are special kinds of dynamical systems. The three main characteristic properties are the

- **discrete range space** of the signals, i.e. the input, output and state variables can get their values from a set of discrete values. *Example*: a two state switch has the range space {'off','on'}.
- **events** occurs when a variable changes its discrete value from one to an other. *Example*: the switch goes from 'on' to 'off' and remains in that state.
- **discrete time** means that the time is also measured at discrete points. *Important*: The subsequent time instances are not necessarily equidistant. The time is usually measured when an event occurs in the system, and not between them. For example, the system starts working at t=0 and an event occurs at t=5. Then only t=0 and t=5 are recorded, and t=1,2,3,4 are not.

An important feature of DES is that the only the **order of events** are considered. We are not interested in the exact occurrence time of the event, but its relative occurrence to an other event. For example, the actual event occurred before or after another event. With the help of DES we can describe parallel and sequential events.

Discrete time linear state space models

$$
\begin{aligned}
x(k+1) &= \Phi x(k) + \Gamma u(k) \qquad (\textit{state equation}) \\
y(k) &= Cx(k) + Du(k) \qquad (\textit{output equation})
\end{aligned}
$$

given initial condition $x(0)$;
vector valued signals

$$x(k) \in \mathcal{R}^n \ , \ y(k) \in \mathcal{R}^p \ , \ u(k) \in \mathcal{R}^r$$

system parameters:

$$\Phi \in \mathcal{R}^{n \times n} \ , \ \Gamma \in \mathcal{R}^{n \times r} \ , \ C \in \mathcal{R}^{p \times n} \ , \ D \in \mathcal{R}^{p \times r}$$

(Not necessarily) equidistant $(t_k - t_{k-1} = \Delta h)$

$$x(k) = x(t_k) \quad , \quad u(k) = u(t_k) \quad , \quad y(k) = y(t_k)$$

It is important to not confuse the discrete time systems and DES. In the previous lectures we used Discrete Time Linear State Space models which can be described by a set of linear difference equations. The usual matrix representation can be seen here.

Discrete event systems – discrete time state space models

Generalization of discrete time linear state space models

$$x(k+1) = \Psi(x(k), u(k)) \qquad (\textit{state equation})$$
$$y(k) = h(x(k), u(k)) \qquad (\textit{output equation})$$

with given initial condition $x(0)$ and nonlinear state Ψ and output function h.

Discrete event system:

1. discrete time with non-equidistant sampling
2. the range space of the signals is discrete
3. event: change in the discrete value of a signal

The discrete time (DT) state space models can be generalized to get a discrete event system. Creating a nonlinear state function Ψ which depends on the actual state x(k) and the actual input u(k), we can describe the state transition operations. The $\Psi(x(k), u(k))$ function defines the next state for each state-input pair. Similarly a nonlinear output function $h(x(k), u(k))$ can be defined which gives the current output for the current state-input pair. This can be a starting point of a DES.
The main difference that in DES the range spaces of the signals are restricted to discrete values. In a discrete time system the variables may have any value from a specified interval. For example:

- DT: $x \in [0, 10]$
- DES: $x \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$

The second difference is the handling of time. In a discrete time system we usually have a clock, and we update the system variables at every tick, even if there is no change in the input/output/state variables. In discrete event systems, the time is only recorded when an event (=change in the discrete values of the variables) occurs. We are not interested in the elapsed time between two events, only the order of them is important.

