Industrial Automation

(Automação de Processos Industriais)

Discrete Event Systems

http://www.isr.tecnico.ulisboa.pt/~jag/courses/api20b/api2021.html

Prof. Paulo Jorge Oliveira, original slides
Prof. José Gaspar, rev. 2020/2021

Syllabus:

Chap. 5 – CAD/CAM and CNC [1 week]

…

Chap. 6 – Discrete Event Systems [2 weeks]

Discrete event systems modeling. Automata.

Petri Nets: state, dynamics, and modeling.

Extended and strict models. Subclasses of Petri nets.

…

Chap. 7 – Analysis of Discrete Event Systems [2 weeks]

Some pointers to Discrete Event Systems

History: http://prosys.changwon.ac.kr/docs/petrinet/1.htm

Tutorial: http://vita.bu.edu/cgc/MIDEDS/
http://www.daimi.au.dk/PetriNets/

Analyzers, and Simulators: http://www.ppgia.pucpr.br/~maziero/petri/arp.html (in Portuguese)
http://wiki.daimi.au.dk:8000/cpntools/cpntools.wiki
http://www.informatik.hu-berlin.de/top/pnk/download.html

Bibliography:

* **Introduction to Discrete Event Systems**, Christos Cassandras and Stephane Lafortune. Springer, 2008.
* **Discrete Event Systems - Modeling and Performance Analysis,** Christos G. Cassandras, Aksen Associates, 1993.
* **Petri Net Theory and the Modeling of Systems,** James L. Petersen, Prentice-Hall,1981.
* **Petri Nets and GRAFCET: Tools for Modeling Discrete Event Systems** R. David, H. Alla, Prentice-Hall, 1992

Generic characterization of systems resorting to input / output relations

In some systems each input determines a **single output value**.

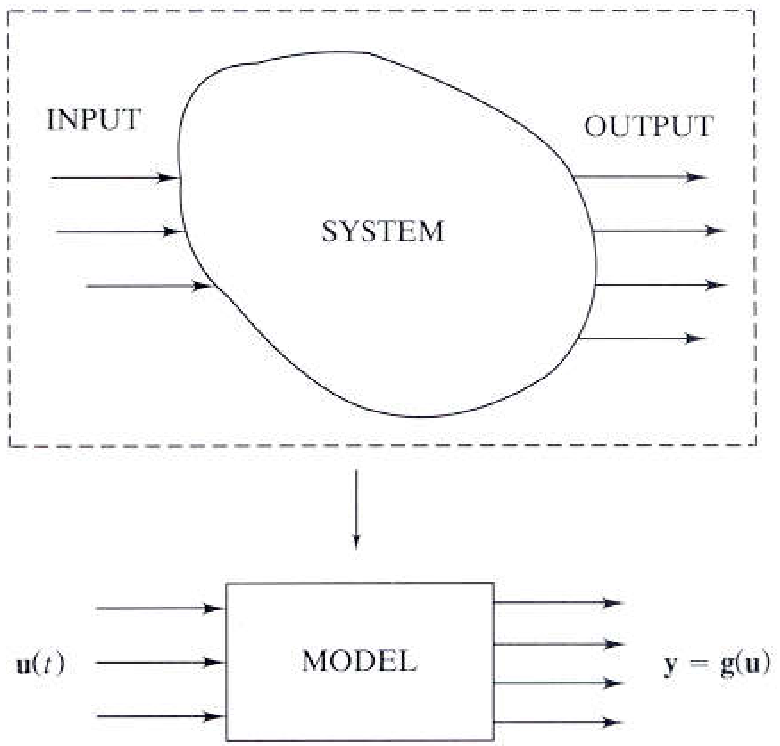


Figure 1.1. Simple modeling process.

Other systems are **dynamic**. An input implies a time evolving response.

Typically one uses state space equations:

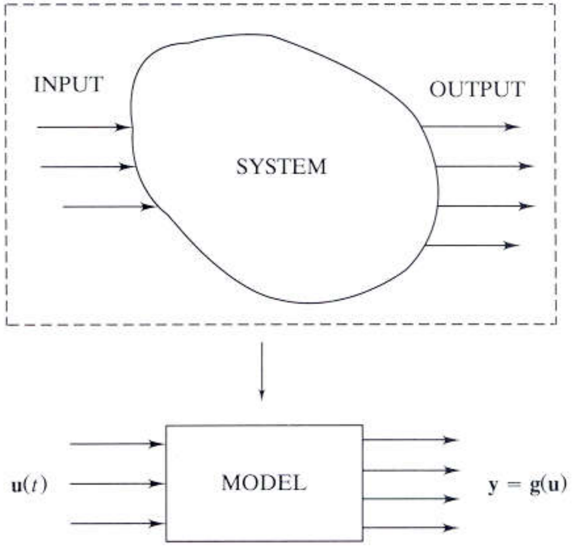
$$\begin{cases} \dot{x}(t) = f(x(t), u(t), t) \\ y(t) = g(x(t), u(t), t) \end{cases}$$

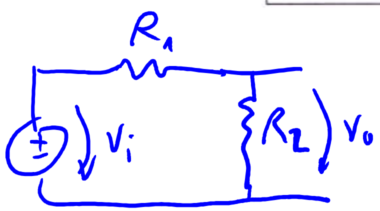
in continuous time (or in discrete time).

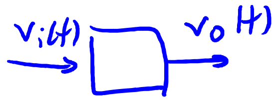
Example: **voltage divider** circuit vs **RC circuit** (capacitor charge circuit).
Given an input one cannot tell the capacitor voltage without knowing its initial condition.

Generic characterization of systems resorting to input / output relations

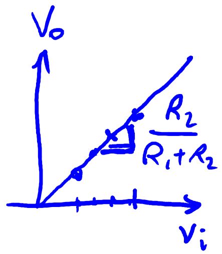
Case1: each input determines a **single output value**







$V_o = \frac{R_2}{R_1+R_2} V_i$, $V_o = f(V_i)$

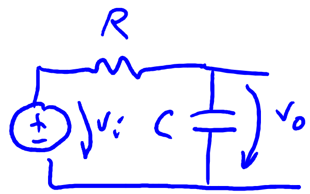


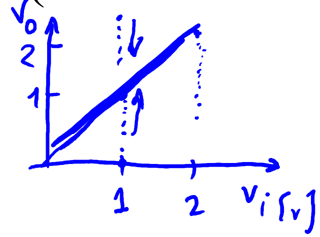
Case2: **dynamic system**, an input implies a time evolving response.

Typically, one uses state space equations:

$$\begin{cases} \dot{x}(t) = f(x(t), u(t), t) \\ y(t) = g(x(t), u(t), t) \end{cases}$$

in continuous time (or in discrete time).

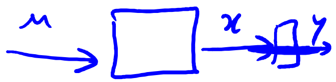




$v_i = R i_c + v_c$

$= R C \frac{dv_c}{dt} + v_c$

$u = \tau \dot{x} + x$



$\tau \dot{x} + x = u$

Example / Comment: ***voltage divider*** *circuit vs* ***RC circuit*** *(capacitor charge circuit), given an input one cannot tell the capacitor voltage without knowing its initial condition.*

Control: Open loop vs closed-loop (⇔ the use of feedback)

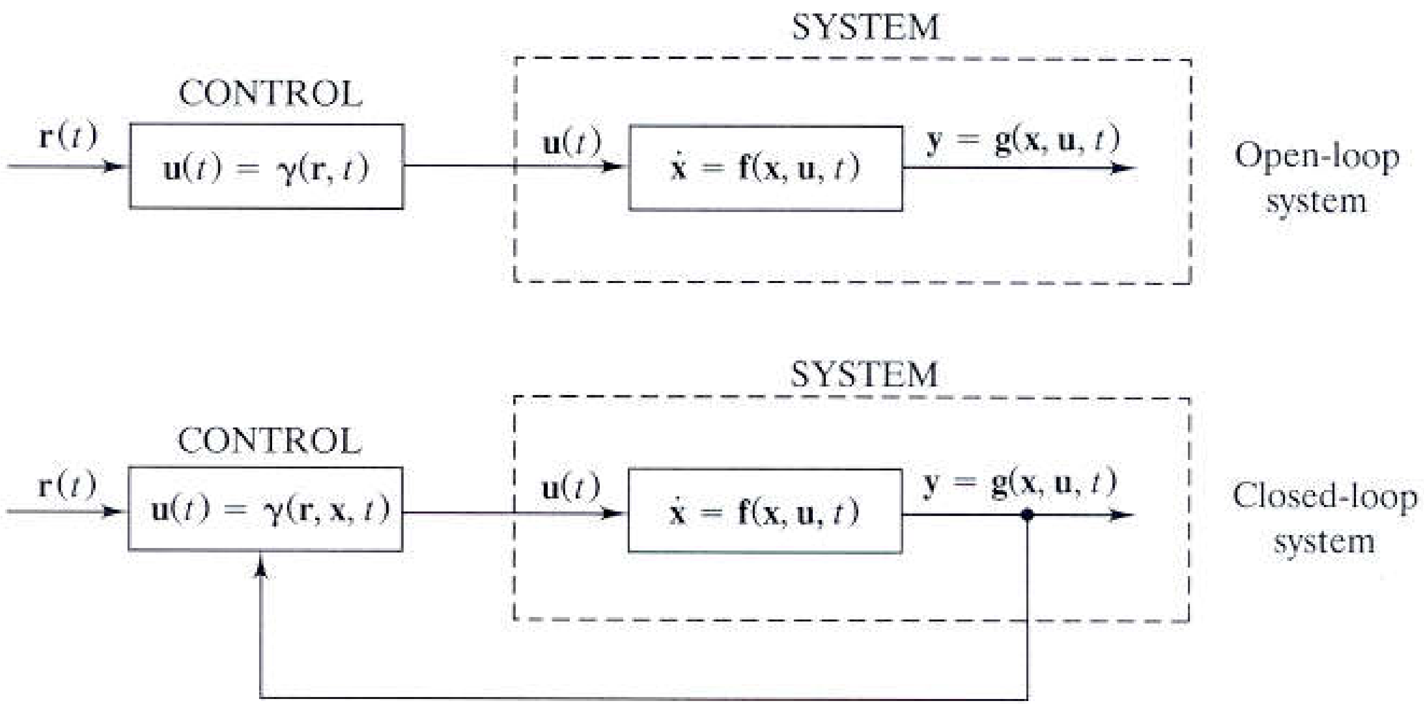


Figure 1.17. Open-loop and closed-loop systems.

Advantages of feedback? Approach model uncertainties, disturbances, etc.

Control will be revisited in the DES supervision chapter.

Example of closed-loop with feedback

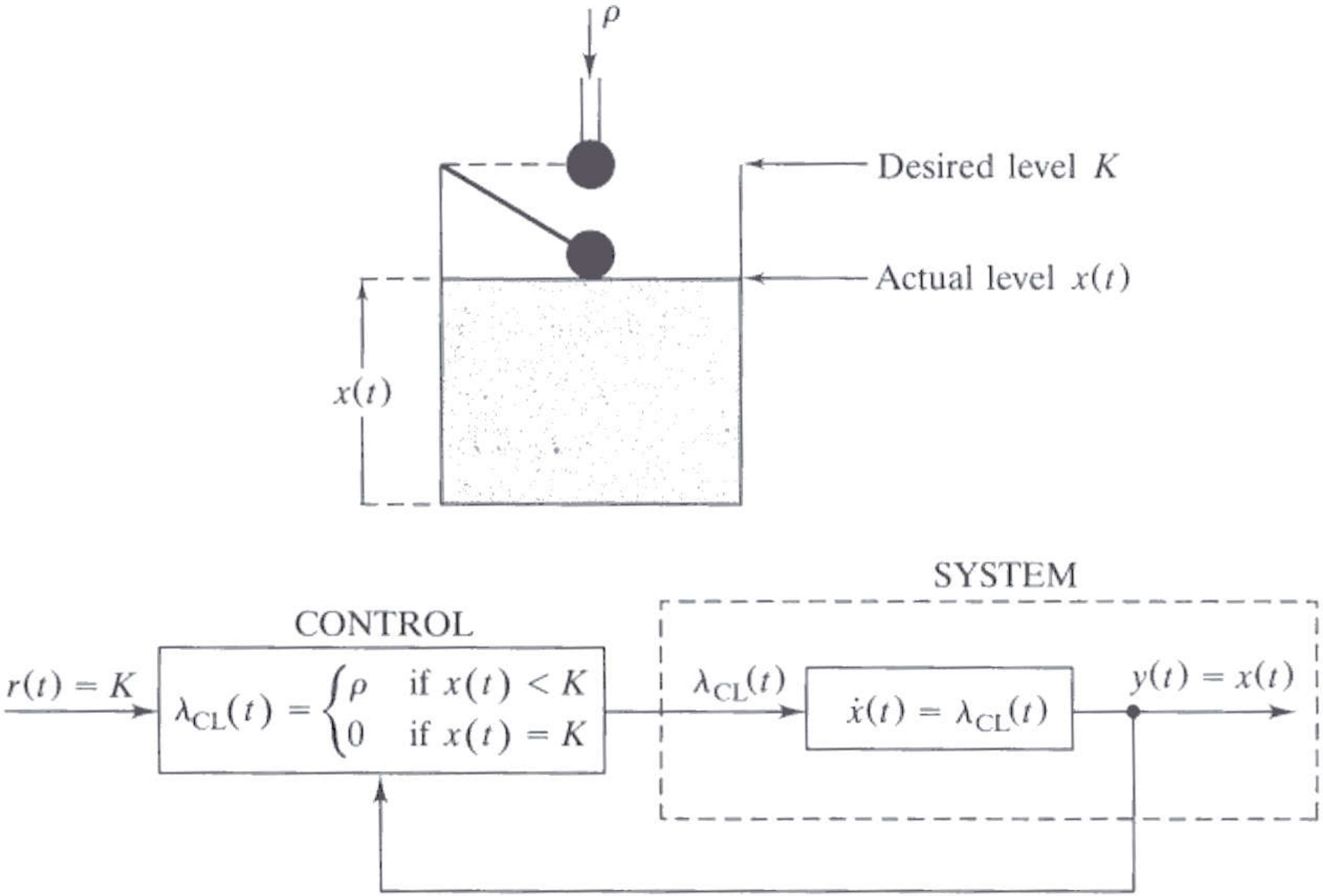


Figure 1.18. Flow system of Example 1.11 and closed-loop control model.

