

Industrial Automation

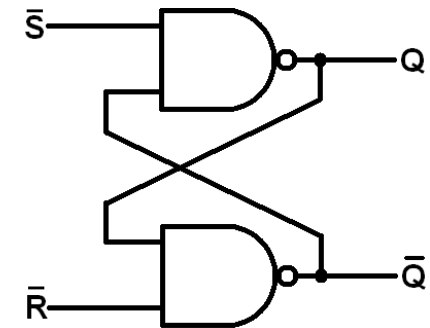
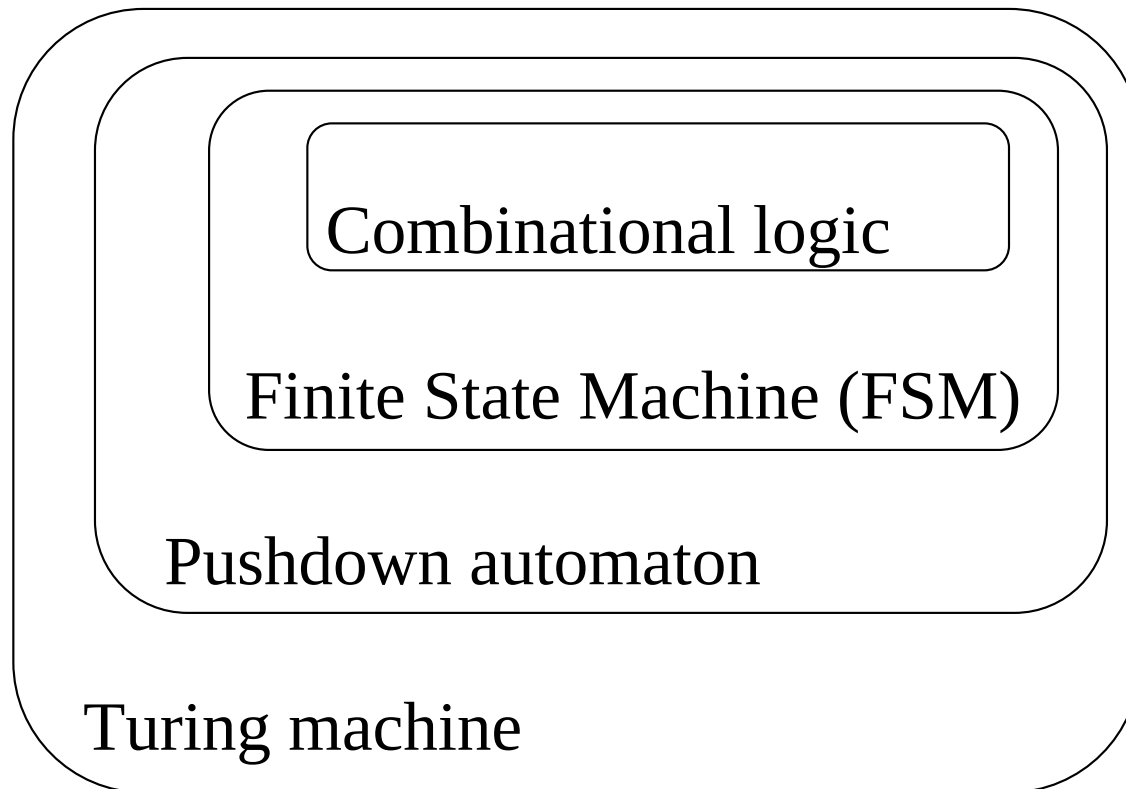
(Automação de Processos Industriais)