Automaton - abstract model: $\mathbf{A} = (Q, \Sigma, \delta; \Sigma_O, \varphi)$

- **Set of states**: Q
- **finite alphabet** of the input tape: $\Sigma = \{\#; a, b, ...\}$
- **State transition function**: $\delta : Q \times \Sigma \to Q$
- *Set of initial and final states*: $Q_I, \; Q_F \; \subseteq \; Q$
- **finite alphabet** of the output tape: $\Sigma_O = \{\#; \alpha, \beta, ...\}$
- **Output function**: $\varphi : Q \to \Sigma_O$

Graphical description: weighted directed graph

- **Vertices**: states (Q)
- **Edges**: state transitions (δ)
- **Edge weights**: input symbols (Σ)

Automata models are one of the possible representations of DES. An automaton is defined with the following notions:

- set of states Q: it contains all of the possible states of the automaton
- input alphabet Σ: contains the possible input values
- state transition function δ: defines the next state for each state-input pair
- set of initial and final states Q_I, Q_F
- output alphabet Σ_O: contains the possible output values
- output function φ assigns an output value to each state

Operation of automata

Given

- Initial state: $q_0 \in Q_I \subseteq Q$
- The content of the input tape: $S = [\sigma_1, \sigma_2, ..., \sigma_n]$, $\sigma_i \in \Sigma$

Compute

- Final state: if $q_f \in Q_F \subseteq Q$, then the automaton **accepts** the input
- The content of the output state: $S_O = [\zeta_1, \zeta_2, ..., \zeta_n]$, $\zeta_i \in \Sigma_O$

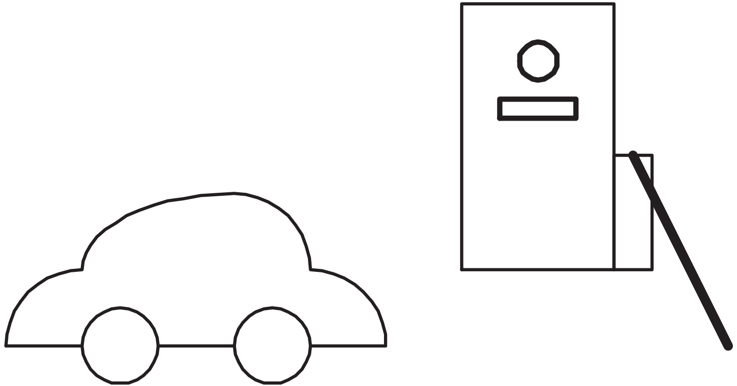
The automata operates as follows. Initially the automaton is in its initial state. Then it reads the first symbol from the input tape. The automaton steps into the next state according to the state transition function considering the actual state-input pair. The automaton writes an output symbol to the output tape according to the output function. After the whole input tape is processed the automaton is either in a final state or not. In the former case the automaton accepts the input which means it is a valid operation.

Automata - discrete event systems

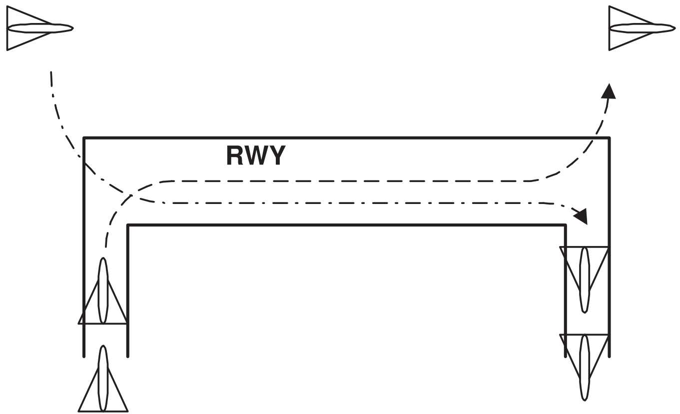
	Automaton model	Discrete event state space model
State space	Q	$\mathcal{X} \in \mathbb{Z}^n$
Input u	string from Σ	discrete time discrete valued signal
Output y	string from Σ_O	discrete time discrete valued signal
State equation	$q(k+1) = \delta(q(k), u(k))$	$x(k+1) = \Psi(x(k), u(k))$
Output equation	$y(k) = \varphi(x(k))$	$y(k) = h(x(k), u(k))$

Here you can see a side-by-side comparison of automata and discrete event state space models.