Discrete Event Systems: Examples

Consider e.g. a milk distribution truck in Manhattan. How to model its motion?

Set of events **E** = {N, S, E, W}

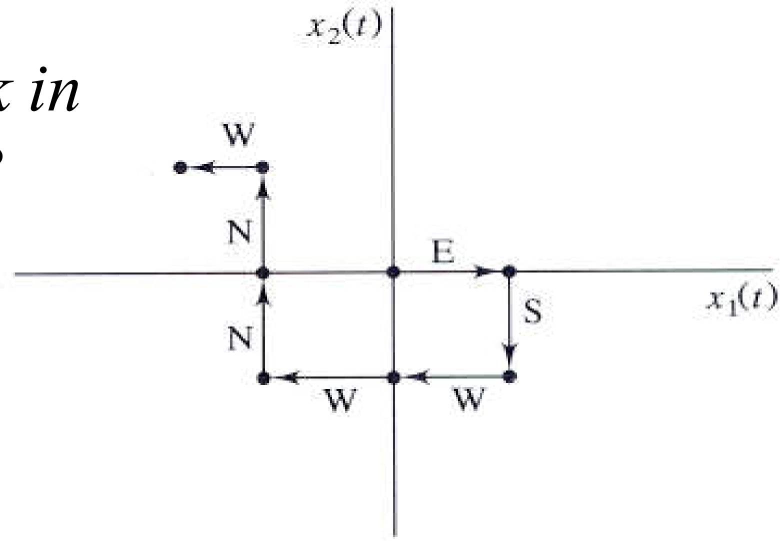


Figure 1.20. Random walk on a plane for Example 1.12.

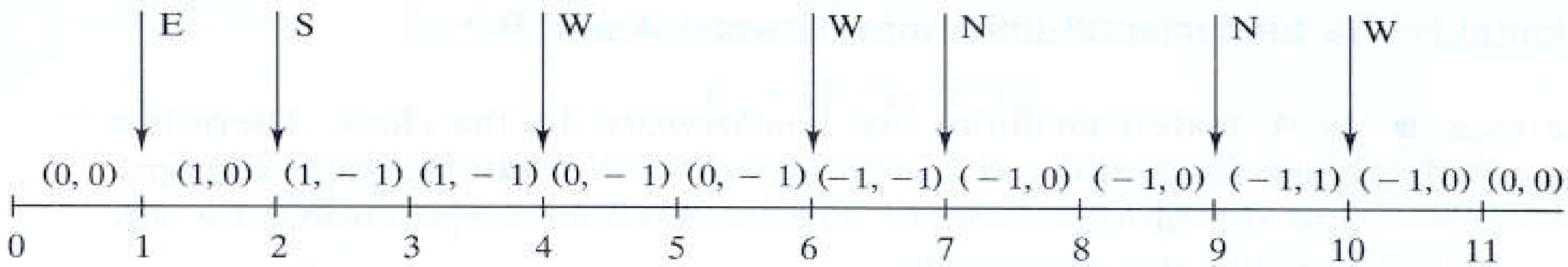
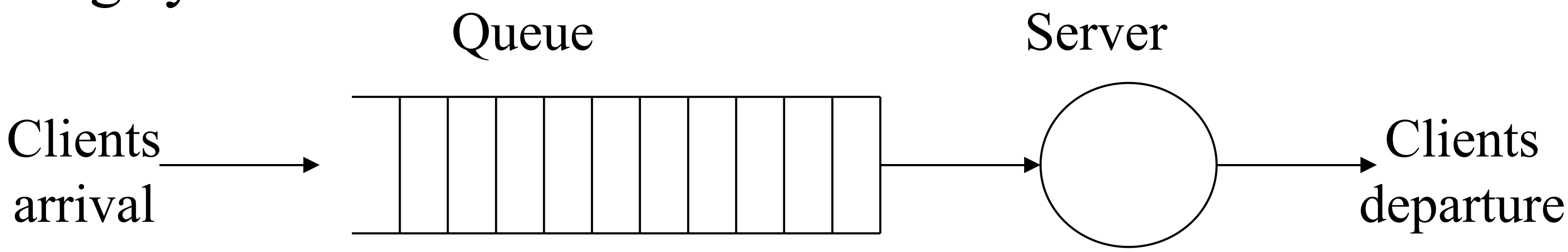


Figure 1.21. Event-driven random walk on a plane.

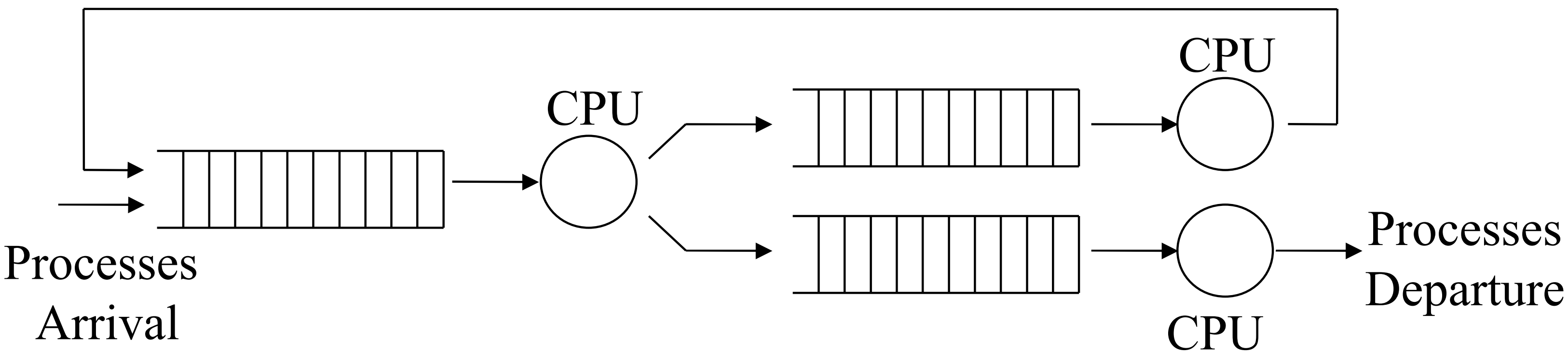
Discrete Event Systems: Examples

Queueing systems



Set of events, **E** = {arrival, departure}

Computational Systems



Characteristics of systems with continuous variables

1. State space is continuous

2. The state transition mechanism is ***time-driven***

Characteristics of systems with discrete events (DES)

1. State space is discrete

2. The state transition mechanism is ***event-driven***

Intrinsic characteristic of discrete events systems: Polling is avoided!

Taxonomy of Systems

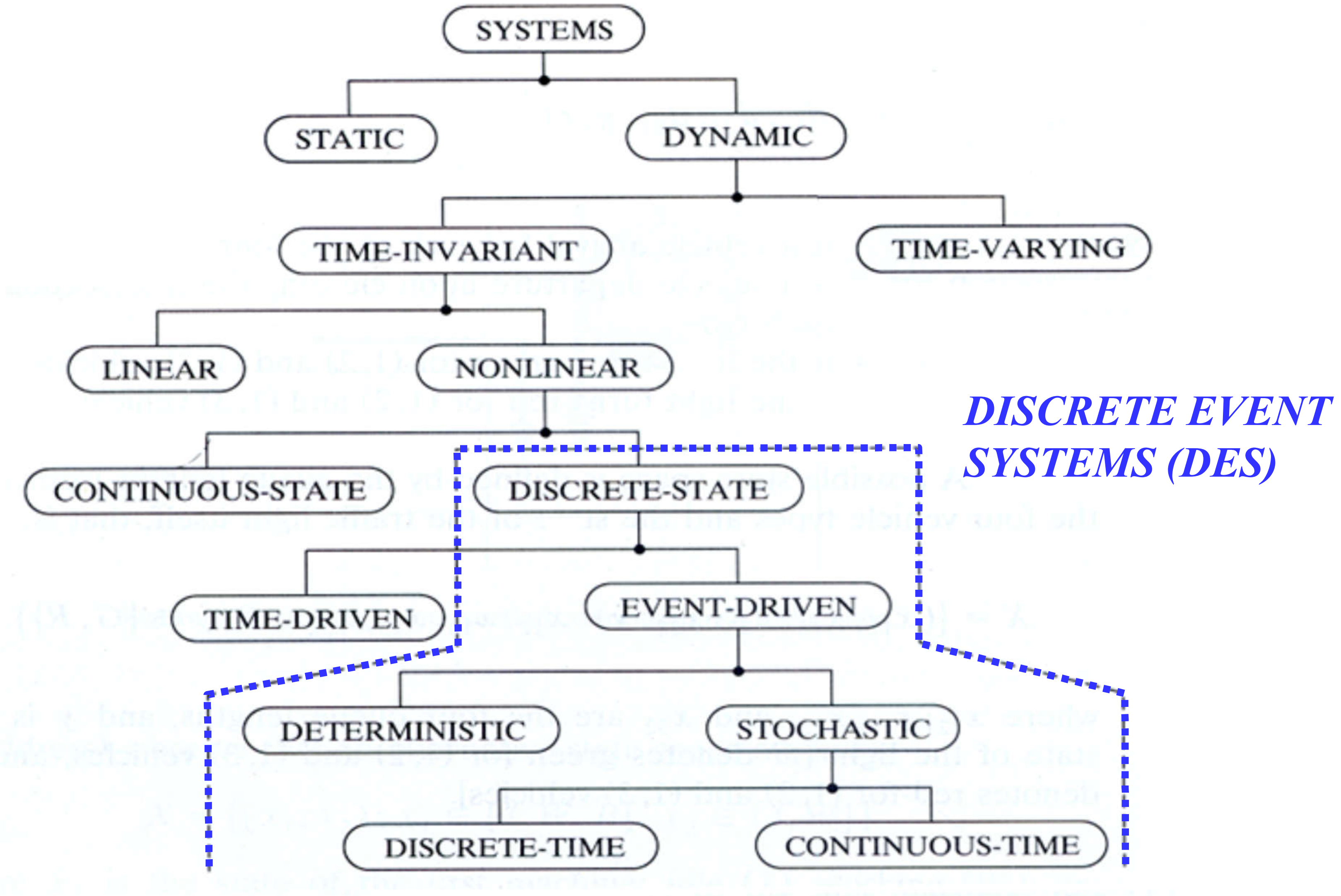


Figure 1.29: Major system classification

Levels of abstraction in the study of Discrete Event Systems

Example 1: Language of a
"chocolate selling machine":

(i) Waiting for a coin.
(ii) *Received 1 euro coin.*
Chocolate A given. Go to (i).
(iii) *Received 2 euro coin.*
Chocolate B given. Go to (i).

2 actuators:
Give chocolate A
Give chocolate B

4 sensors:
Received 1 euro coin,
Received 2 euro coin,
Chocolate A given,
Chocolate B given.

Q: How to model
(i) a self playing piano / "pianola",
(ii) a recognizer of digits spoken by a person?

Languages

↓

Timed languages

↓

Stochastic timed languages

Systems Theory Objectives

- Modeling and Analysis
- *Design* and synthesis
- Control / Supervision
- Performance assessment and robustness
- Optimization

Applications of Discrete Event Systems

- Queueing systems
- Operating systems and computers
- Telecommunications networks
- Distributed databases
- Automation

Discrete Event Systems

Typical modeling methodologies

Automata

GRAFCET / SFC

Petri nets

Augmenting in

modeling capacity

&

complexity

Automata Theory and Languages

Genesis of computation theory

Definition: A **language L**, defined over the alphabet **E** is a **set of *strings*** of finite length with events from **E**.