Discrete Event Systems: **Turing Machines, *Busy Beaver***

<http://users.isr.ist.utl.pt/~jag/courses/api1617/api1617.html>

Prof. José Gaspar, 2016/2017

Automata theory



Current state	Input SR	Next state
xx	11	11
xx	10	10
xx	01	01
xx	00	xx

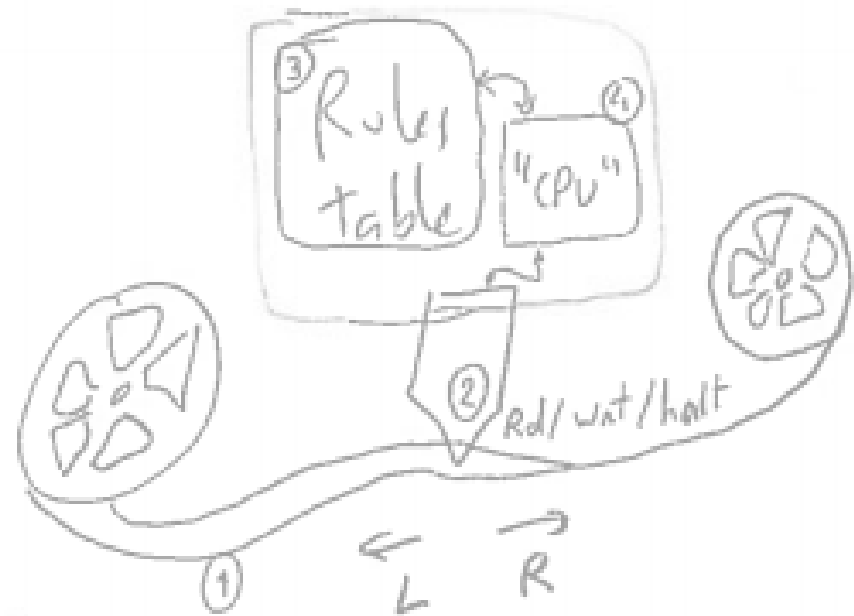
SR latch is a FSM example. The input $(S,R)=(0,0)$ keeps the memorized values

$$(Q, \bar{Q}) = (x, x)$$

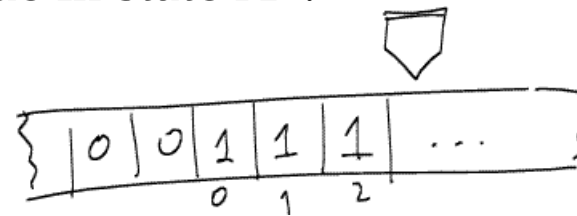
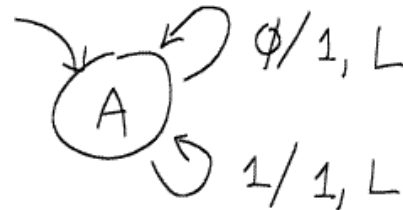
Turing machine (TM)

Components:

- (1) Infinite length magnetic **tape**
- (2) Read/Write head
- (3) **Rules table, FSM**
- (4) State register



Example of a simple **Rules table / FSM**. Using this FSM the TM **writes forever ones** into the tape. Read the FSM as “if 0 or 1 is read from the tape, then write 1 to the tape, move tape to the left and continue in state A”.



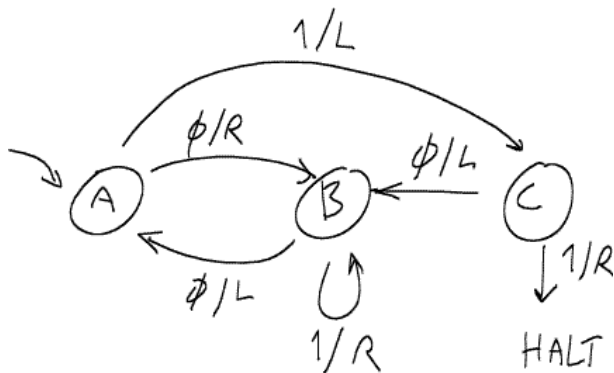
Note: a TM is not just an FSM; for example, it contains also an **infinite memory**.

Turing machine example: *Busy Beaver*

The objective is to fill the TM tape, with **ones**, as **many as possible**, using a rules table (FSM) with a **minimum number of states**.



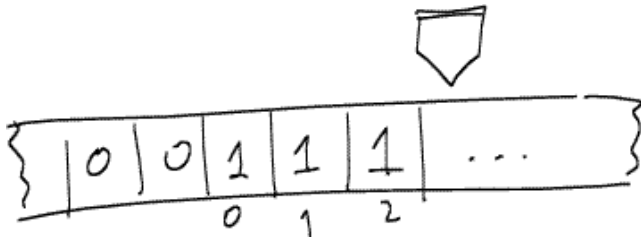
One implementation of the 3-states 2-symbols Busy Beaver is:



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

Turing machine:

- (1) **tape** and
- (2) read/write head



```
function TM_reset
```

```
global TMT
```

```
TMT= struct('pos',0,
            'val',[],
            'valNeg',[]);
```

```
function ret= TM_tape( op, arg1 )
```

```
% Tape for a Turing machine. Basic operations:
```

```
% read/write and move Left/Right/None
```

```
global TMT; if isempty(TMT), TM_reset; end
```

```
switch op
```

```
case 'reset', TM_reset;
```

```
case 'left', TMT.pos= TMT.pos+1;
```

```
case 'right', TMT.pos= TMT.pos-1;
```

```
case 'null_move' % do nothing
```

```
case 'read', % 1st call may need tape
```

```
    realloc_if_needed( TMT.pos );
```

```
    if TMT.pos>=0, ret= TMT.val( TMT.pos+1 );
```

```
    else ret= TMT.valNeg( -TMT.pos );
```

```
end
```

```
case 'write', % 1st call may need tape
```

```
    realloc_if_needed( TMT.pos );
```

```
    if TMT.pos>=0, TMT.val( TMT.pos+1 )= arg1;
```

```
    else TMT.valNeg( -TMT.pos )= arg1;
```

```
end
```

```
otherwise, error('inv op')
```

```
end
```