Introductory example: Garage gate



Simple example: Runway



Overview - Petri nets: modelling and dynamics

Petri net - abstract description: $\mathbf{PN} = (P, T, I, O)$

Static description (structure)

- set of **places (conditions)**: P
- set of **transitions (events)**: T
- **Input (pre-condition) function**: $I : T \rightarrow P^{\infty}$
- **Output (consequence) function**: $O : T \rightarrow P^{\infty}$

Graphical description: bipartite directed graph

- **Vertices**: places (P) and transitions (T) (partitions)
- **Edges**: input and output functions (I, O)

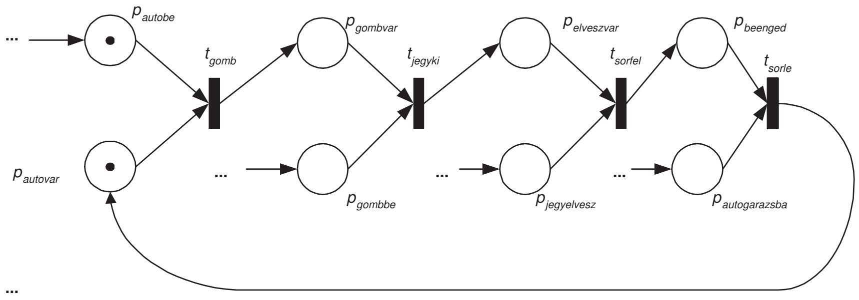
Petri nets are also popular tools to represent DES. The explanation of the formal definition is given here.

- set of places (conditions): P - places refer to preconditions or consequences of events. Represented by circles. *Example*: garage gate is *waiting for a car*.
- set of transitions (events): T transitions refer to events that may occur in the system Represented by black rectangles. *Example*: the driver *press the button*.
- Input function: assigns input places (preconditions) to the transitions (events)
- Output function: assigns output places (consequences) to the transitions (events)

Graphically Petri nets are represented by bipartite directed graphs. Places - circles, transitions - rectangles.
Edges: input and output functions. Edge direction: from a place to a transition if the place is a precondition of the event (the place is in the input function of the transition). From a transition to a place if the place is the consequence of the event.

Example: garage gate – 1

Petri net model - graphical description



Here is a simple example, which models the operation of the garage gate.
Remark: the places p_{autobe}, p_{gombbe}, $p_{jegyelevesz}$, $p_{autogarazsba}$ are the operations of the car driver, who can be modeled by a different Petri net.

Example: garage gate – 2

Petri net model - formal description

Places (states; inputs):

$$P = \{p_{autovar}, p_{gombvar}, p_{elveszvar}, p_{beenged} \; ; \; p_{autobe}, p_{gombbe}, p_{jegyelevesz}, p_{autogarazsba}\}$$

Transitions:

$$T = \{t_{gomb}, t_{jegyki}, t_{sorfel}, t_{sorle}\}$$

Input function:

$$I(t_{gomb}) = \{p_{autobe}, p_{autovar}\} \quad , \quad I(t_{jegyki}) = \{p_{gombbe}, p_{gombvar}\}$$
$$I(t_{sorfel}) = \{p_{jegyelvesz}, p_{elveszvar}\} \quad , \quad I(t_{sorle}) = \{p_{beenged}, p_{autogarazsba}\}$$

Output function:

$$O(t_{gomb}) = \{p_{gombvar}\} \quad , \quad O(t_{jegyki}) = \{p_{elveszvar}\}$$
$$O(t_{sorfel}) = \{p_{beenged}\} \quad , \quad O(t_{sorle}) = \{p_{autovar}\}$$

Dynamics of Petri nets

Marking function: marking points (**tokens**)

$$\mu : \mathbf{P} \rightarrow \mathcal{N} \quad , \quad \mu(p_i) = \mu_i \geq 0$$
$$\underline{\mu}^T = [\mu_1, \mu_2, \ldots, \mu_n] \quad , \quad n = |\mathbf{P}|$$