Examples: **E**= {α, β, γ}

L_1= {ε, αα, αβ, γβα}, where ε is the null/empty string

L_2= {all *strings* of length 3}

How to build a machine that "talks" a given language?

or

What language "talks" a system?

Operations / Properties of languages

E^* = **Kleene-closure** of E: set of all strings of finite length of E, including the null element ε.

Concatenation of L_a and L_b:

$$L_a L_b := \left\{ s \in E^* : s = s_a s_b , s_a \in L_a , s_b \in L_b \right\}$$

Prefix-closure of $L \subseteq E^*$:

$$\overline{L} := \left\{ s \in E^* : \exists_{t \in E^*} \, st \in L \right\}$$

Operations / Properties of languages

Example 2.1 (Operations on languages)
Let $E = \{a, b, g\}$, and consider the two languages $L_1 = \{\varepsilon, a, abb\}$ and $L_4 = \{g\}$. Neither L_1 nor L_4 are prefix-closed, since $ab \notin L_1$ and $\varepsilon \notin L_4$. Then:

$$
\begin{aligned}
L_1 L_4 &= \{g, ag, abbg\} \\
\overline{L_1} &= \{\varepsilon, a, ab, abb\} \\
\overline{L_4} &= \{\varepsilon, g\} \\
L_1 \overline{L_4} &= \{\varepsilon, a, abb, g, ag, abbg\} \\
L_4^* &= \{\varepsilon, g, gg, ggg, \ldots\} \\
L_1^* &= \{\varepsilon, a, abb, aa, aabb, abba, abbabb, \ldots\}
\end{aligned}
$$

[Cassandras99]

Automata Theory and Languages

Motivation: *An* ***automaton*** *is a device capable of representing a language according to some rules.*

Definition: A deterministic **automaton** is a 5-*tuple*

$$(\mathbf{E}, \mathbf{X}, \mathbf{f}, \mathbf{x_0}, \mathbf{F})$$

where:

E	- finite alphabet (or possible events)	
X	- finite set of states	
f	- state transition function	$\mathbf{f}: \mathbf{X} \times \mathbf{E} \rightarrow \mathbf{X}$
$\mathbf{x_0}$	- initial state	$\mathbf{x_0} \subset \mathbf{X}$
F	- set of final states or marked states	$\mathbf{F} \subset \mathbf{E}$

[Cassandras93]

Word of caution: the word “state” is used here to mean “step” (Grafcet) or “place” (Petri Nets)

Example 1 of an automaton:

input event

current state	**α**	**β**	**γ**
x	*x*	*z*	*z*
y	*x*	*y*	*y*
z	*y*	*z*	*y*

next state

(E, X, f, x_0, F)

$\mathbf{E} = \{\alpha, \beta, \gamma\}$
$\mathbf{X} = \{x, y, z\}$
$\mathbf{x_0} = x$
$\mathbf{F} = \{x, z\}$

$f(x, \alpha) = x$ $f(x, \beta) = z$ $f(x, \gamma) = z$
$f(y, \alpha) = x$ $f(y, \beta) = y$ $f(y, \gamma) = y$
$f(z, \alpha) = y$ $f(z, \beta) = z$ $f(z, \gamma) = y$

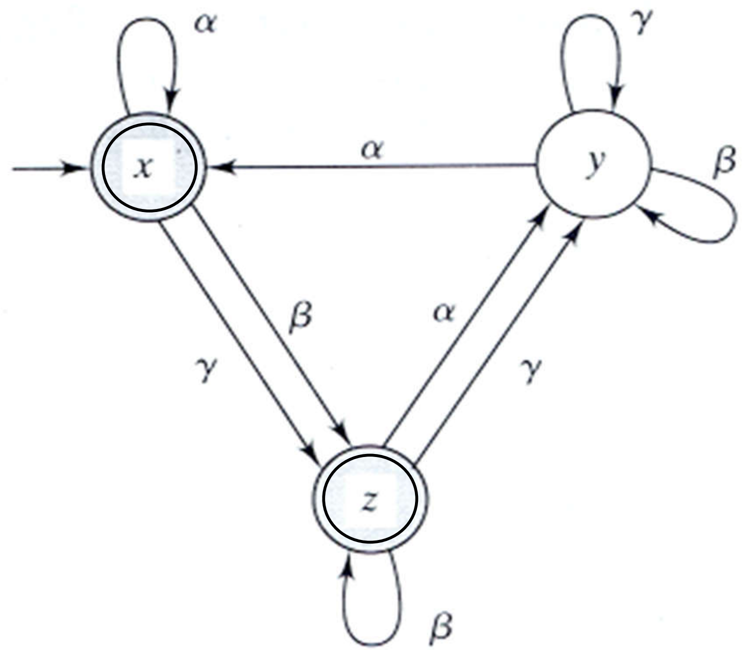


Figure 2.1. State transition diagram for Example 2.3.

Example 2 of a stochastic automaton

(E, X, f, x_0, F)

$\mathbf{E} = \{\alpha, \beta\}$

$\mathbf{X} = \{0, 1\}$

$\mathbf{x_0} = 0$

$\mathbf{F} = \{0\}$

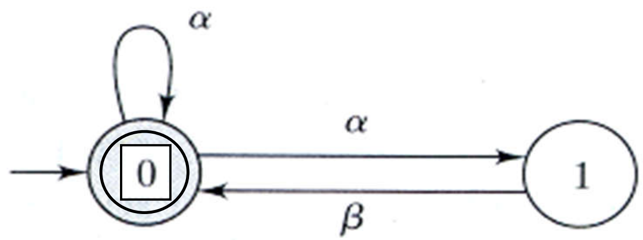


Figure 2.4. State transition diagram for the nondeterministic automaton of Example 2.7.

$f(0, \alpha) = \{0, 1\}$ $f(0, \beta) = \{\}$
$f(1, \alpha) = \{\}$ $f(1, \beta) = 0$

Given an automaton

$$\mathbf{G = (E, X, f, x_0 ,F)}$$

the **Generated Language** is defined as

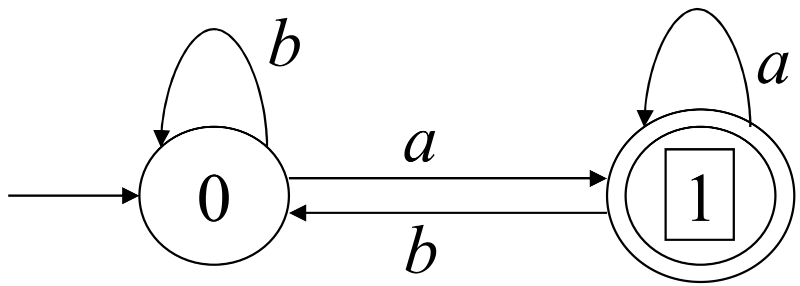
$$\boldsymbol{L}(G) := \{s \in E^* : f(x_0,s) \text{ is defined}\}$$

*Note: if f is always defined for all events then L(G)==E**

and the **Marked Language** is defined as

$$\boldsymbol{L_m}(G) := \{s \in E^* : f(x_0,s) \in F\}$$

Example 3: marked language of an automaton



$$L(\text{G}) := \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, baa, \ldots\}$$

$$L_m(\text{G}) := \{a, aa, ba, aaa, baa, bba, \ldots\}$$

Concluding, in this example L_m(G) means all strings with events *a* and *b*, ended by event *a*.

Automata equivalence:

The automata G_1 e G_2 are **equivalent** if

$$\boldsymbol{L}(\mathrm{G}_1) = \boldsymbol{L}(\mathrm{G}_2)$$

and

$$\boldsymbol{L}_\mathrm{m}(\mathrm{G}_1) = \boldsymbol{L}_\mathrm{m}(\mathrm{G}_2)$$

Example 4: two equivalent automata

Objective: To validate a sequence of events

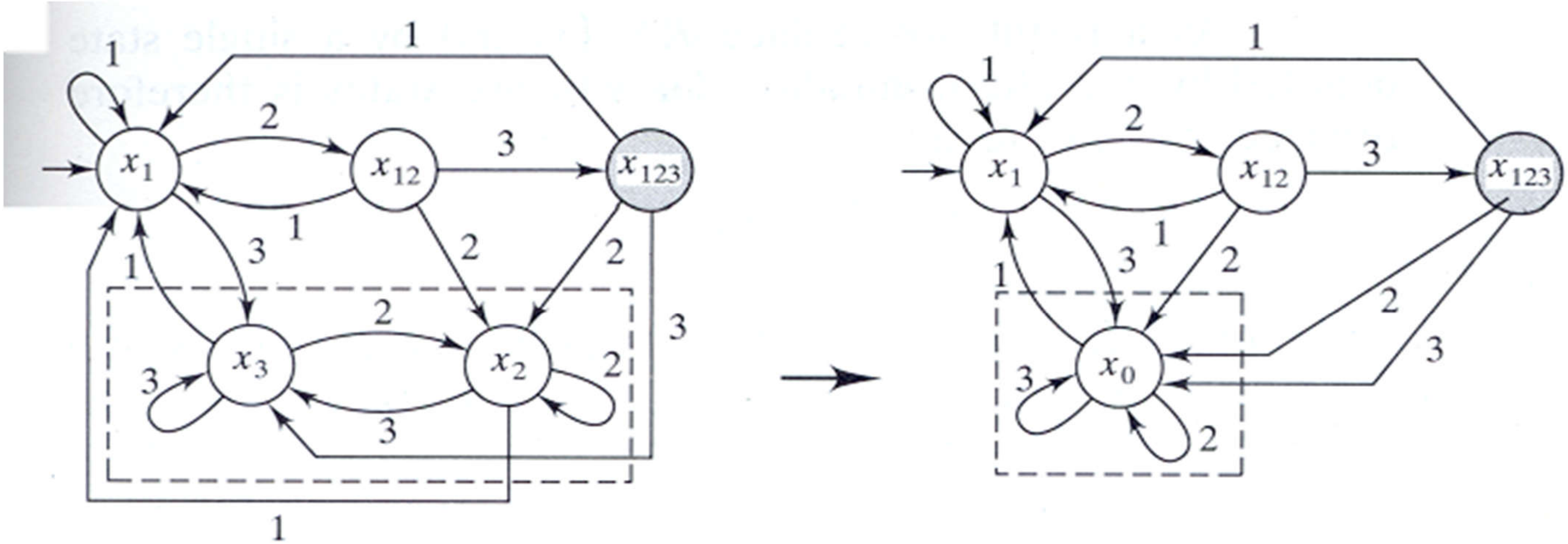
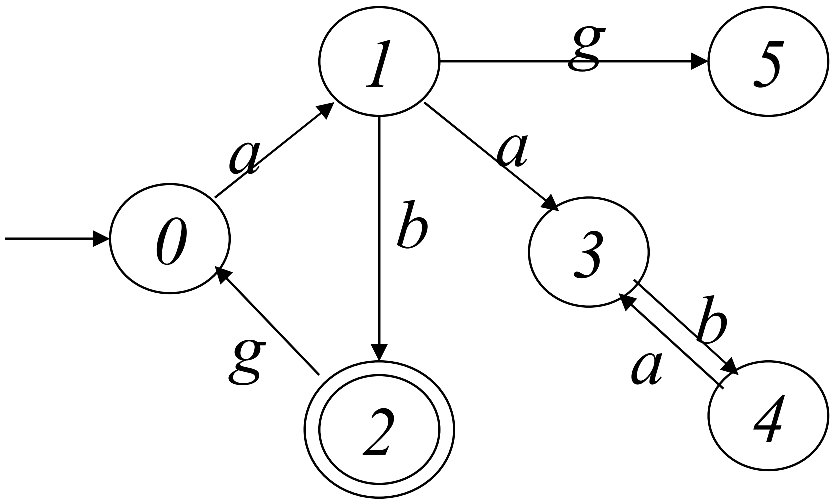


Figure 2.6. State transition diagrams for digit sequence detector in Example 2.9.

Deadlocks (*inter-blocagem*)

Example 5:



The state *5* is a *deadlock*.

The states *3* and *4* constitute a *livelock*.

How to find the *deadlocks* and the *livelocks*?

