Industrial Automation

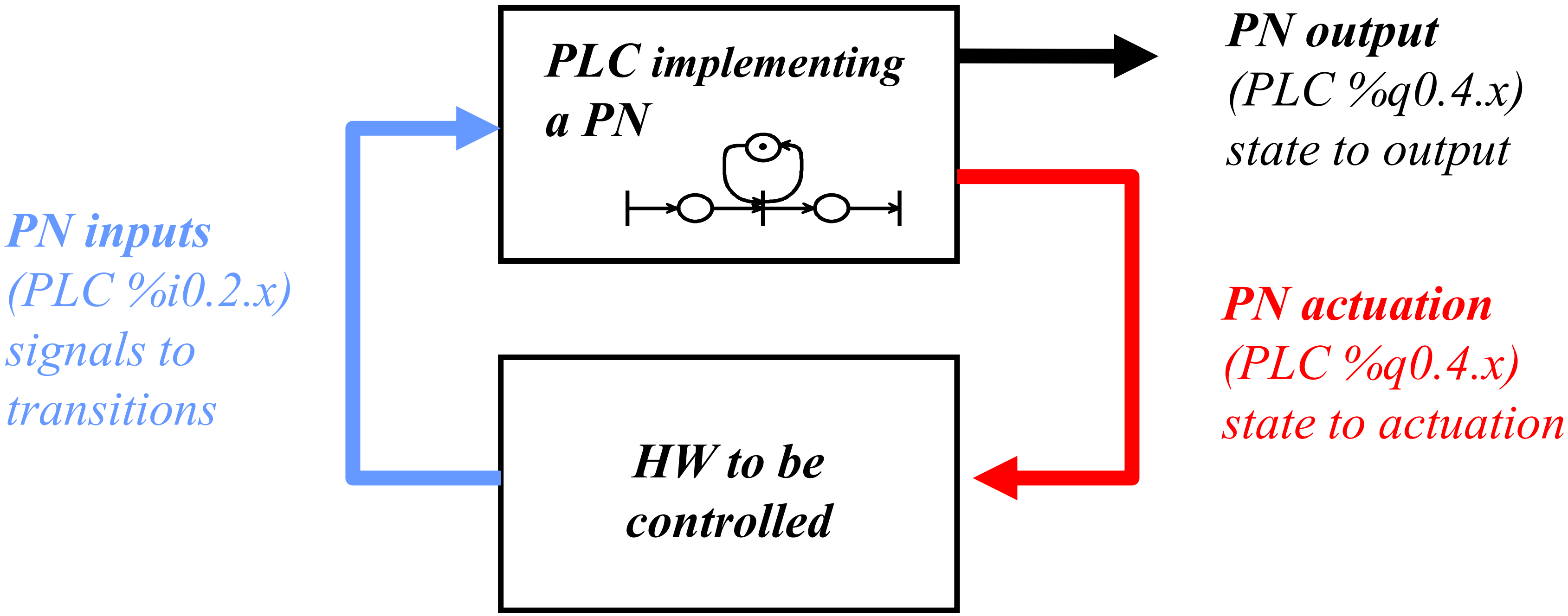
(Automação de Processos Industriais)

Analysis of Discrete Event Systems
Running a Petri net with I/O

http://users.isr.ist.utl.pt/~jag/courses/api19b/api1920.html

Prof. José Gaspar, rev. 2020

Running a Petri net with HW inputs and outputs



Main systems:

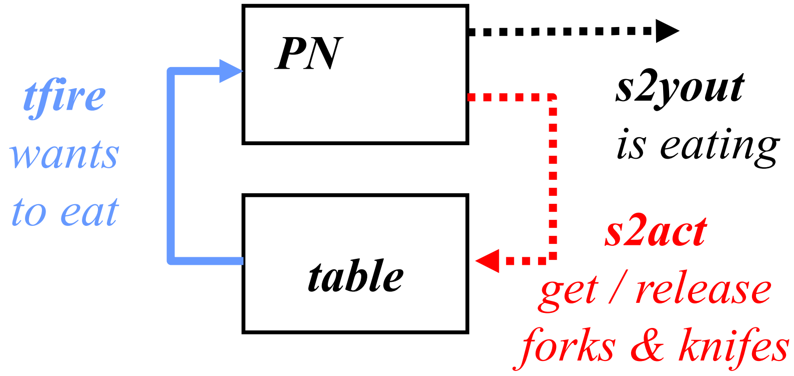
(a) Petri net, run in PC or PLC
(b) Hardware to be controlled