Transition **fires** (operates): when its pre-conditions are "true" (there is a **token** on its input places)

$\underline{\mu}^{(i)}[t_j > \underline{\mu}^{(i+1)}$

after firing the consequences become "true"

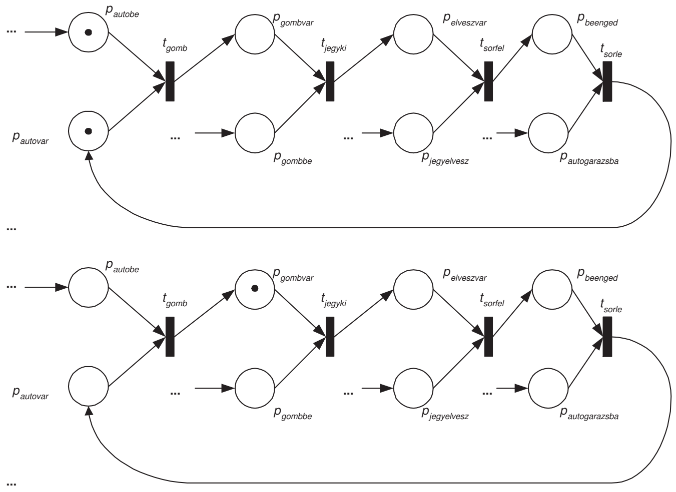
Firing (operation) sequence

$$\underline{\mu}^{(0)}[t_{j0} > \underline{\mu}^{(1)}[t_{j1} > ...[t_{jk} > \underline{\mu}^{(k+1)}$$

The firing sequence is denoted by $\underline{\mu}^{(0)}[t_{j0} > \underline{\mu}^{(1)}[t_{j1} > ...[t_{jk} > \underline{\mu}^{(k+1)}$. Here $\underline{\mu}^{(0)}$ denotes the initial state of the Petr net, and $\underline{\mu}^{(k)}$ is the state after the kth step. t_{jk} is the label of the transition. $\underline{\mu}^{(0)}[t_{j0} > \underline{\mu}^{(1)}$ means that transition t_{j0} has fired and changed the marking from $\underline{\mu}^{(0)}$ to $\underline{\mu}^{(1)}$.

Example: garage gate – 3

One operation steps



The operation of the garage gate is demonstrated here. It can be seen that only transition t_{gomb} is enabled because there is one token on its input places p_{autobe} and $p_{autovar}$. After the firing of t_{gomb} the tokens are removed from the input places and one token appeared on the output place of the transition $p_{gombvar}$. In this situation there is no enabled transition. t_{jegyki} is not enabled because there is no token on place p_{gombbe}

Example: garage gate – 4

Formal description of an operation step

Marking vector

$$\underline{\mu}^T = [\mu_{autovar}, \mu_{gombvar}, \mu_{elveszvar}, \mu_{beenged} \; ; \; \mu_{autobe}, \mu_{gombbe}, \mu_{jegyelevesz}, \mu_{autogarazsba}]$$

Operation (firing) of transition t_{gomb}

$$\underline{\mu}^{(1)}[t_{gomb} > \underline{\mu}^{(2)}$$

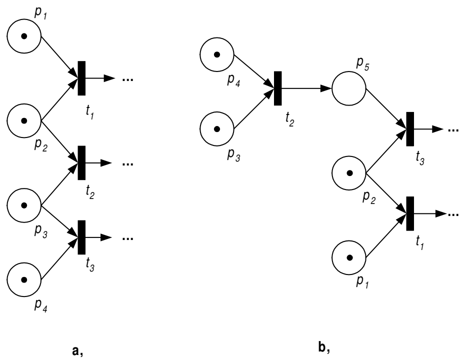
$$\underline{\mu}^{(1)} = [1, \; 0, \; 0, \; 0 \; ; \; 1, \; 0, \; 0, \; 0]^T$$