Need methodologies for the analysis of Discrete Event Systems

Deadlock:

in general the following relations are verified

$$L_m(G) \subseteq \overline{L}_m(G) \subseteq L(G)$$

An automaton G has a **deadlock** if

$$\overline{L}_m(G) \subset L(G)$$

and is **not blocked** when

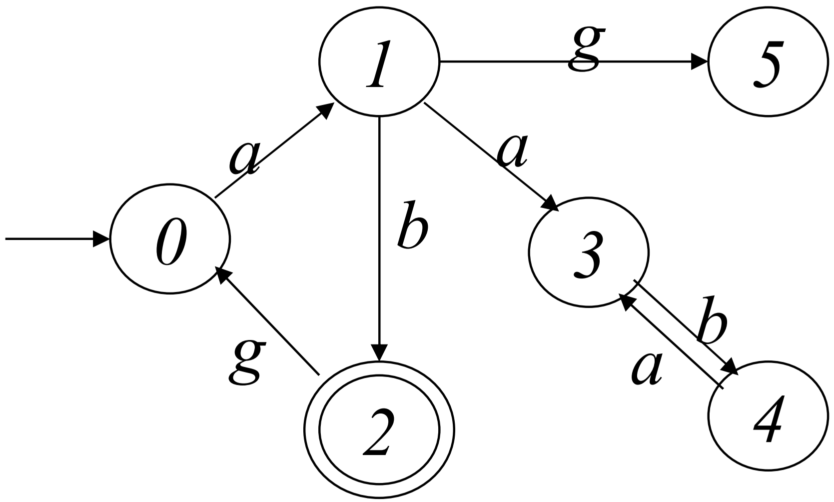
$$\overline{L}_m(G) = L(G)$$

Deadlock:

$$L_m(G) = \{ab, abgab, abgabgab, \ldots\}$$

Example:

$$L(G) = \left\{ \begin{matrix} \varepsilon, a, ab, ag, aa, aab, \\ abg, aaba, abga, \ldots \end{matrix} \right\}$$



$$\left(L_m(G) \subset L(G)\right)$$

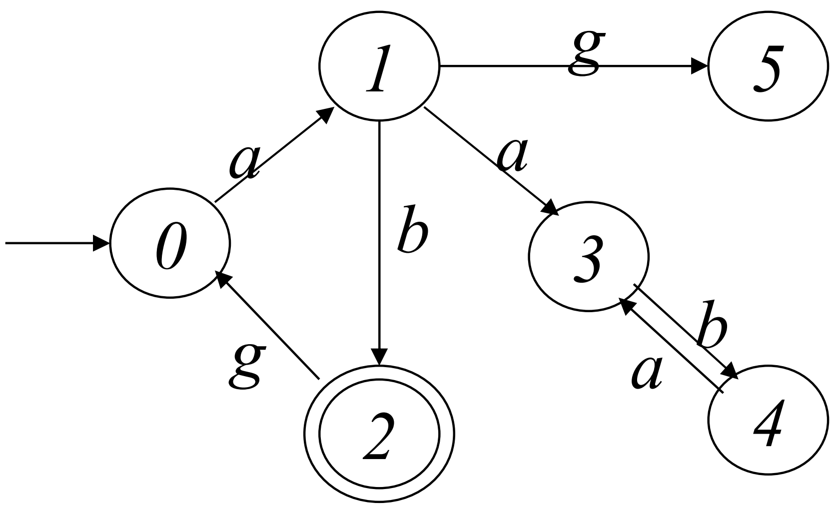
The state *5* is a *deadlock.*

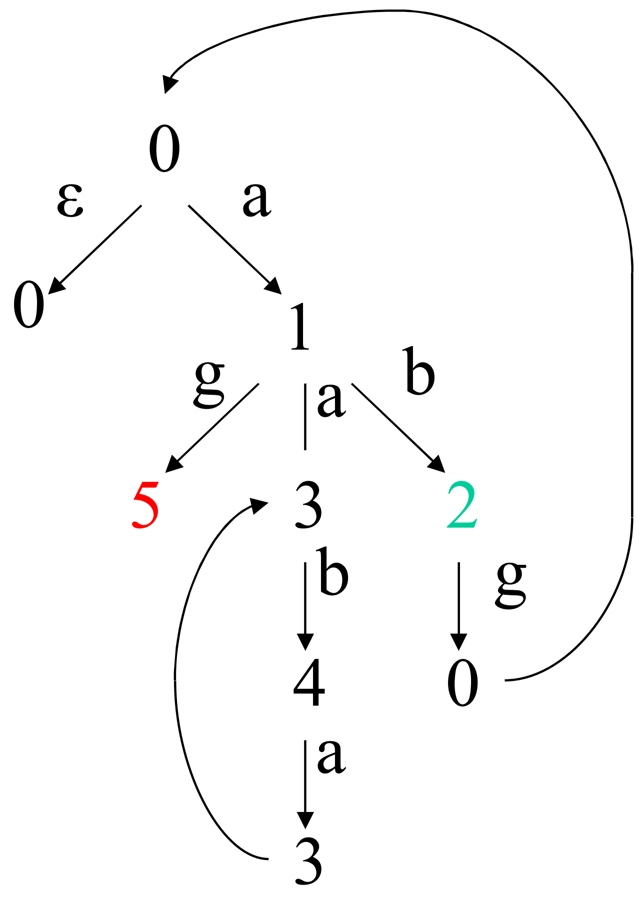
$$\overline{L}_m(G) \neq L(G)$$

The states *3* and *4* constitute a *livelock*.

Alternative way to detect deadlocks:

Example:





The state *5* is a *deadlock*.

The states *3* and *4* constitute a *livelock*.

Timed Discrete Event Systems

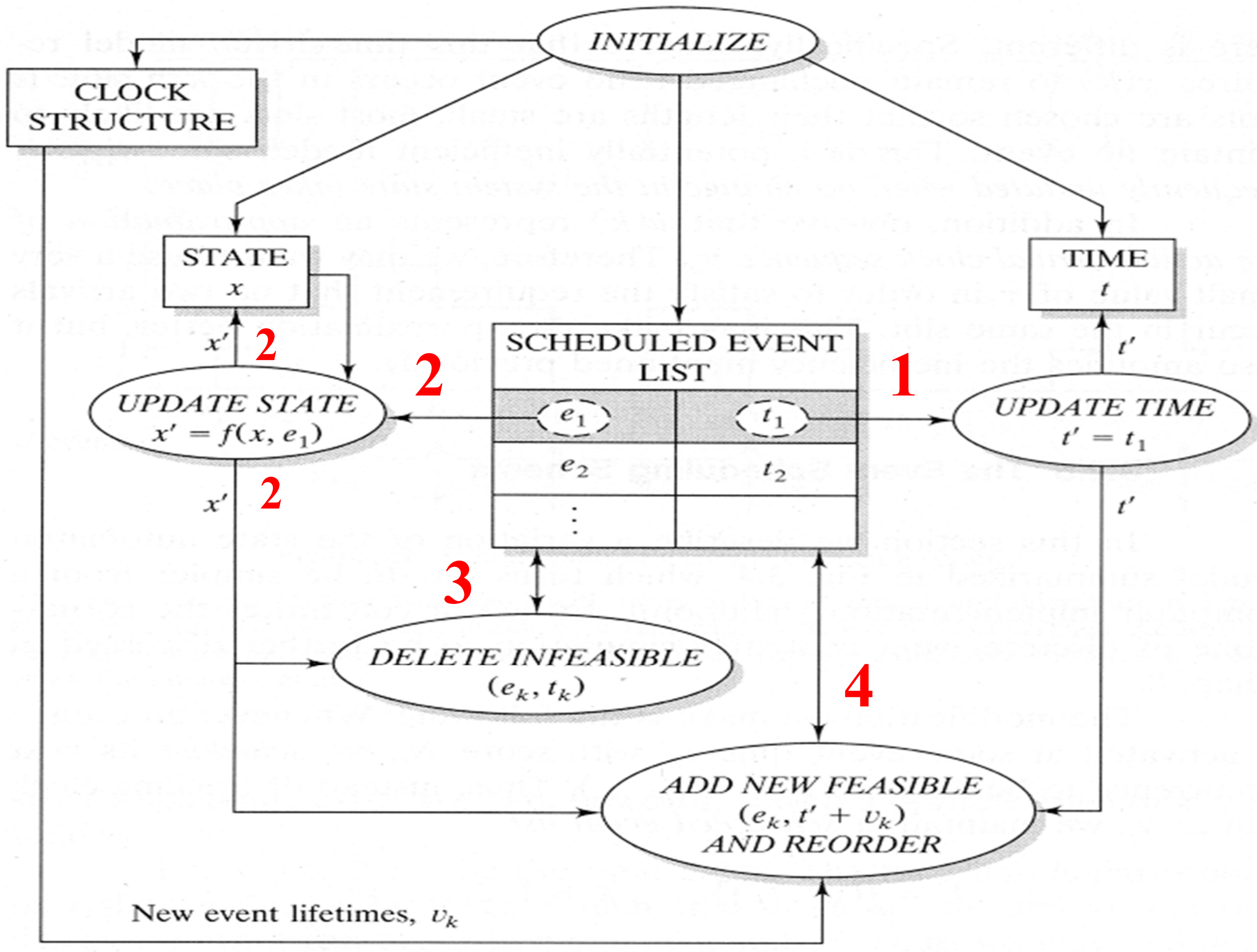


Figure 3.10. The event scheduling scheme.

Examples of Automata Classes and Applications

Automaton Class	Recognizable language	Applications	
Finite state machine (FSM), e.g. Moore machines or Mealy machines	Regular languages	Text processing, compilers, and hardware design	*Very small memory (just the state / number of states)*
Pushdown automaton (PDA)	Context-free languages	Programming languages, artificial intelligence, (originally) study of the human languages	*Memory : ∞ Stack*
Turing machine (nondeterministic, deterministic, multitape, ...)	Recursively enumerable languages	Theory, complexity	*Memory : ∞ Tape*

Another development direction: ***parallelism*** *(next slides)*

Petri nets

Developed by Carl Adam Petri in his PhD thesis in 1962.

Definition: A marked Petri net is a *5-tuple*

$$(\mathbf{P}, \mathbf{T}, \mathbf{A}, \mathbf{w}, \mathbf{x_0})$$

where:

P	- set of places	
T	- set of transitions	
A	- set of arcs	$A \subset (P \times T) \cup (T \times P)$
w	- weight function	$w: A \rightarrow \mathbb{N}$
$\mathbf{x_0}$	- initial marking	$x_0 : P \rightarrow \mathbb{N}$

[Cassandras93]

Example of a Petri net

(P, T, A, w, x_0)

$P=\{p_1, p_2, p_3, p_4, p_5\}$

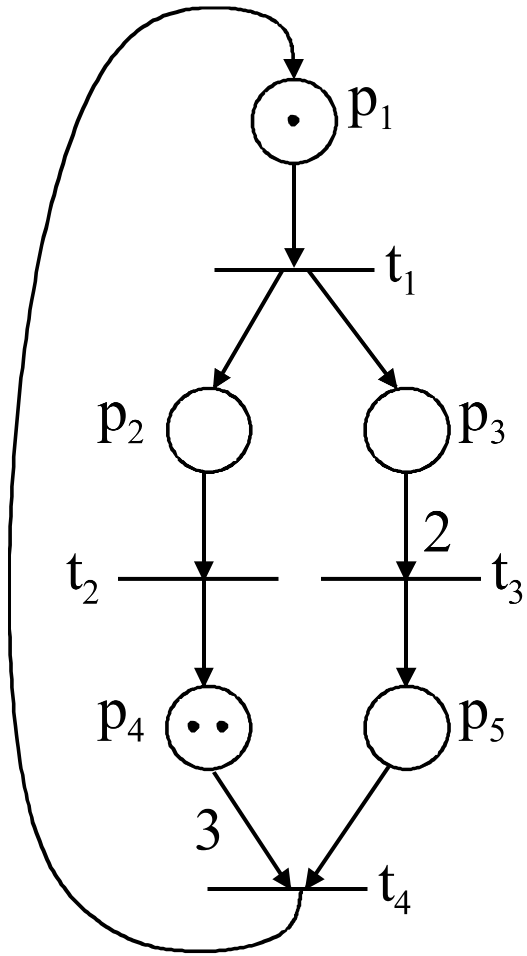
$T=\{t_1, t_2, t_3, t_4\}$

$A=\{(p_1, t_1), (t_1, p_2), (t_1, p_3), (p_2, t_2), (p_3, t_3),$
$(t_2, p_4), (t_3, p_5), (p_4, t_4), (p_5, t_4), (t_4, p_1)\}$

$w(p_1, t_1)=1, w(t_1, p_2)=1, w(t_1, p_3)=1, w(p_2, t_2)=1$
$w(p_3, t_3)=2, w(t_2, p_4)=1, w(t_3, p_5)=1, w(p_4, t_4)=3$
$w(p_5, t_4)=1, w(t_4, p_1)=1$

$x_0 = \{1, 0, 0, 2, 0\}$

Petri net graph



Petri nets

Rules to follow to create a Petri net:

- **Arcs** indicate directed connections
  connect **places** to **transitions** and
  connect **transitions** to **places**

- A transition can have no places directly as inputs (source) ,
  i.e. must exist ***arcs*** *between transitions and places*
- A transition can have no places directly as outputs (sink),
  i.e. must exist ***arcs*** *between transitions and places*

- The same happens with the input and output transitions for places

Alternative definition of a Petri net

A marked Petri net is a *5-tuple*

$$(\mathbf{P}, \mathbf{T}, \mathbf{I}, \mathbf{O}, \mu_0)$$

where:

P	- set of places	
T	- set of transitions	
I	- transition input function	$\mathbf{I} : \mathbf{T} \rightarrow \mathbf{P}^\infty$
O	- transition output function	$\mathbf{O} : \mathbf{T} \rightarrow \mathbf{P}^\infty$
μ_0	- initial marking	$\mu_0 : \mathbf{P} \rightarrow \mathbf{N}$