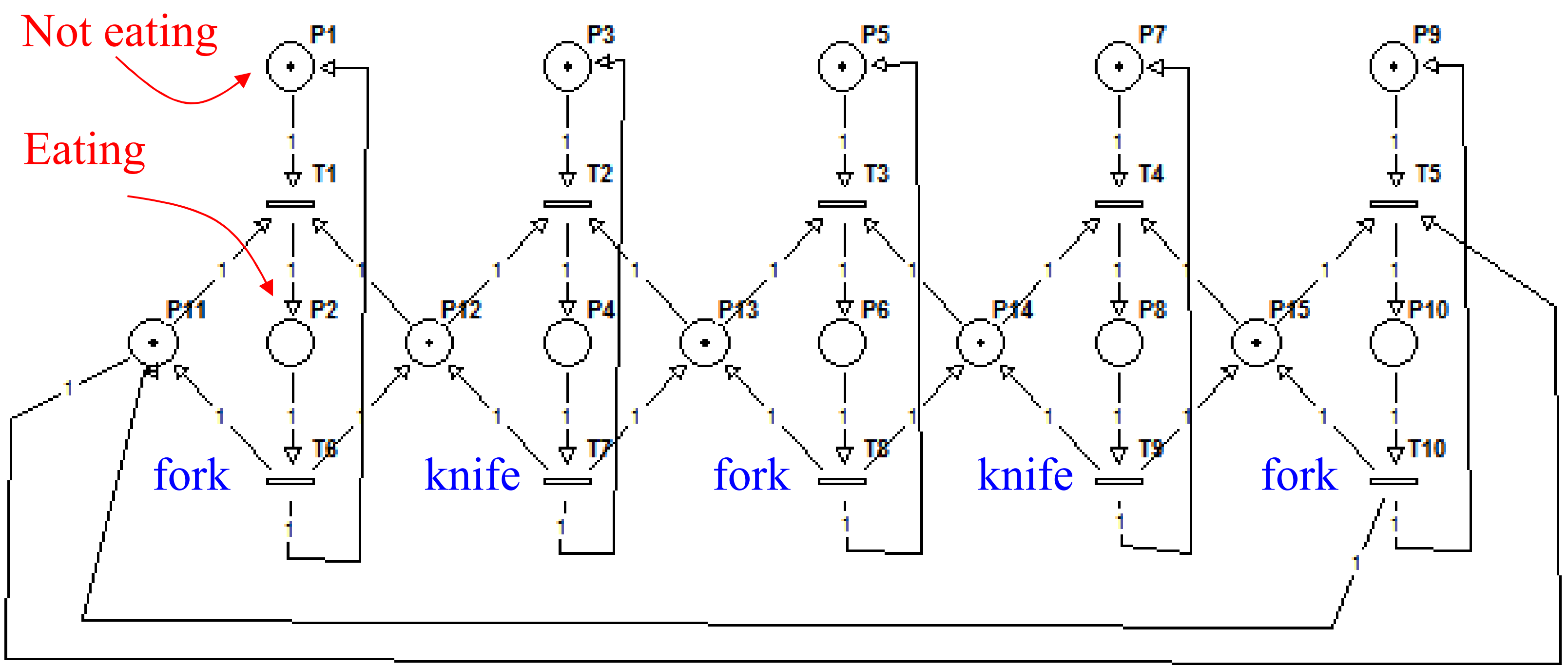
Interface functions, run in PC or PLC:

(1) state/places to actuation,
(2) signals to transitions,
(3) state/places to output

Example 1: Philosophers Dinner

This PN has inputs "Philosopher i wants to eat".