Turing machine: (3) rules table, **FSM** of Busy Beaver

```
function FSM= def_BusyBeaver3
```

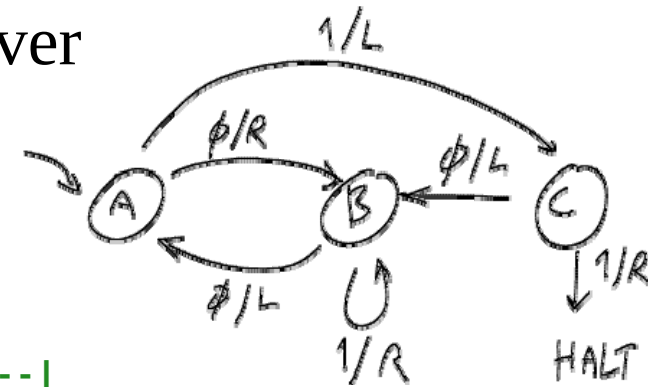
```
% FSM has four columns:
```

```
% curr_state, true_false_condition, actions, next_state
```

```
%      |- T/F cond -----|      |- write action and move action ----|
```

```
FSM= {
```

```
  'A', 'TM_tape("read")==0', 'TM_tape("write",1); TM_tape("right");', 'B';
  'A', 'TM_tape("read")==1', 'TM_tape("write",1); TM_tape("left");', 'C';
  'B', 'TM_tape("read")==0', 'TM_tape("write",1); TM_tape("left");', 'A';
  'B', 'TM_tape("read")==1', 'TM_tape("write",1); TM_tape("right");', 'B';
  'C', 'TM_tape("read")==0', 'TM_tape("write",1); TM_tape("left");', 'B';
  'C', 'TM_tape("read")==1', 'TM_tape("write",1); TM_tape("null_move");', 'halt';
};
```



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

Alternative, more compact, representation:

```
function FSM= def_BusyBeaver3
```

```
tbl= {'A01RB', 'A11LC', ...
      'B01LA', 'B11RB', ...
      'C01LB', 'C11NH'};
```

```
FSM= convert_table_to_list( tbl );
```

Turing machine:

(4) state register, curr_state
for **running** the machine

Recall the first line of the table:

```
FSM{1, :} =  
'A'  
'TM_tape("read")==0'  
'TM_tape("write",1); TM_tape("right");'  
'B'
```

and read it as “if current state is A
and tape read is zero, then write 1 to
the tape, move tape right, and the
next state is B”.

```
function TM_run  
TM_tape( 'reset' );  
FSM= TM_ini( 'BusyBeaver3' );  
curr_state= FSM{1,1};  
while ~strcmpi(curr_state, 'halt')  
    for i=1:size(FSM,1)  
        if strcmpi(FSM{i,1}, curr_state) ...  
            && eval( FSM{i,2} )  
            % found state and true condition  
            eval( FSM{i,3} );  
            % curr_state <- next state  
            curr_state= FSM{i,4};  
            break;  
        end  
    end  
end
```

Download the complete implementation from:

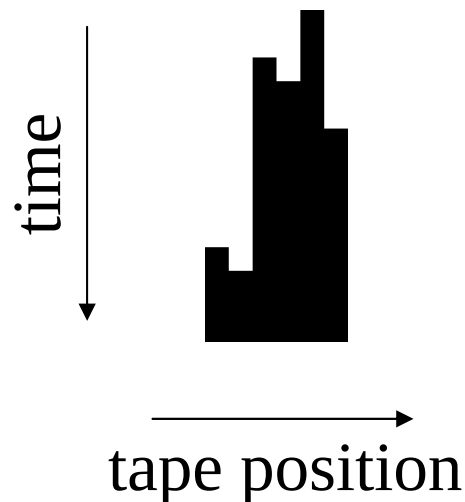
http://users.isr.ist.utl.pt/~jag/course_utils/Turing_Machines_sim/Turing_Machines_sim.html

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**



4-state Busy Beaver:

a0 -> b1r a1 -> b1l
b0 -> a1l b1 -> c0l
c0 -> h1r c1 -> d1l
d0 -> d1r d1 -> a0r

halts after **107 time steps**
fills **13 ones**

<i>States</i>	<i>Halts after n time steps</i>	<i>Fills m ones in the tape</i>
2	6	4
3	21	6
4	107	13
5	47,176,870 ?	4098 ?
6	> 7.4 × 10³⁶⁵³⁴	> 3.5 × 10¹⁸²⁶⁷

