Data Science and Automation

Lesson 20

PLC – SFC Language

History

Before the ‘60 the SEQUENTIAL-ORIENTED CONTROL was considered only an EXTENSION of the DIGITAL CONTROL. No other kinds of control (such as the one for continuous systems) were considered.

At the end of the ‘60 the FINITE STATE AUTOMATA were introduced. Their formal models allow an in-depth MATHEMATICAL ANALYSIS, that is non very suitable for the design of algorithms used in industrial automation.

History

In the ’70, automation systems were only designed using graphical representation (similar to circuits) or textual descriptions.

In 1975, GRAFCET was introduced in France, where a commission to formalize a design language was established. GRAFCET was a sequence-oriented language.

In the ’80, the GRAFCET language was accepted by the IEC and included into the norm IEC 848.

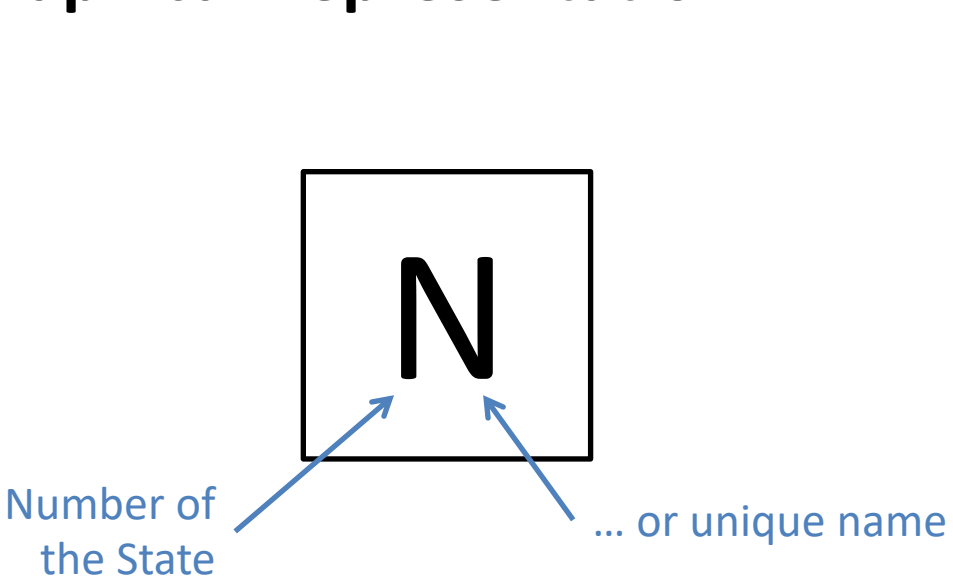
Basic Elements

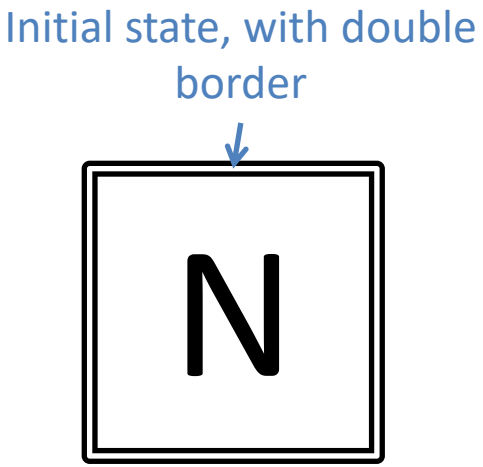
State (phase)

Definition

A STATE is a precise operative condition of a complex system.

Graphical Representation





Basic Elements

State (phase)

Remarks

A STATE is an INVARIANT condition of the system, which can be modified only when an EVENT occurs.

During the execution, all the STATES in all the part of the system can only be in one of the following condition: ACTIVE or INACTIVE.

Basic Elements

State (phase)

Syntax

The PLC defines two variables for each state:

- *MARKER*: it is a boolean variable that is TRUE if the state is active, FALSE otherwise
- *TIMER*, it is an internal variable that contains the duration of the activation (if the state is active)

Usually we indicate the two variables as follows:

- MARKER: NameOfTheState.X
- TIMER: NameOfTheState.T

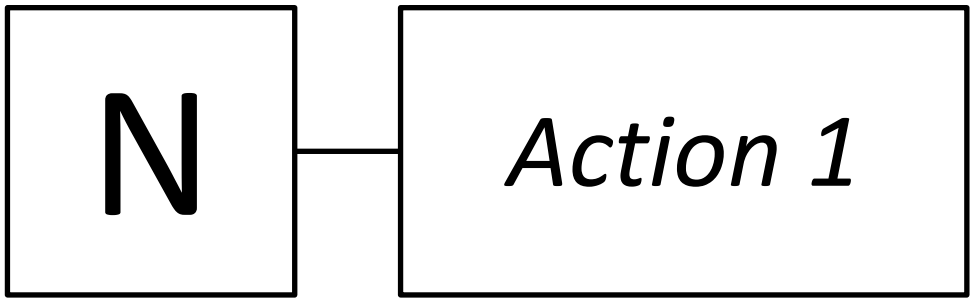
Basic Elements

Action

Definition

We call ACTIONS all the OPERATIONS executed by the system when it is in a precise operative condition (aka STATE).