[Peterson81]

Note: $\mathbf{P}^\infty$ = bag of places (is more general than a set of places)

Example of a Petri net and its graphical representation

Alternative definition

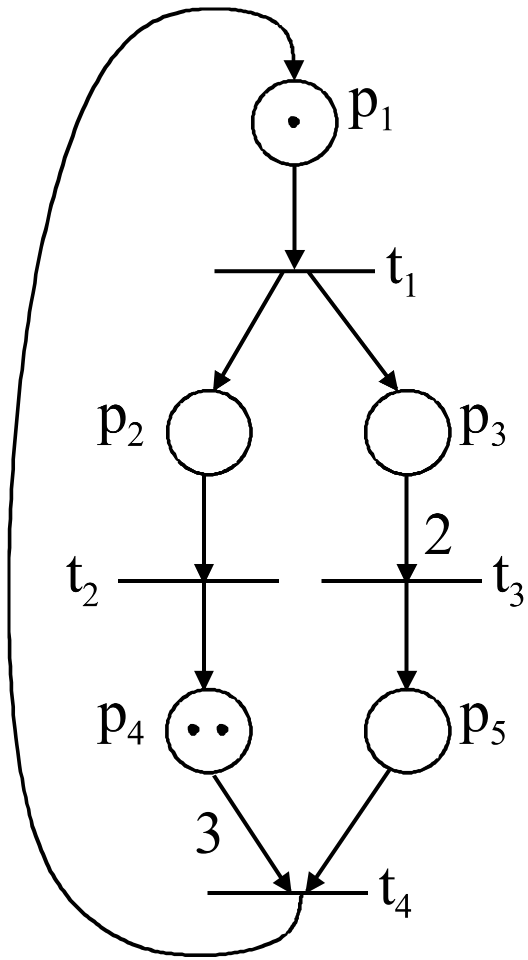
(P, T, I, O, μ_0)

$P=\{p_1, p_2, p_3, p_4, p_5\}$

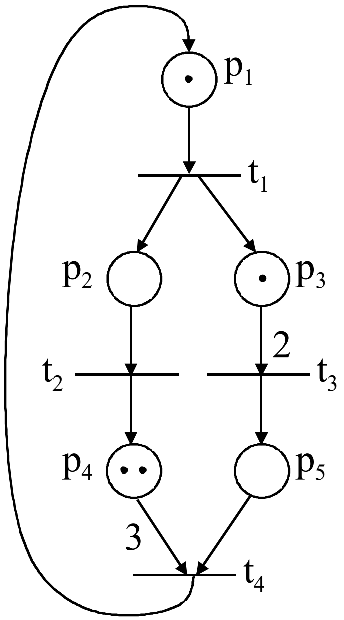
$T=\{t_1, t_2, t_3, t_4\}$

$I(t_1)=\{p_1\}$ $O(t_1)=\{p_2 , p_3\}$
$I(t_2)=\{p_2\}$ $O(t_2)=\{p_4\}$
$I(t_3)=\{p_3, p_3\}$ $O(t_3)=\{p_5\}$
$I(t_4)=\{p_4, p_4, p_4, p_5\}$ $O(t_4)=\{p_1\}$

$\mu_0 = \{1, 0, 0, 2, 0\}$



Petri nets: State, Markings, Weights of Arcs



The **state** of a Petri net is characterized by the marking of all places

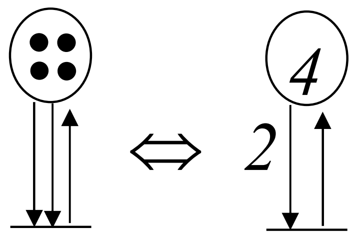
$\mu = (\mu_{p1}, \mu_{p2}, \mu_{p3}, \mu_{p4}, \mu_{p5})$

The set of all possible markings of a Petri net corresponds to its **state space**:

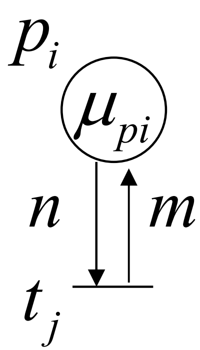
{(1,0,1,2,0), (0,1,2,2,0), (0,0,0,3,1), (1,0,0,0,0)}

How does the state of a Petri net evolve?

Simplifying notation of markings and cardinality (weight) of the arcs:



Formal nomenclature:



$n = \#(p_i, I(t_j))$

$m = \#(p_i, O(t_j))$

Execution Rules for Petri Nets (Dynamics of Petri nets)

A transition $t_j \in T$ is ***enabled*** if:

$$\forall p_i \in P: \quad \mu(p_i) \quad \geq \quad \#(p_i, I(t_j))$$

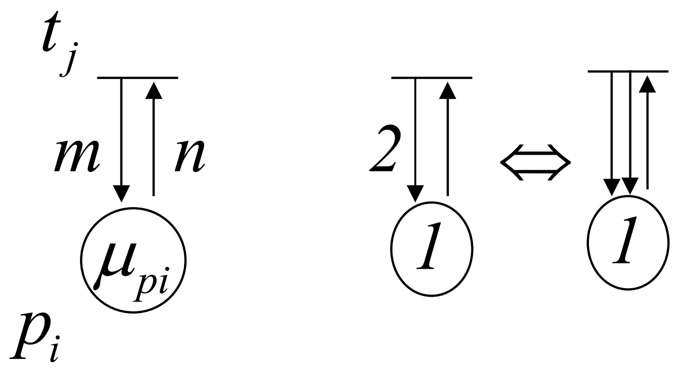
A transition $t_j \in T$ may ***fire*** whenever enabled, resulting in a new marking given by:

$$\mu'(p_i) \quad = \quad \mu(p_i) \quad - \quad \#(p_i, I(t_j)) \quad + \quad \#(p_i, O(t_j))$$

$\#(p_i, I(t_j))$ = multiplicity of the arc from p_i to t_j
$\#(p_i, O(t_j))$ = multiplicity of the arc from t_j to p_i

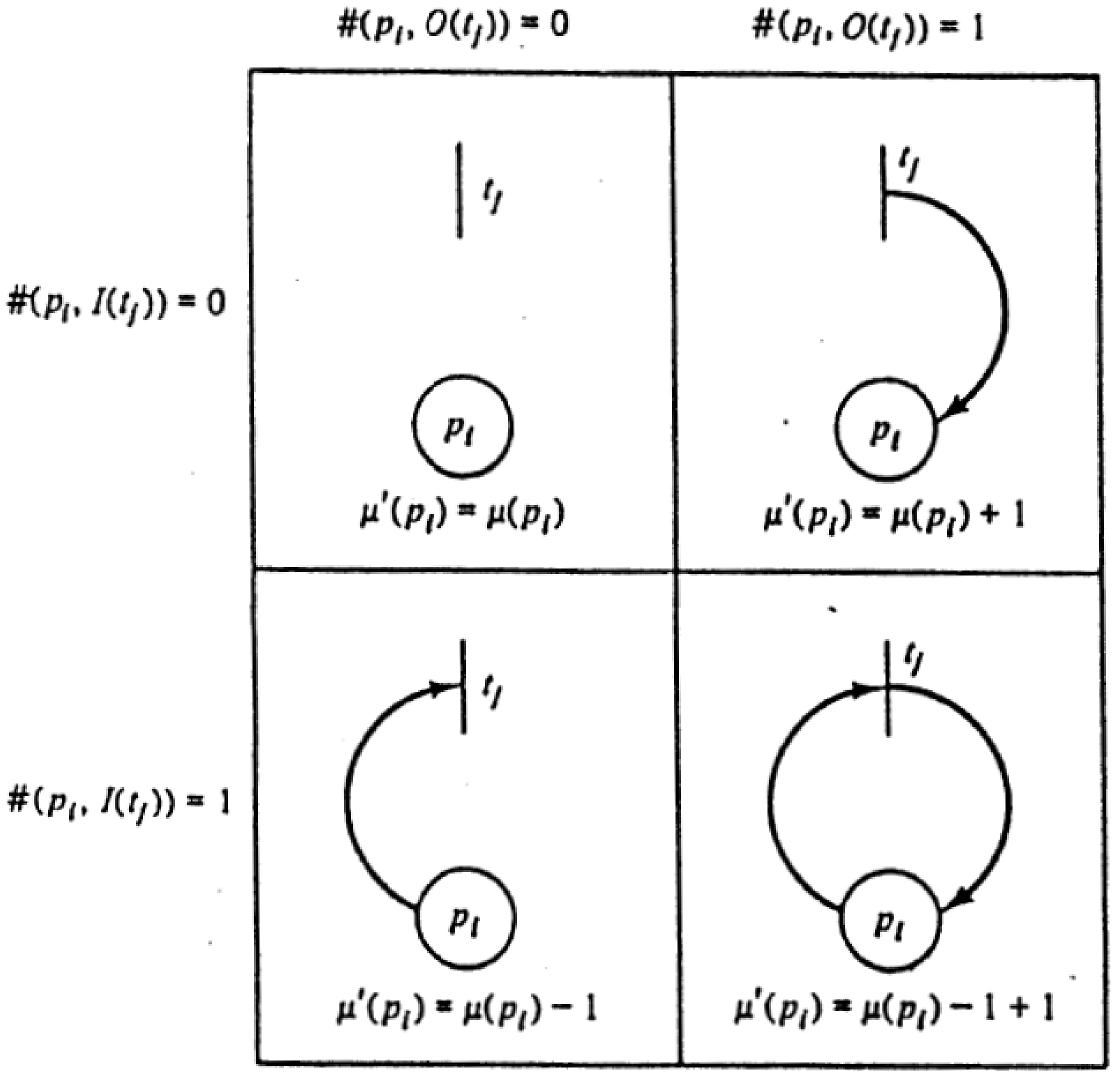
[Peterson81 § 2.3]

Execution Rules for Petri Nets
(Dynamics of Petri nets)



$$n = \#(p_i, I(t_j))$$

$$m = \#(p_i, O(t_j))$$



$$\mu'(p_i) = \mu(p_i) - \#(p_i, I(t_j)) + \#(p_i, O(t_j))$$

[Peterson81 §2.3]

Later this dynamic equation will be generalized using vector notation $\mu_{k+1} = \mu_k + (D^+ - D^-)q_k$

Petri nets

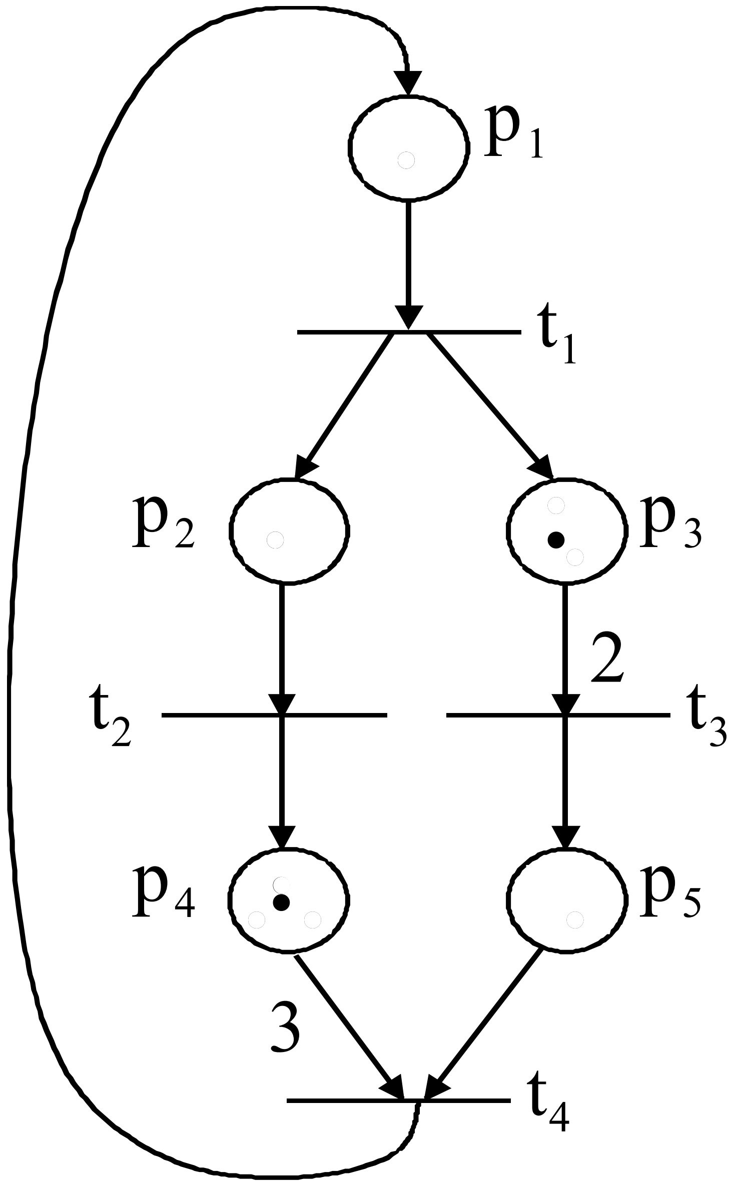
Example of evolution of a Petri net

Initial marking:

$\mu_0 = \{1, 0, 1, 2, 0\}$

This discrete event system can not change state.