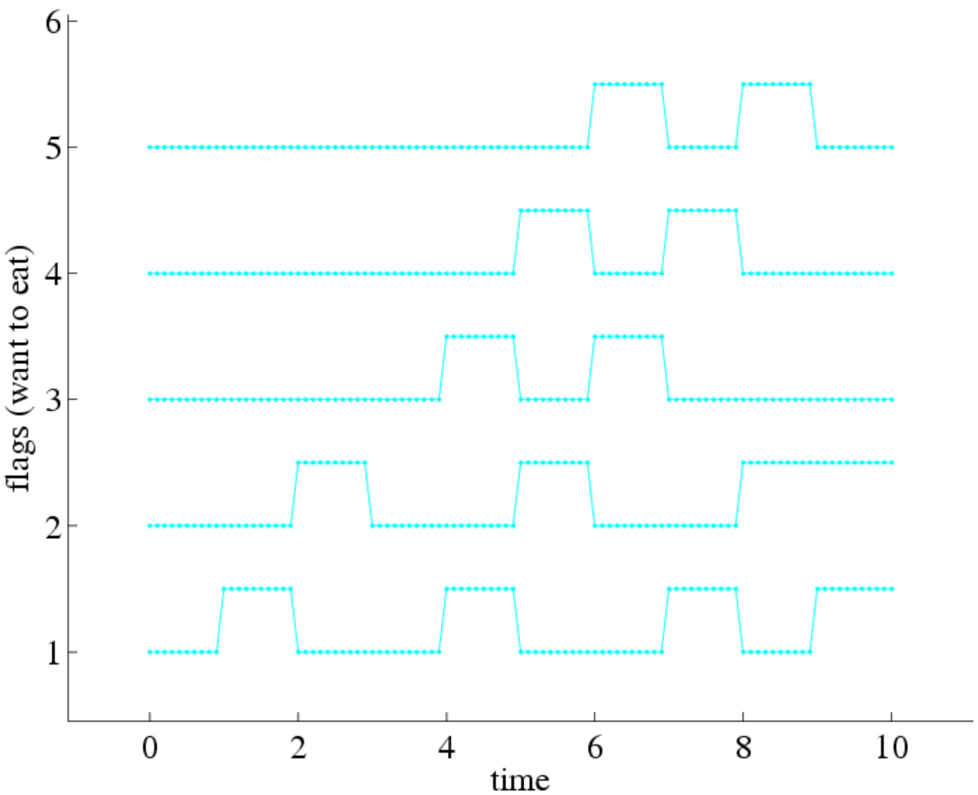




Philosopher1, Philosopher2, Philosopher3, Philosopher4, Philosopher5

Running a Petri net with HW inputs and outputs

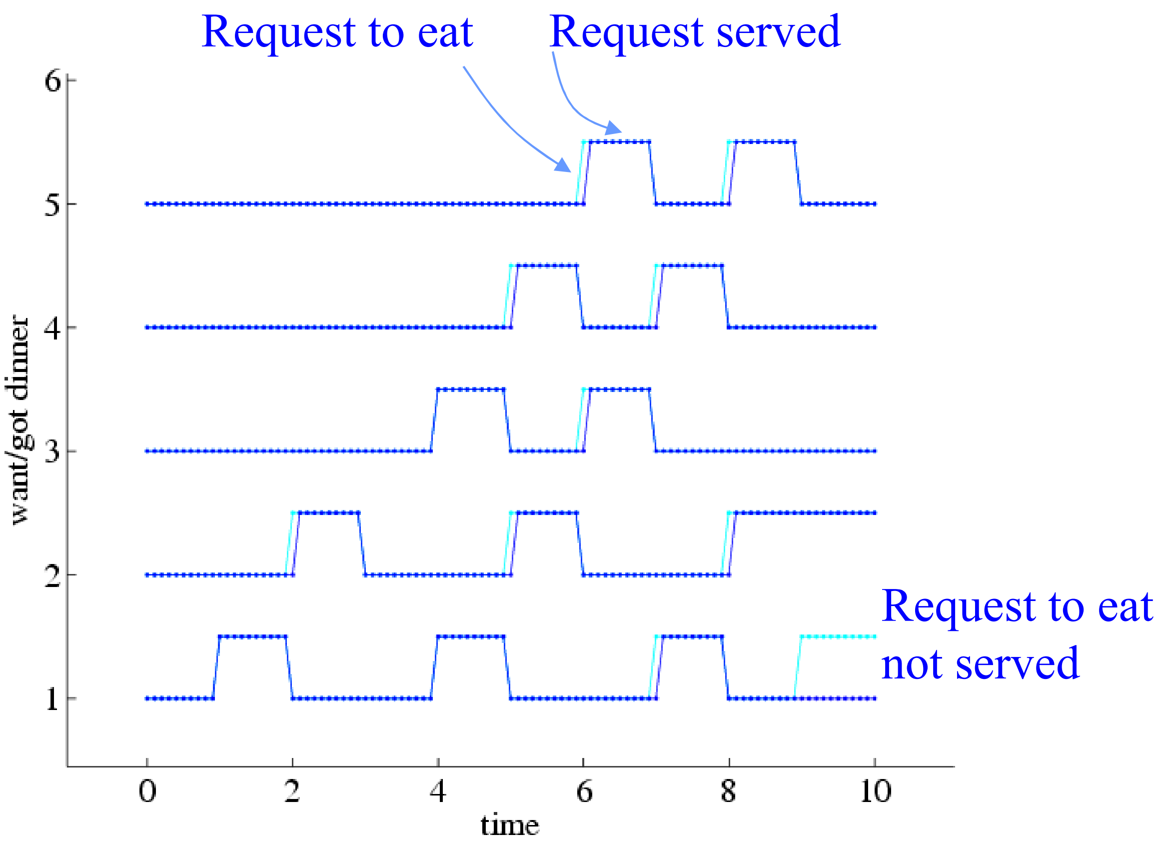
Example: Philosophers Dinner – input / events




```
% first column = time in seconds
% next 5 columns = want to eat flags at time t
%

tu= [...
    0.0  want_to_eat( [] ) ; ...
    1.0  want_to_eat( 1  ) ; ...
    2.0  want_to_eat( 2  ) ; ...
    3.0  want_to_eat( []  ) ; ...
    4.0  want_to_eat( [1 3] ) ; ...
    5.0  want_to_eat( [2 4] ) ; ...
    6.0  want_to_eat( [3 5] ) ; ...
    7.0  want_to_eat( [4 1] ) ; ...
    8.0  want_to_eat( [5 2] ) ; ...
    9.0  want_to_eat( [1 2] ) ; ...
    ];


function y= want_to_eat(kid)
y= zeros(1,5);
for i=1:length(kid)
    y(kid(i))= 1;
end
```

Running a Petri net with HW inputs and outputs

Example: Philosophers Dinner – simulation



Note: See complete demo in the webpage of the course.

Note2: Modern operating systems must work better than failing early like in this PN simulation. E.g. two programs requiring simultaneously much CPU and memory, the O.S. has managers that hold the resources (CPU, memory, etc), queue the requests and in most cases even preempt the resources (CPU).