$$\underline{\mu}^{(2)} = [0, \; 1, \; 0, \; 0 \; ; \; 0, \; 0, \; 0, \; 0]^T$$

The marking vector contains the token quantities of the places in the given order. $\underline{\mu}^{(1)} = [1, \; 0, \; 0, \; 0 \; ; \; 1, \; 0, \; 0, \; 0]^T$ means that there is 1 token on place $p_{autovar}$ and p_{autobe}, and 0 token on the other places. The places referring to the garage and the car driver are separated by ; .

Parallel events

More than one enabled (fireable) transition:
concurrency (independent conditions), conflict, confusion



If more than one transition is enabled at the same time, then the firing of one may affect the firing of the other. The following cases may occur:

- **concurrency**: the firing of a transition does not affect the enabling of the other transition. The two transitions have independent preconditions, i.e. they have no common input places. *Example*: t_1 and t_3 in Figure a.
- **conflict**: the firing of one transition cancels the enabling of the other transition. The transitions have at least one common input place with less tokens than it is required for the firing of both transitions. *Example*: t_1 and t_2 in Figure a. The common place is p_2 with one token. If t_1 fires first, then it removes one token from p_1 and p_2 therefore t_2 is no more enabled. However if there are 2 tokens on p_2 then after the firing of t_1 one token still remains on p_2, hence t_2 can fire too. The same situation applies to t_2 and t_3.
- **confusion**: sometimes the situation is not clear, as two transitions may be in concurrent or conflict according to the firing order. *Example*: in Figure b, there are 3 transitions. It can be seen, that t_1 and t_2 are concurrent, because they do not affect the firing of each other. If t_1 fires first, then t_3 will never be enabled. However if t_2 fires first, then t_1 and t_3 both become enabled in a conflicted situation.

Conflict resolution

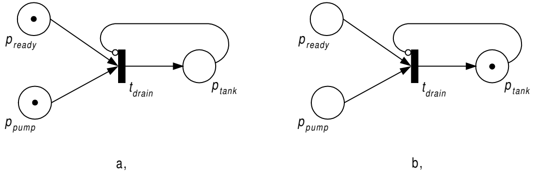
Using **inhibitor edges**:

priority given by the user

test edges

Other solutions:

capacity of the places

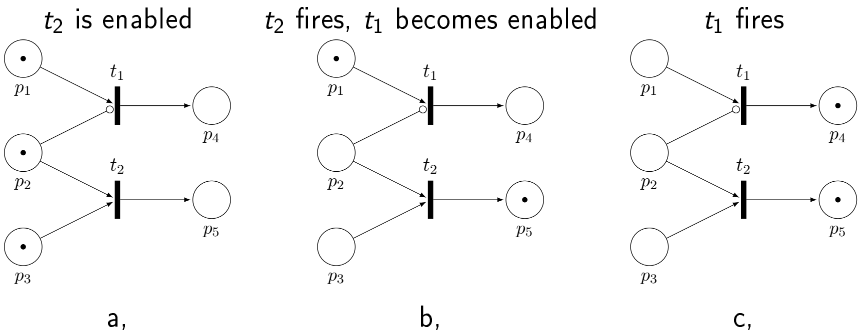


Conflict situation are usually not preferred in a system, because it makes the operation non deterministic. Possible resolutions of conflicts are:

- **inhibitor edges**: an inhibitor edge is the opposite of the usual edge. A transition whose input place is connected with an inhibitor edge is enabled if there is *no token* on that place. *Example*: in Figure a and b you can see the Petri net model of the filling of a tank. p_{tank} is connected to t_{drain} with an inhibitor edge. This means that t_{tank} is enabled if there is one token on p_{ready} AND p_{pump} AND p_{tank} is *empty*. In Figure b, you can see the marking after the firing of t_{tank}.