Graphical Representation



Basic Elements

Action

Remarks

Each STATE of the system is composed of a finite set of ACTIONS.

From the point of view of a CONTROL SYSTEM, which is responsible for carrying out the LOGICAL-SEQUENTIAL control PROGRAM, an ACTION is equivalent of one or more PROCEDURES, i.e. a SET OF INSTRUCTIONS that are serially executed while the STATE is ACTIVE, so until an EVENT does not change the STATE.

Basic Elements

Action

Syntax

Each PLC defines, for each action, three variables:

- A_m, an unique identifier of the action
- Q_m, that defines the type of the action
- V_m, a boolean variable representing the status of the action

N.B.: In many development environment the actions (and also the transitions) can be defined also using different languages of the norm IEC 61131.

Basic Elements

Action

Syntax

There are many action types:

- N: Normal non stored

  If the action A_n is a N action, it will be executed for each execution cycle of the PLC if it is associated to an active state.

- P: Pulse

  If the action A_n is a P action, it will be executed only the first execution cycle of the PLC, in which the state is active.

Basic Elements

Action

Syntax

There are many action types:

- S: Set

  If the action A_n is an S action, it will be executed for each execution cycle of the PLC until the same action A_n (in a subsequent phase) will have the type R.

- R: Reset

  If the action A_n is a R action (as said before), it terminates the execution of a previously enabled action with the type S.

Elementi base

Action

Syntax

There are many action types:

- L: Limited Time

  If the action A_n is a L action, it will be executed for each execution cycle of the PLC for t#Ts seconds starting from the activation of the state. N.B.: If the state is disabled, the action terminates.

- D: Time Delayed

  If the action A_n is a D action, it will be executed for each execution cycle of the PLC after t#Ts seconds (starting from the activation of the state)

Basic Elements

Action

Syntax

There are many action types:

- SD: Stored / time Delayed

  If the action A_n is a SD action, it is equivalent to an S action delayed of t#Ts seconds.

- SL: Stored / time Limited

  If the action A_n is a SL action, it is equivalent to an S action disabled after t#Ts seconds (no matter what is the state currently enabled).

Basic Elements

Transition

Definitions

The PASSING from a PREVIOUS STATE to a SUBSEQUENT STATE following an event, is called TRANSITION.

The LOGICAL verification that determines the occurrence or not of an EVENT is called CONDITION.

Remarks

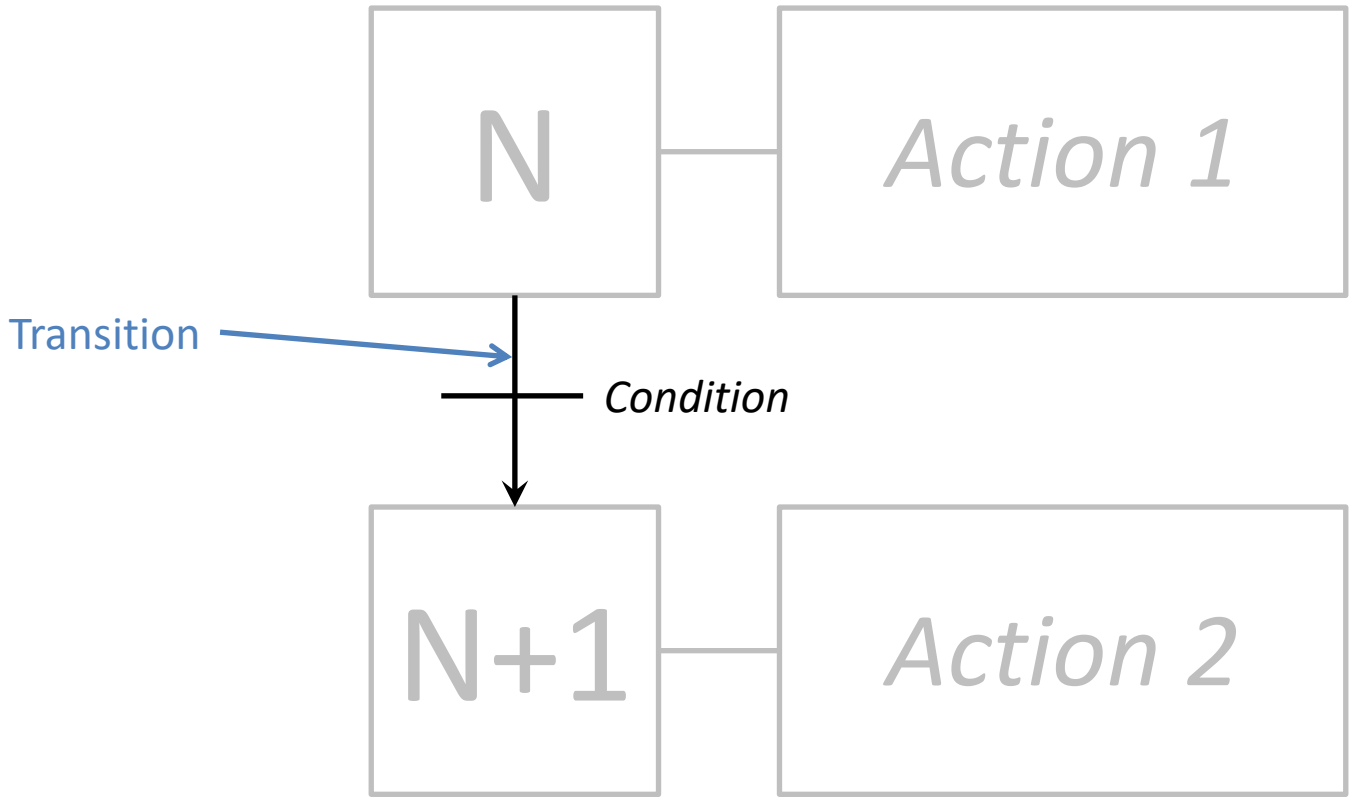
Each CONDITION is associated to a TRANSITION.