Running a Petri net with HW inputs and outputs

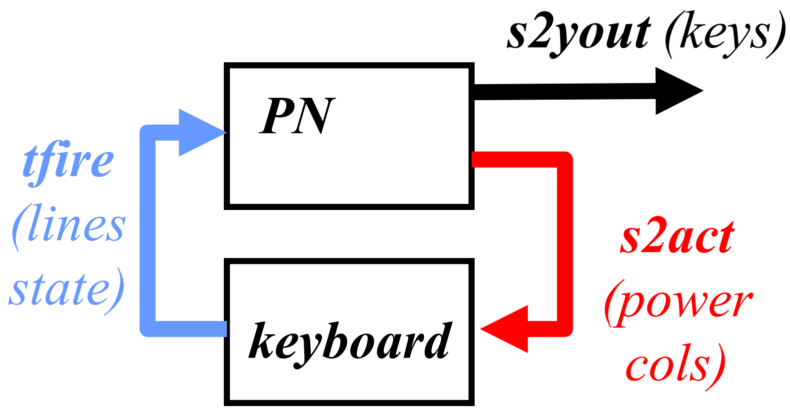
Running a generic Petri net

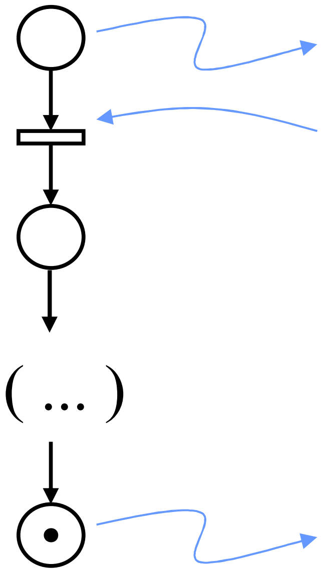
```
function [tSav, MPSav, youtSav]= PN_sim(Pre, Post, M0, ti_tf)
%
% Simulating a Petri net, using a SFC/Grafcet simulation methodology.
% See book "Automating Manufacturing Systems", by Hugh Jack, 2008
% (ch20. Sequential Function Charts)
%
% Petri net model:
%  M(k+1) = M(k) +(Post-Pre)*q(k)
%  Pre and Post are NxM matrices, meaning N places and M transitions

% 0. Start PN at state M0
%
MP=M0;
ti=ti_tf(1); tf=ti_tf(2); tSav= (ti:5e-3:tf)';
MPSav= zeros( length(tSav), length(MP) );
youtSav= zeros( length(tSav), length(PN_s2yout(MP)) );

for i= 1:length(tSav)

    % 1. Check transitions (update state)
    tm= tSav(i);
    qk= PN_tfire(MP, tm);
    qk2= filter_possible_firings(MP, Pre, qk(:));
    MP= MP +(Post-Pre)*qk2;

    % 2. Do place activities
    yout= PN_s2yout(MP);

    % Log all results
    MPSav(i,:)= MP';
    qkSav(i,:)= qk2';
    youtSav(i,:)= yout;

end
```

```
function qk2= filter_possible_firings(M0, Pre, qk)
% verify Pre*q <= M
% try to fire all qk entries

M= M0;
mask= zeros(size(qk));
for i=1:length(qk)
    % try accepting qk(i)
    mask(i)= 1;
    if any(Pre*(mask.*qk) > M)
        % exceeds available markings
        mask(i)= 0;
    end
end
qk2= mask.*qk;
```

Example 2: Keyboard Reading

output = columns power
input = lines read





1. state to actuation:
   power kb columns
2. signals to transitions:
   wait signal on kb lines
3. state to output:
   key X is pressed

Code template (Matlab):

Main systems
a) PN_sim.m
b) PN_device_kb_IO.m

Interface functions
1) PN_s2act.m
2) PN_tfire.m
3) PN_s2yout.m

Running a Petri net with HW inputs and outputs

```
function lines= PN_device_kb_IO(act, t)

% Define 4x3-keyboard output line-values given actuation on the 3 columns
% and an (internal) time table of keys pressed
% Input:
%  act: 1x3 : column actuation values
%  t   : 1x1 : time
% Output:
%  lines: 1x4 : line outputs

global keys_pressed
if isempty(keys_pressed)
    % first column = time in seconds
    % next 12 columns = keys pressed at time t
    keys_pressed= [...
        0  mk_keys([]) ; 1  mk_keys(1)   ; ...
        2  mk_keys([]) ; 3  mk_keys(5)   ; ...
        4  mk_keys([]) ; 5  mk_keys(9)   ; ...
        6  mk_keys([]) ; 7  mk_keys([1 12]) ; ...
        8  mk_keys(12) ; 9  mk_keys([]) ; ...
        ];
end

% pressed keys yes/no
ind= find(t>=keys_pressed(:,1));
if isempty(ind)
    lines= [0 0 0 0]; % default lines output for t < 0
    return
end
keys_t= keys_pressed(ind(end), :);

% if actuated column and key pressed match, than activate line
lines= sum( repmat(act>0, 4,1) & reshape(keys_t(2:end), 3,4)', 2);
lines= (lines > 0)';
```

Keyboard simulator: generate line values given column values

```
function y= mk_keys(kid)
y= zeros(1,12);
for i=1:length(kid)
    y(kid(i))= 1;
end
```

Running a Petri net with HW inputs and outputs

Prototypes of the interfacing functions

The implementation of these functions is to be done by each group in the laboratory.

```
function act= PN_s2act(MP)

% Create 4x3-keyboard column actuation
%
% MP:  1xN : marked places (integer values >= 0)
% act: 1x3 : column actuation values (0 or 1 per entry)
```