It is in a *deadlock!*



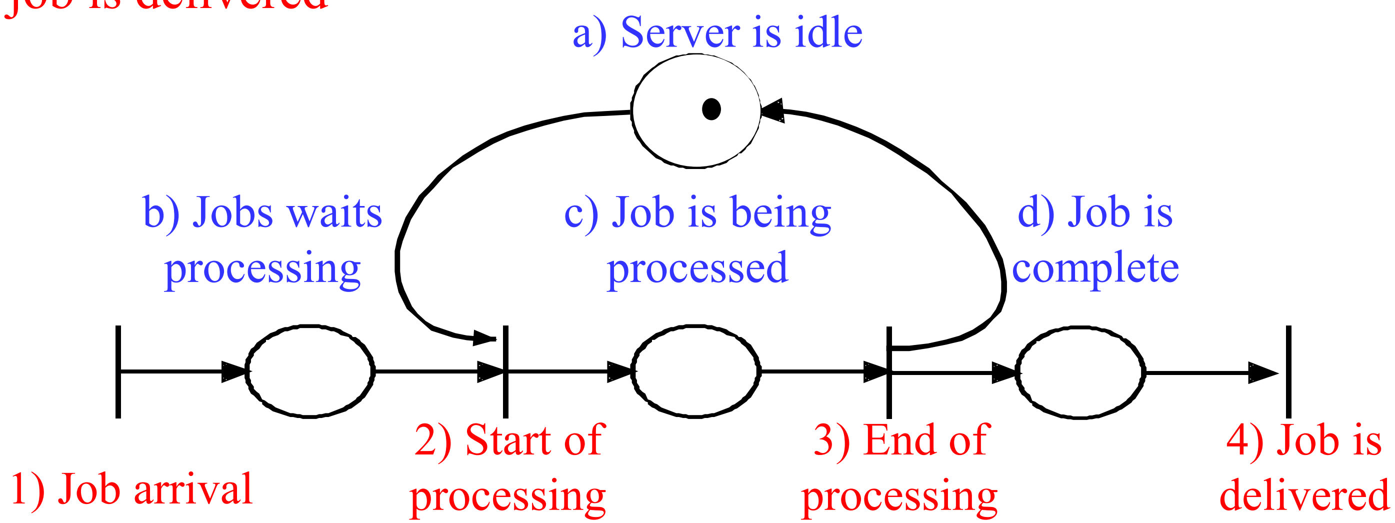
Petri nets: Conditions and Events (Places and Transitions)

Example: Machine waits until an order appears and then machines the ordered part and sends it out for delivery.

Conditions:
a) The server is idle.
b) A job arrives and waits to be processed
c) The server is processing the job
d) The job is complete

Events
1) Job arrival
2) Server starts processing
3) Server finishes processing
4) The job is delivered

Event	Pre-conditions	Pos-conditions
1	-	b
2	a, b	c
3	c	d, a
4	d	-

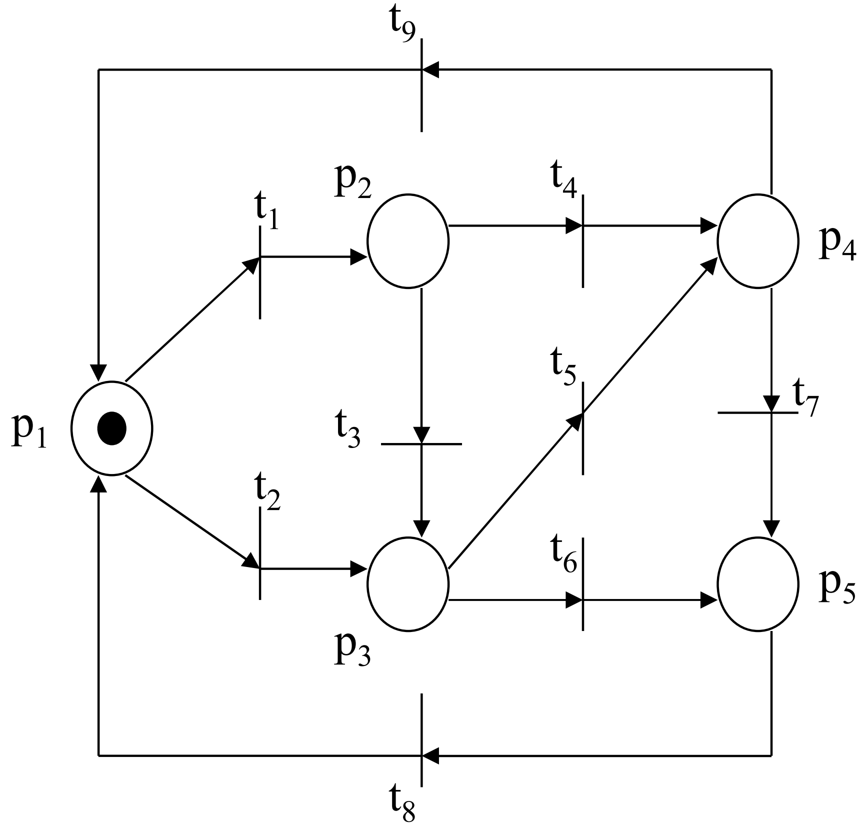


Discrete Event Systems

Example of a simple automation system modeled using PNs

An automatic soda selling machine accepts
50c and $1 coins and
sells 2 types of products:
SODA A, that costs $1.50 and
SODA B, that costs $2.00.

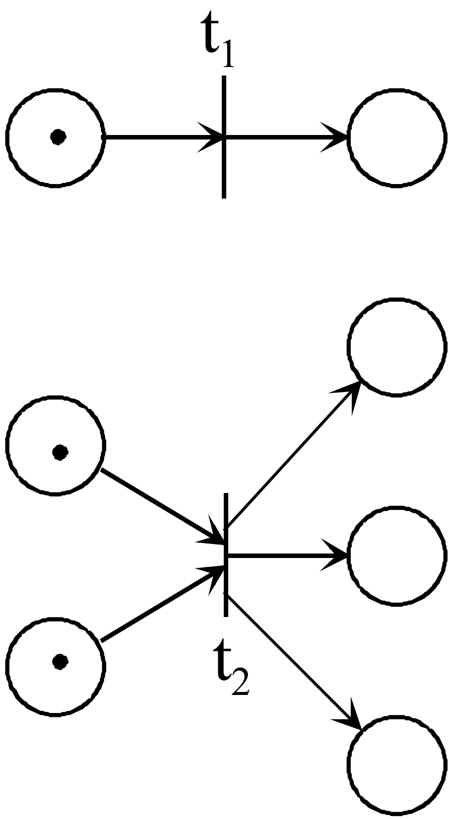
Assume that the money return operation is omitted.



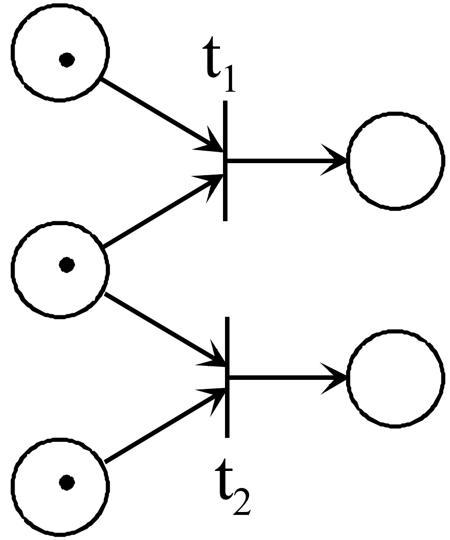
p_1: machine with $0.00;
t_1,t_3,t_5,t_7: coin of 50 c introduced;
t_2,t_4,t_6: coin of $1 introduced;
t_9: SODA A sold, t_8: SODA B sold.