Conflict resolution

Inhibitor edges - Example

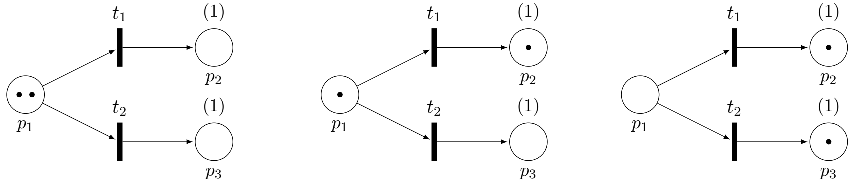


The firing sequence can be defined with inhibitor edges. Initially only t_2 is enabled, because there is one token on p_2 and p_3. t_1 is not enabled, because there is one token on p_2 that is connected to t_1 with an inhibitor edge (Figure a). After t_2 fired t_1 becomes enabled, because the token from p_2 is removed by t_2 (Figure b). Now t_1 can fire and the final marking of the Petri net can be seen on Figure c.

Conflict resolution

Capacity of places - Example

t_1 and t_2 are enabled t_1 fires, only t_2 is enabled t_2 fires

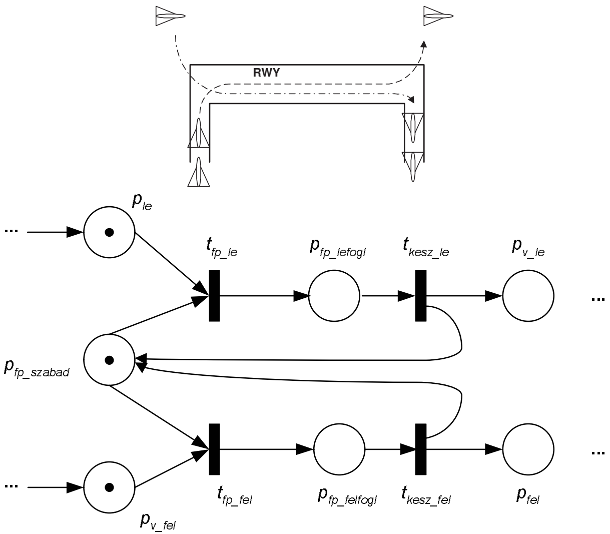


a, b, c,

The capacity of the places can be seen between round brackets. In this example p_2 and p_3 has a capacity of 1, which means only one token can be there at the same time. p_1 has unlimited capacity.
Initially there are 2 tokens on p_1 and both t_1 and t_2 are enabled (Figure a). Lets fire t_1.
After the firing of t_1, p_2 is full, and cannot receive more tokens, because of its capacity. Therefore only t_2 can fire (Figure b). After firing t_2 the final marking can be seen in Figure c.

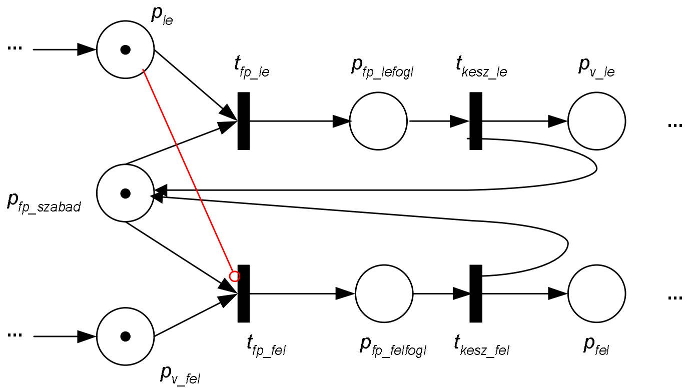
Note, that the initial conflict was not resolved, but at the second step (Figure b) the capacity of p_2 prevented the firing of t_1.