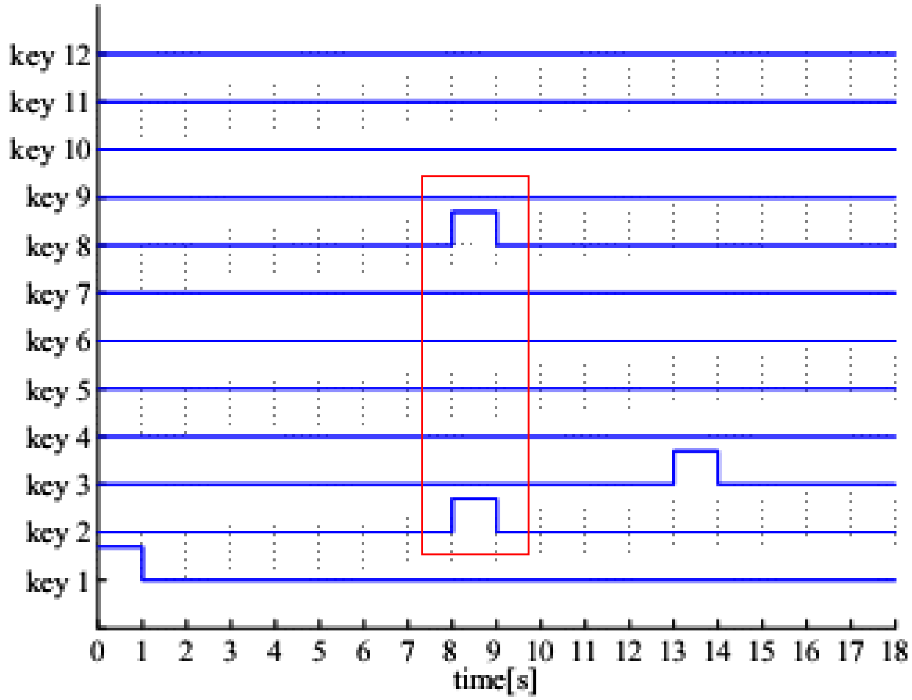
```
function qk= PN_tfire(MP, t)

% Possible-to-fire transitions given PN state (MP) and the time t
%
% MP: 1xN : marked places (integer values >= 0)
% t : 1x1 : time
% qk: 1xM : possible firing vector (to be filtered later with enabled
%             transitions)
```

```
function yout= PN_s2yout(MP)

% Show the detected/undetected key(s) given the Petri state
%
% MP: 1xN : marked places (integer values >= 0)
```

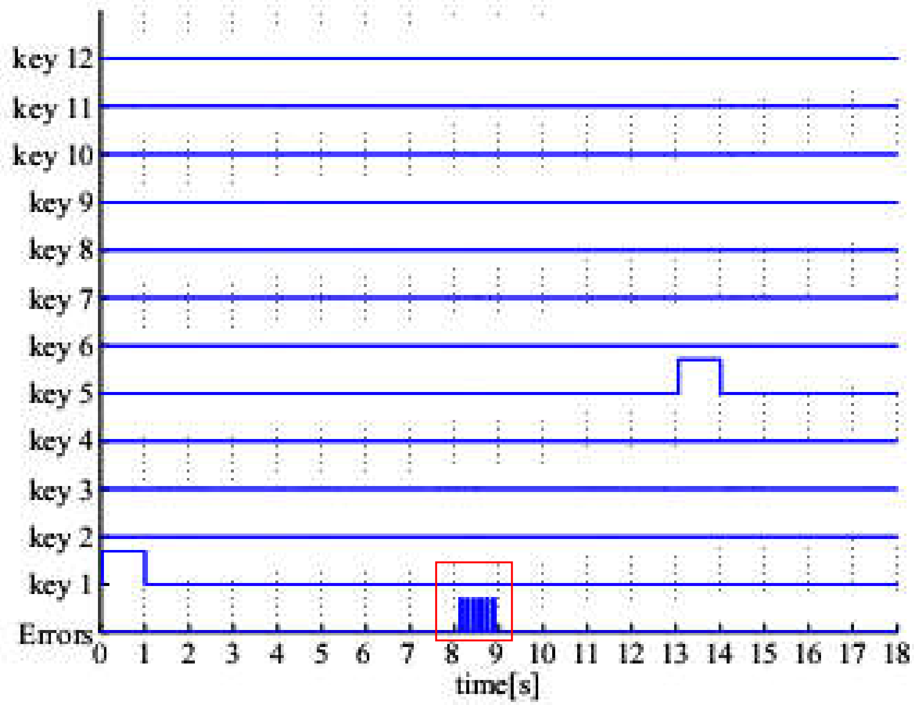
Running a Petri net with HW inputs and outputs

Laboratory assignment: detect keys pressed by the user and just accept those keys when there are not multiple keys pressed at the same time.