Petri nets: Modeling mechanisms

Concurrence

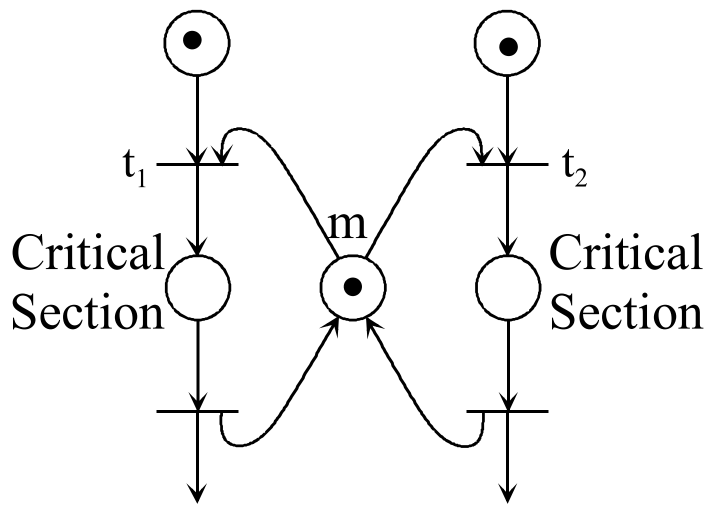


Conflict



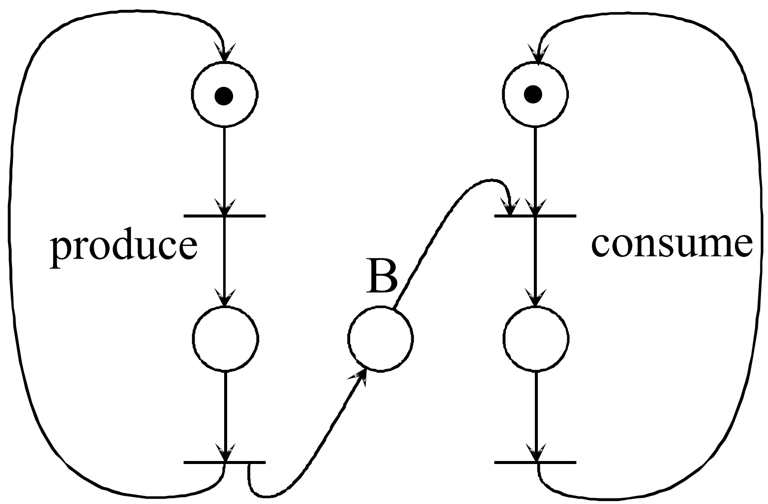
Petri nets: Modeling mechanisms

Mutual Exclusion



Place m represents the permission to enter the critical section

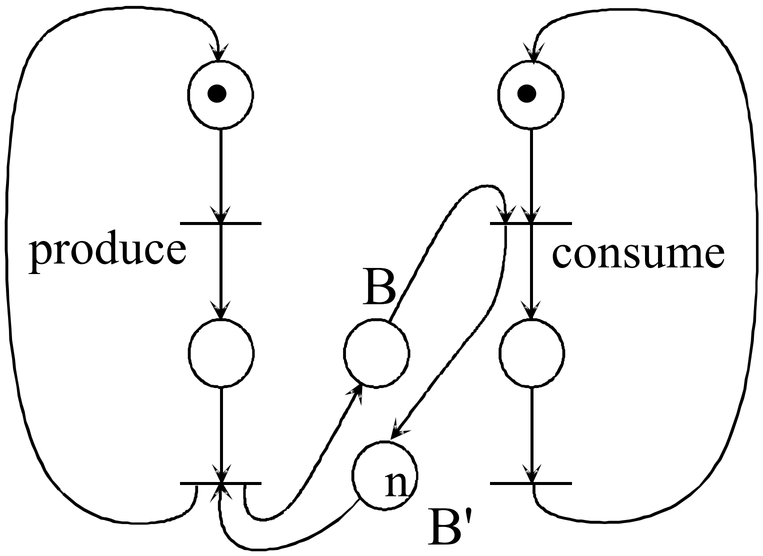
Producer / Consumer



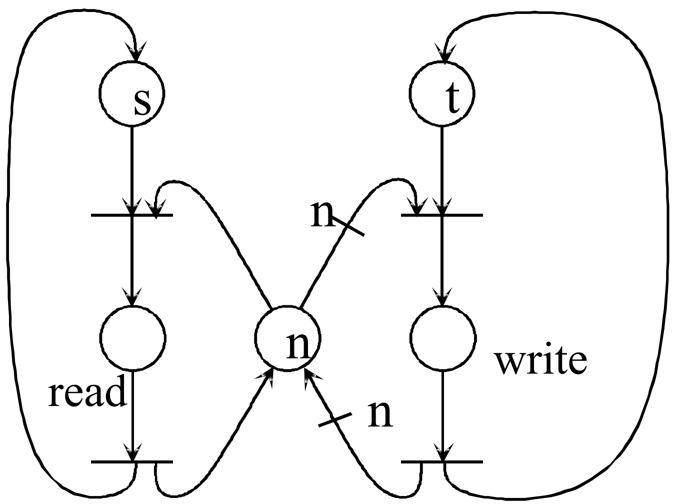
B= buffer holding produced parts

Petri nets: Modeling mechanisms

Producer / Consumer with finite capacity

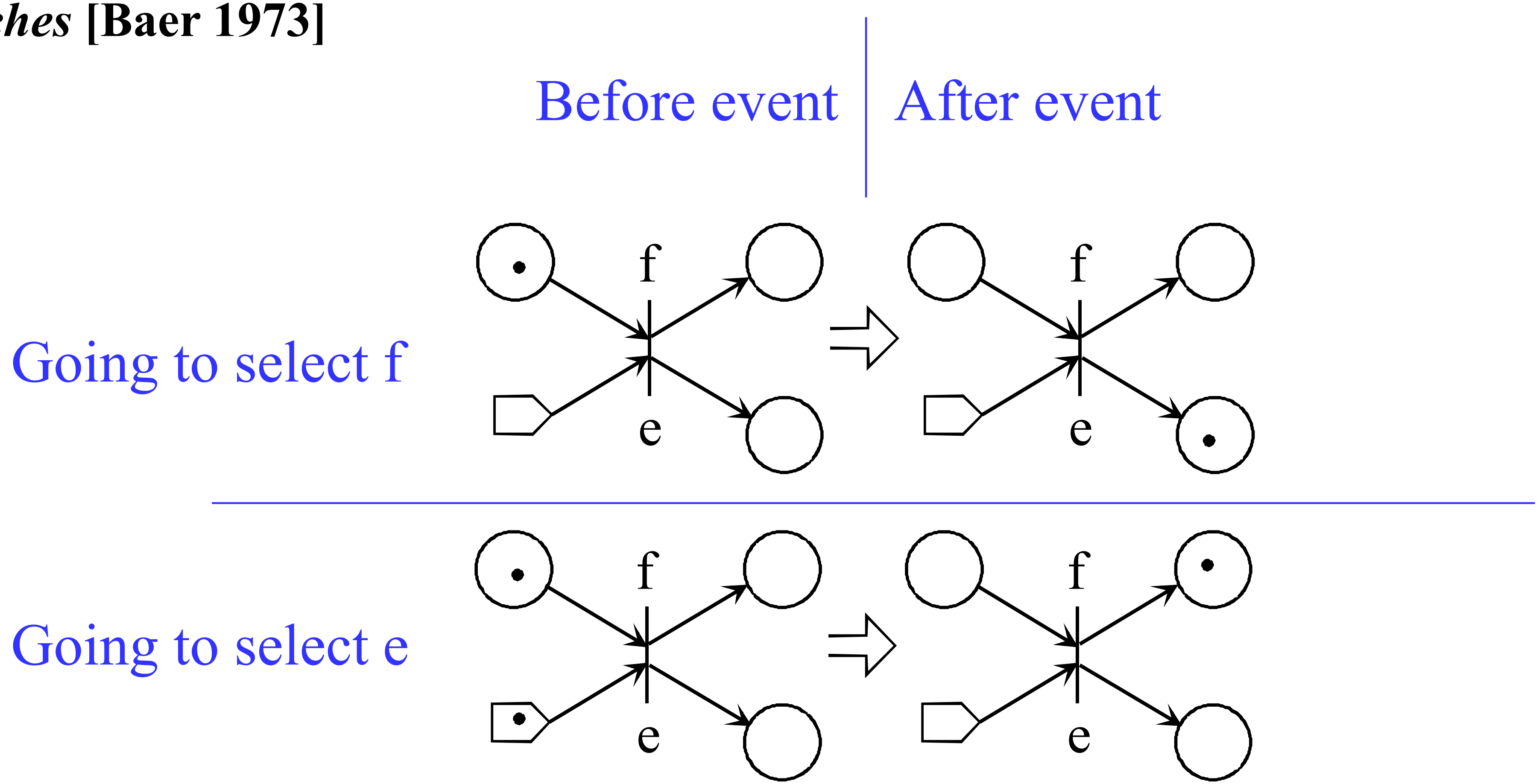


s Readers / t Writers



Extensions to Petri nets

***Switches* [Baer 1973]**



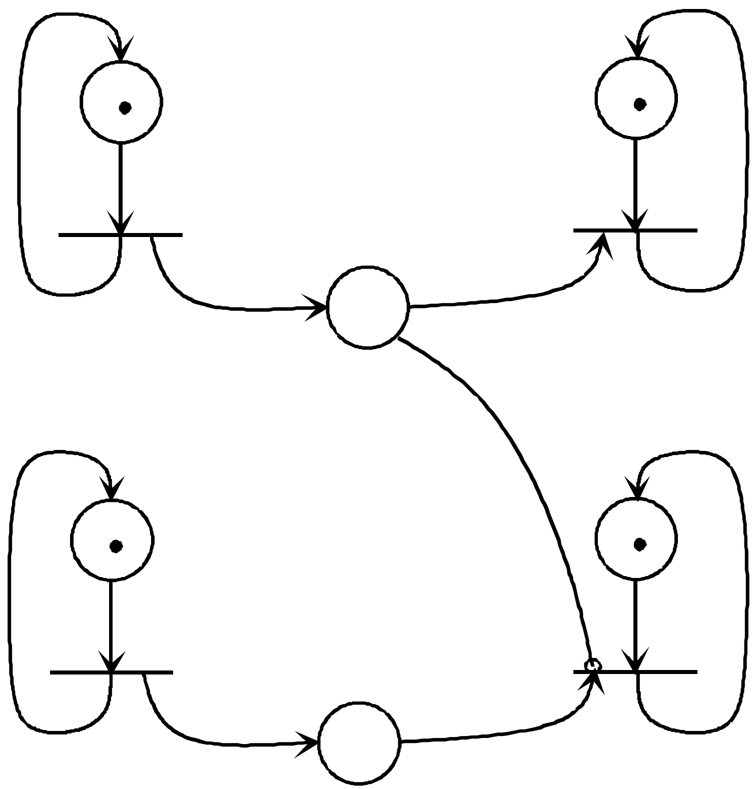
Possible to be implemented with restricted Petri nets.

Extensions to Petri nets

Inhibitor Arcs

Equivalent to

nets with priorities



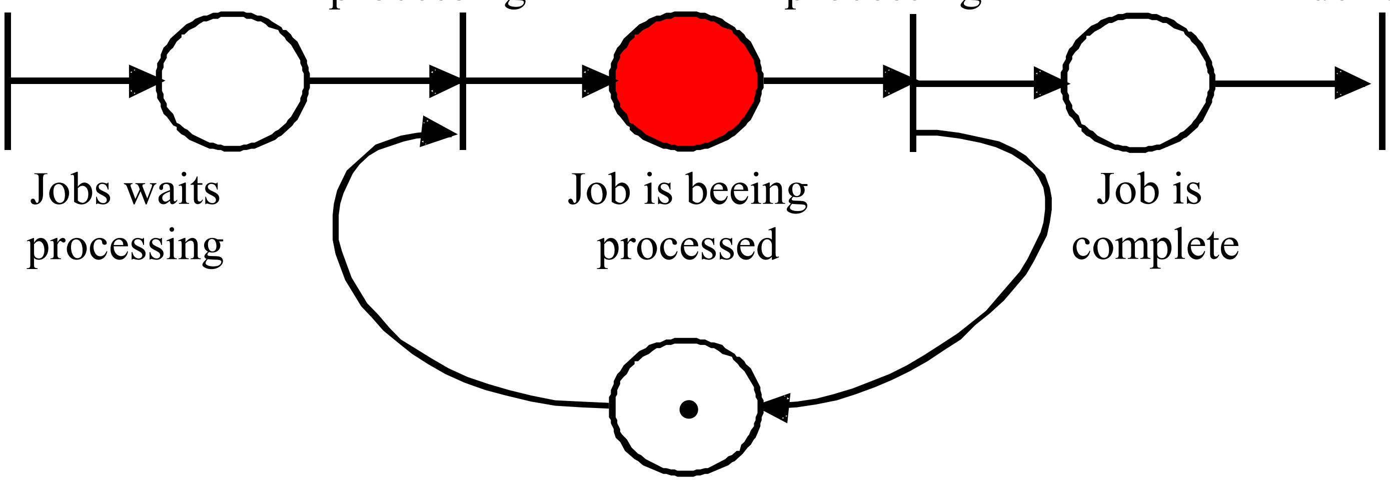
Can be implemented with restricted Petri nets?

Zero tests...

Infinity tests...

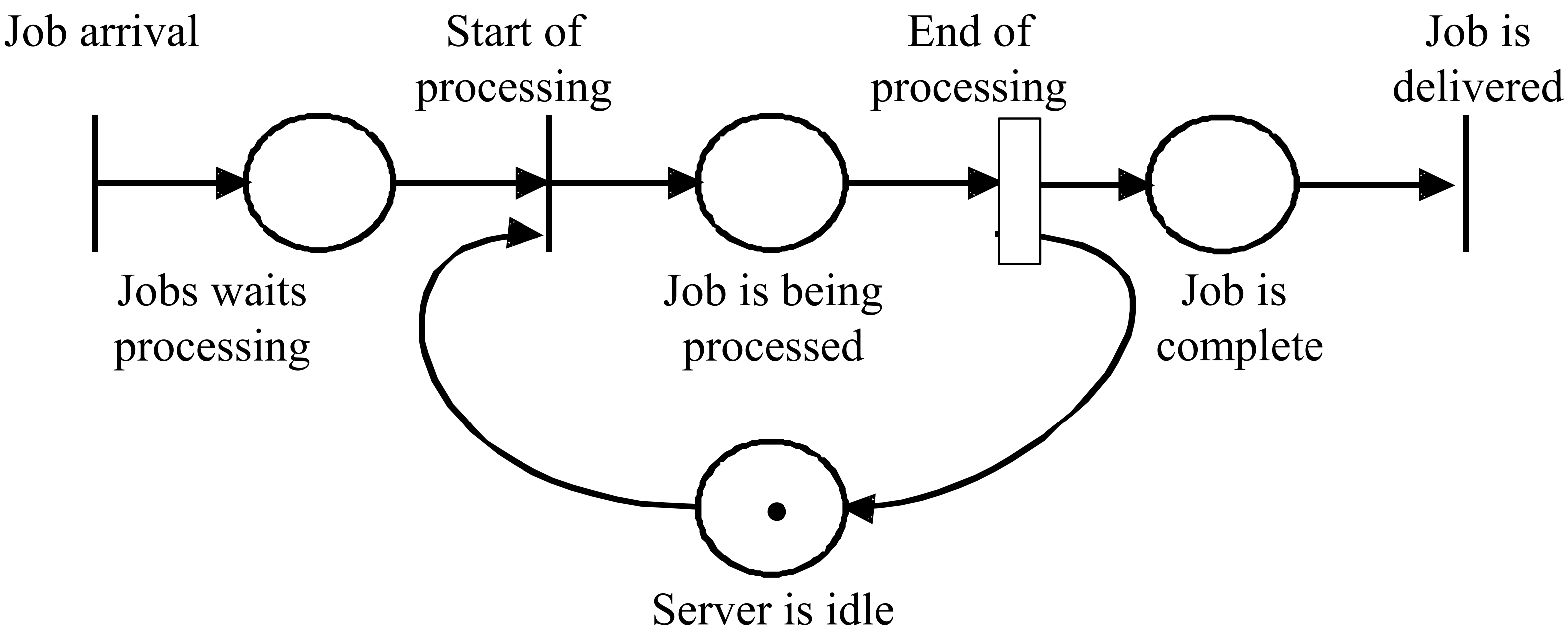
Extensions to Petri nets

P-Timed nets



Extensions to Petri nets

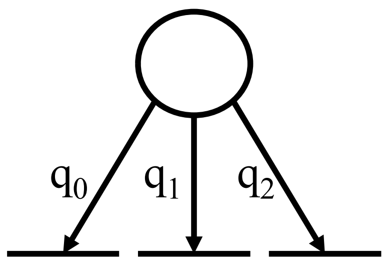
T-Timed nets



Extensions to Petri nets

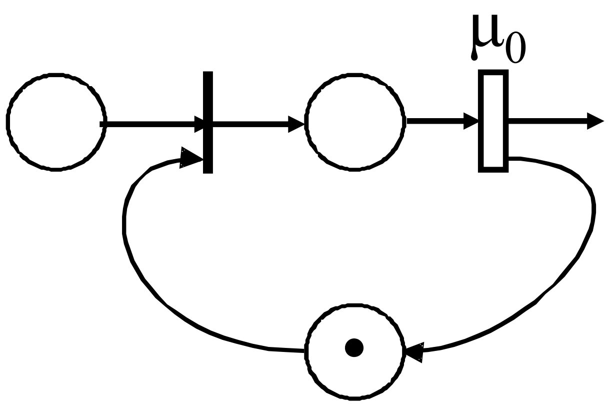
Stochastic nets

Stochastic switches



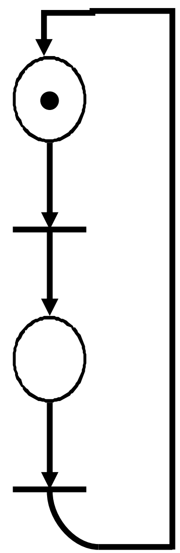
$q_0+q_1+q_2=1$

Transitions with stochastic timings described by a stochastic variable with known pdf



Discrete Event Systems

Sub-classes of Petri nets

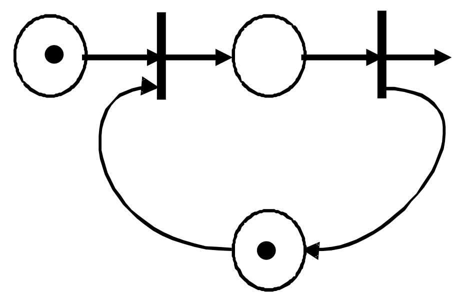


State Machine:

Petri nets where each transition has exactly one input arc and one output arc.