Petri net model of a runway – 1



Petri net model of a runway – 2

Conflict resolution: landing aircraft has priority



The conflict resolution with inhibitor is demonstrated on the runway example. An obvious solution is that the landing aircraft has priority. If the runway is free, the landing aircraft lands first. The aircraft waiting for take off can use the runway only if there is no landing aircraft.

The inhibitor edge from p_{le} to t_{fp_fel} realizes this solution.

Overview - Solution of Petri net models

The solution problem

Abstract problem statement

Given:

- a *formal description* of a discrete event system model
- *initial state(s)*
- *external events*: system inputs

Compute:

- the sequence of *internal (state and output) events*

The solution is **algorithmic**! **The problem is NP-hard!**

The general problem description of the solution of a discrete event system is given here. We need to know all of the following 3 things to clearly determine the solution.

- the **formal description** of the DES, e.g. an automaton or a Petri net
- the **initial state** of the system, to know where to start the simulation
- the **external events** which are the system inputs that operates the system.

The task is to compute the sequence of internal events, which are the changes in the states and the outputs of the system.

Unlike the solution of state space models, the solution of a discrete event system is **algorithmic**. It means that the solution is given by executing the steps of a (simulation) algorithm, instead of solving a set of differential equations for example. The computational complexity of such problems may be NP-hard (nondeterministic polynomial-time hard)

Petri net models – reachability graph

Solution: marking (systems state) sequences
reachability graph (tree) (weighted directed graph)

- *vertices*: markings
- *edges*: if exists transition the firing of which connects them
- *edge weights*: the transition and the external events

Construction:

1. *start*: at the given initial state (marking)
2. *adding a new vertex*: by firing an enabled transition (with the effect of inputs!)

May be NP-hard (in conflict situation or non-finite operation)

- A vertex is terminal if there is no enabled transition at that state.

It can be seen that the construction may be NP hard, especially if there are conflicts and infinitely firing transitions. The size of the reachability graph may grow explosively!

The state space of Petri net models

State vector: marking in *internal* places
in- and out-degree is at least 1

$$x(k) \quad \sim \quad \underline{\mu}_x^{(k)}$$

Inputs: marking in *input* places
in-degree is zero

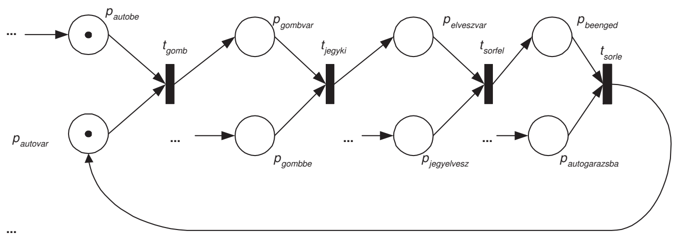
$$u(k) \quad \sim \quad \underline{\mu}_u^{(k)}$$

From the system theory point of view it is useful to separate the input/output variables from the state variables.
In a Petri net model, the state variables are the places with input AND output edges.

The input variables are represented by places with no input edges.

Example: garage gate

Petri net model



$$\underline{\mu}_x^T = [\mu_{autovar}, \mu_{gombvar}, \mu_{elveszvar}, \mu_{beenged}]$$
$$\underline{\mu}_u^T = [\mu_{autobe}, \mu_{gombbe}, \mu_{jegyelevesz}, \mu_{autogarazsba}]$$

Here you can see the marking vector of the garage gate example.

- $\underline{\mu}_x^T$ is the state vector, $\mu_{autovar}, \mu_{gombvar}, \mu_{elveszvar}, \mu_{beenged}$ are the markings of the state places.
- $\underline{\mu}_u^T$ is the input vector, $\mu_{autobe}, \mu_{gombbe}, \mu_{jegyelevesz}, \mu_{autogarazsba}$ are the markings on the input places

Remark: the places with input ...$\rightarrow$ are part of the car driver's Petri net, which is not presented here. From the gate's point of view they are input places, because they do not have input edges *in the gate's Petri net*!