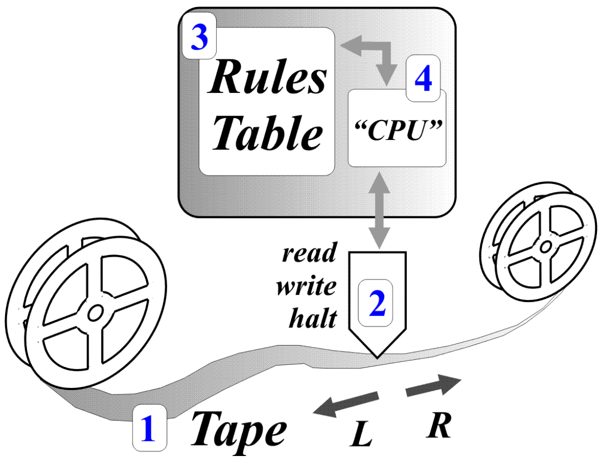
Keys pressed

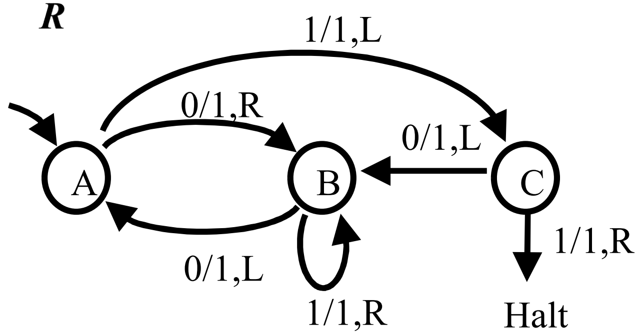


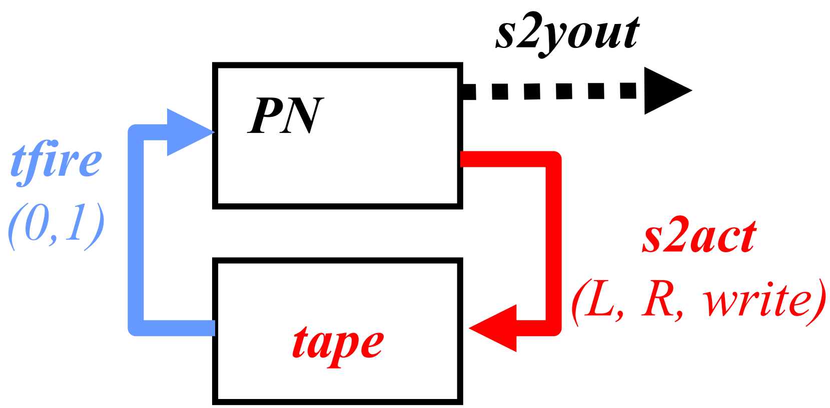
Keys accepted



Example 3: Busy Beaver FSM as PN







outputs = tape left, right, write

input = tape read (one bit, i.e. 0 or 1)

Code template (Matlab):

Main systems

a) PN_sim.m (as before)

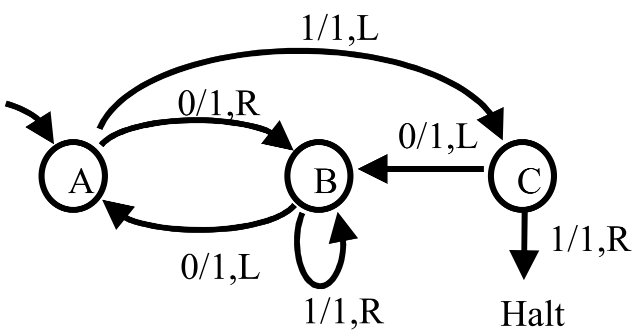
b) TM_tape.m (see Turing)

Interface functions

1) PN_s2act.m
2) PN_tfire.m
3) PN_s2yout.m

Turing machine, Busy-Beaver 3states 2symbols,
graph vs **table**, see input / output :

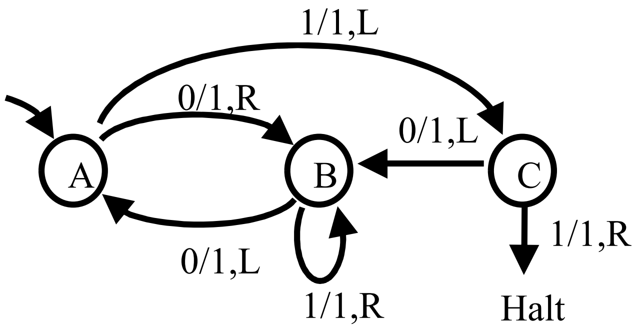
outputs = tape left, right, write (1)
input = tape read (one bit, i.e. 0 or 1)



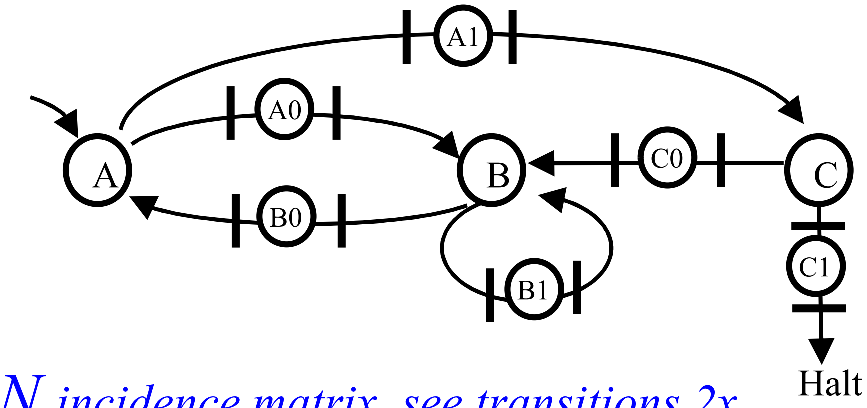
Current state	Input	Action R/W	Action L/R/N	Next state
A	0	write1	right	B
A	1	write1	left	C
B	0	write1	left	A
B	1	write1	right	B
C	0	write1	left	B
C	1	write1	null	halt

Busy-Beaver is a FSM with outputs in the arcs (not the "places"), hence it is a Mealy machine (not a Moore machine). How to represent as a Petri net just with outputs in its places?