Marked Graphs:

Petri nets where each place has lesser than or equal to one input arc and one output arc.



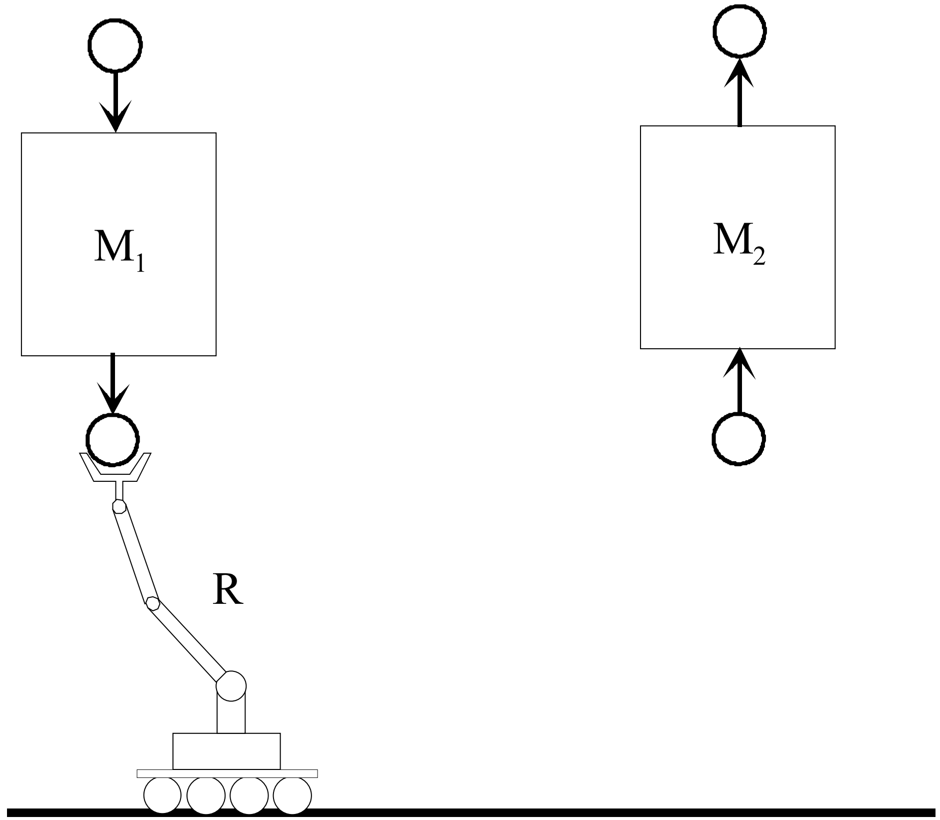
Discrete Event Systems

Example of DES:

Manufacturing system composed by **2 machines** (M_1 and M_2) and a robotic **manipulator** (R). This takes the finished parts from machine M_1 and transports them to M_2.

No buffers available on the machines.
If R arrives near M_1 and the machine is busy, the part is rejected.

If R arrives near M_2 and the machine is busy, the manipulator must wait.

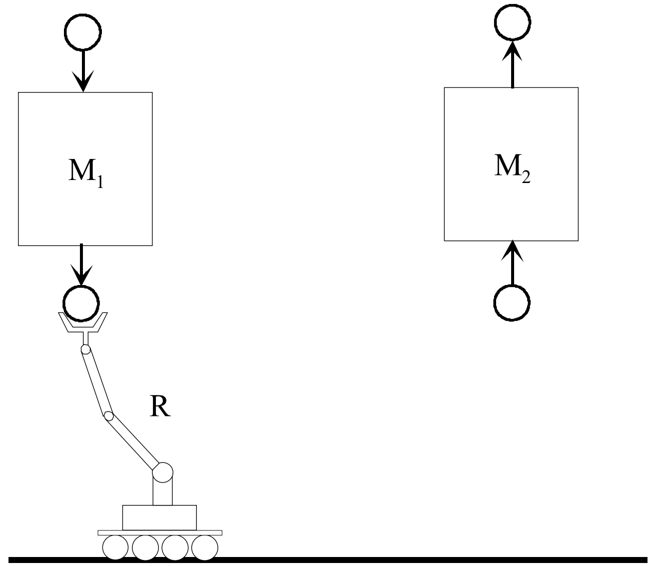


Machining time:

M_1=0.5s $R_{M1 \rightarrow M2}$=0.2s
M_2=1.5s $R_{M2 \rightarrow M1}$=0.1s

Discrete Event Systems

Example of DES:



Define places

M_1 is characterized by places
x_1 = {Idle, Busy, Waiting}

M_2 is characterized by places
x_2 = {Idle, Busy}

R is characterized by places
x_3 = {Idle, Carrying, Returning}

Example of arrival of parts:

$$a(t) = \begin{cases} 1 & in \quad \{0.1, \quad 0.7, \quad 1.1, \quad 1.6, \quad 2.5\} \\ 0 & in \quad other \;\; time \;\; stamps \end{cases}$$

Discrete Event Systems

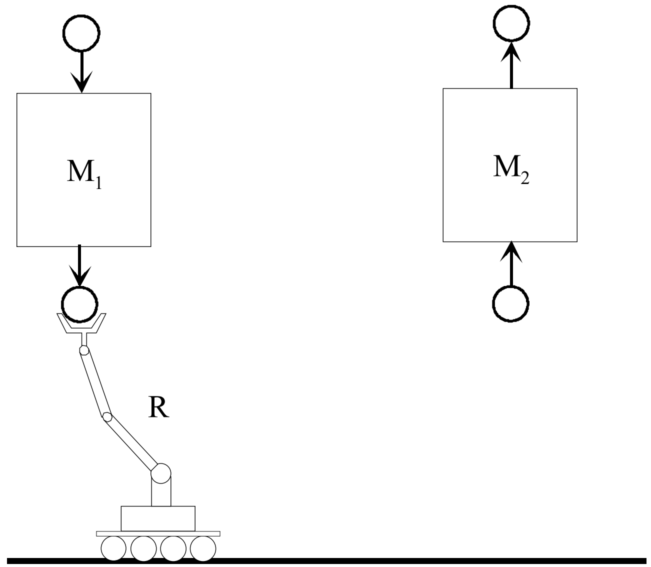
Example of DES:

Definition of events:

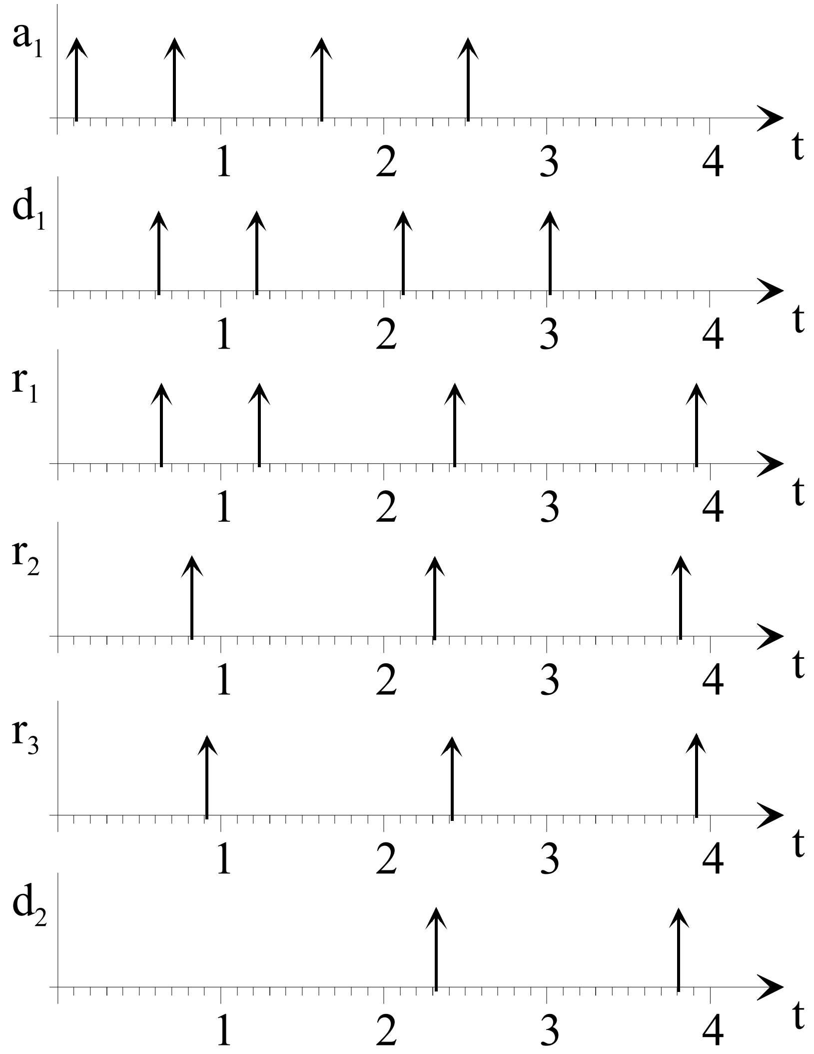
a_1 - loads part in M_1
d_1 - ends part processing in M_1

r_1 - loads manipulator
r_2 - unloads manipulator and loads M_2

d_2 - ends part processing in M_2
r_3 - manipulator at base



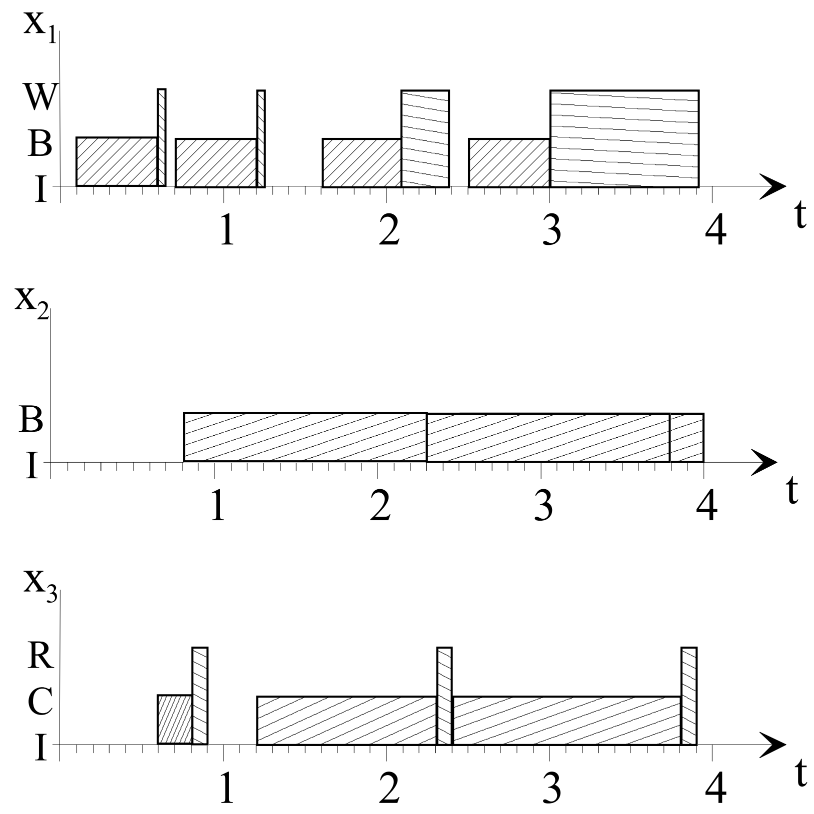
Discrete Event Systems



x_1 = {Idle, Busy, Waiting}

x_2 = {Idle, Busy}

x_3 = {Idle, Carrying, Returning}



Discrete Event Systems

Example of DES:

Events:

a_1 - loads part in M_1
d_1 - ends part processing in M_1
r_1- loads manipulator
r_2- unloads manipulator and loads M_2
d_2- ends part processing in M_2
r_3- manipulator at base