FSM



PN *each arc of the FSM completed with 1 place and 2 transitions*



1. FSM *table*

	0	1
A	B	C
B	A	B
C	B	H
H	-	-

2. PN *incidence matrix, see transitions 2x number of places A, B, C*

	0A	1A	0B	1B	0C	1C
A	-1	-1	+1			
B	+1		-1	-1+1	+1	
C		+1			-1	-1
H						+1

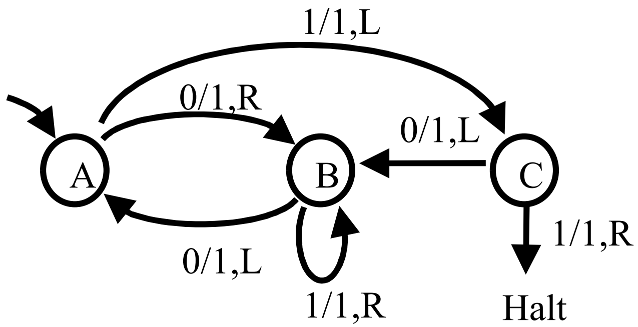
3. FSM *table + outputs*

	0	1
A	B/1,R	C/1,L
B	A/1,L	B/1,R
C	B/1,L	H/1,R
H	-	-

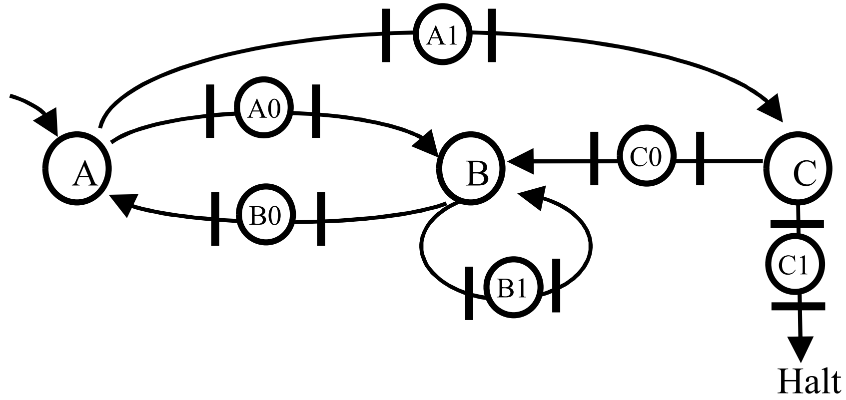
4. PN *incidence matrix with outputs at the arcs*

	Output	0A	0Ax	1A	1Ax	0B	0Bx	1B	1Bx	0C	0Cx	1C	1Cx
A	-	-1		-1			+1						
A0	1,R	+1	-1	+1	-1								
A1	1,L												
B	-		+1			-1		-1	+1		+1		
B0	1,L					+1	-1						
B1	1,R							+1	-1				
C	-				+1					-1		-1	
C0	1,L									+1	-1		
C1	1,N											+1	-1
H	-												+1

Turing-Machine Busy-Beaver: PN shown in previous slide, here implement **Input / Output**



1	A	-
2	A0	1,R
3	A1	1,L
4	B	-
5	B0	1,L
6	B1	1,R
7	C	-
8	C0	1,L
9	C1	1,N
10	H	-

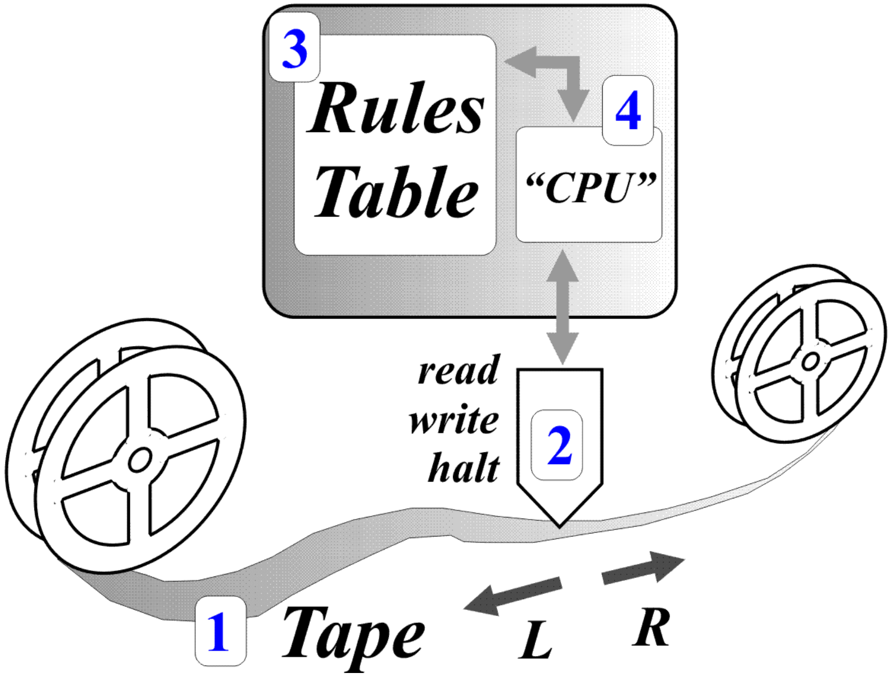



```
% TM_tape.m : left, right, read, write
% at A,B,C, read bit & activate transitions
% at A0,B1 move right
% at A1,B0,C0 move left
```

```
function act= PN_s2act( MP )

act= TM_tape('read');

if max(MP([2 3  5 6  8 9]))>0
    TM_tape('write',1);
end

if MP(3)>0 || MP(5)>0 || MP(8)>0
    TM_tape('left');
elseif MP(2)>0 || MP(6)>0
    TM_tape('right')
else
    % do nothing
end
```

```
function qk= PN_tfire( act, t )

qk= ones(1,12);

if act  % read 1 from tape
    qk([1 5 9])= 0;
    qk([3 7 11])= 1;
else
    qk([1 5 9])= 1;
    qk([3 7 11])= 0;
end
```

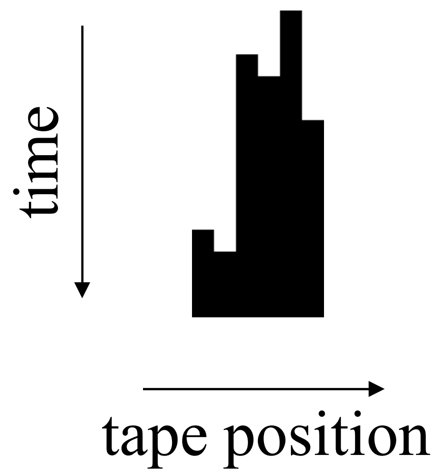
Turing Machine Busy Beaver: simulation results



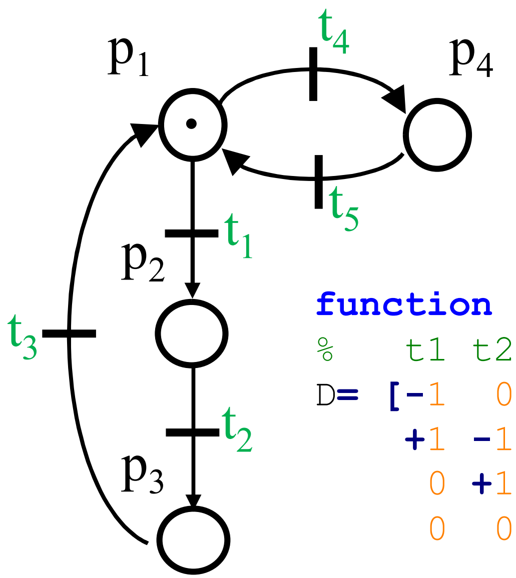
3-state Busy Beaver:

a0 -> b1r a1 -> h1r
b0 -> c0r b1 -> b1r
c0 -> c1l c1 -> a1l

halts after **21 time steps**
fills **6 ones**



Example 4: PN to PLC




```
function PN= define_petri_net
%    t1 t2 t3 t4 t5
D= [-1  0 +1 -1 +1
    +1 -1  0  0  0
     0 +1 -1  0  0
     0  0  0 +1 -1];

Pre = -D.*(D<0);
Post=  D.*(D>0);

M0  = [1 0 0 0]';

% Petri net structure:
% 0.5 sec from p1..p3 to trans t1..t3
% col2=place, col3=trans

T = 0.5;
tt= [T 1 1; T 2 2; T 3 3];
PN= struct('pre',Pre, 'pos',Post, 'mu0',M0,
           'ttimed',tt);
```

```
function tst1_blink

PN          = define_petri_net;
input_map   = define_input_mapping;
output_map  = define_output_mapping;
ofname      = 'tst1_blink.txt';
plc_make_program( ofname, PN,
              input_map, output_map )
```

```
function inp_map= define_input_mapping
% input0 fires transition4
% negative input0 fires t5
inp_map= { ...
    0,        4 ;
    -(0+100), 5 ;
    };
```

```
function output_map= define_output_mapping
% map PN places 1..3 to the first output
bits
zCode= plc_z_code_helper('config_get');
output_map= { ...
    1, zCode.outpMin ; ...
    2, zCode.outpMin+1 ; ...
    3, zCode.outpMin+2 ;
    };
```

See complete code in http://users.isr.ist.utl.pt/~jag/course_utils/pn_to_plc/pn_to_plc.html

Example 4: PN to PLC

```
(* --- PNC: Petri net initialization --- *)

IF %MW100=0 THEN
 %MW201:=1; %MW202:=0; %MW203:=0; %MW204:=0;
 %MW100:=1;
END_IF;

(* --- PNC: Map inputs --- *)

%MW104 := BOOL_TO_INT( %i0.2.0 );
%MW105 := BOOL_TO_INT( NOT(%i0.2.0) );

(* --- PNC: Timed transitions --- *)

MY_TON_1(IN := INT_TO_BOOL(%MW201) (*BOOL*),
          PT := t#500ms (*TIME*),
          Q => timer_output_flag (*BOOL*),
          ET => my_time_1 (*TIME*));
%MW101:= BOOL_TO_INT(timer_output_flag);
MY_TON_2(IN := INT_TO_BOOL(%MW202) (*BOOL*),
          PT := t#500ms (*TIME*),
          Q => timer_output_flag (*BOOL*),
          ET => my_time_2 (*TIME*));
%MW102:= BOOL_TO_INT(timer_output_flag);
MY_TON_3(IN := INT_TO_BOOL(%MW203) (*BOOL*),
          PT := t#500ms (*TIME*),
          Q => timer_output_flag (*BOOL*),
          ET => my_time_3 (*TIME*));
%MW103:= BOOL_TO_INT(timer_output_flag);
```

```
(* --- PNC: Petri net loop code --- *)

IF %MW101>0 AND %MW201>=1
THEN
 %MW201:=%MW201-1;
 %MW202:=%MW202+1;
END_IF;

IF %MW102>0 AND %MW202>=1
THEN
 %MW202:=%MW202-1;
 %MW203:=%MW203+1;
END_IF;

IF %MW103>0 AND %MW203>=1
THEN
 %MW203:=%MW203-1;
 %MW201:=%MW201+1;
END_IF;

IF %MW104>0 AND %MW201>=1
THEN
 %MW201:=%MW201-1;
 %MW204:=%MW204+1;
END_IF;

IF %MW105>0 AND %MW204>=1
THEN
 %MW204:=%MW204-1;
 %MW201:=%MW201+1;
END_IF;
```

```
(* --- PNC: Output bits --- *)

IF INT_TO_BOOL(%MW201)
THEN SET(%q0.4.0);
ELSE RESET(%q0.4.0);
END_IF;
IF INT_TO_BOOL(%MW202)
THEN SET(%q0.4.1);
ELSE RESET(%q0.4.1);
END_IF;
IF INT_TO_BOOL(%MW203)
THEN SET(%q0.4.2);
ELSE RESET(%q0.4.2);
END_IF;
```

See complete code in http://users.isr.ist.utl.pt/~jag/course_utils/pn_to_plc/pn_to_plc.html