Each CONDITION is expressed using a logical function, i.e. a boolean expression that can be TRUE or FALSE.

Basic Elements

Transition

Graphical Representation



Evolution Rules

Remind!!

Rule 1

A transition is ENABLED if all the upstream phases are activated.

A transition can be OVERCOME if it is enabled and the associated condition is true.

Evolution Rules

Rule 2

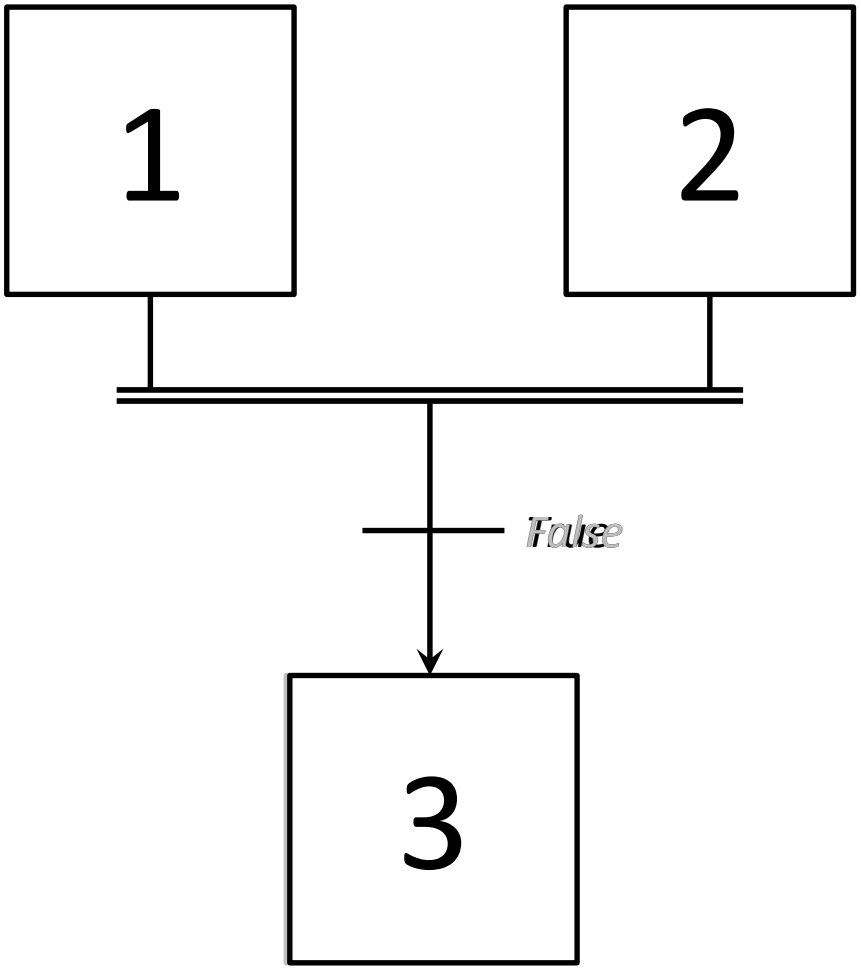
When a transition can be overcome, it is always overcome: all the upstream phases are disabled and all the downstream phased are enabled.

It is simple: when a condition is TRUE, all the upstream phases are disabled and all the downstream phased are enabled!

N.B.: The correct sequence is state– transition – state – transition - ....

Evolution Rules

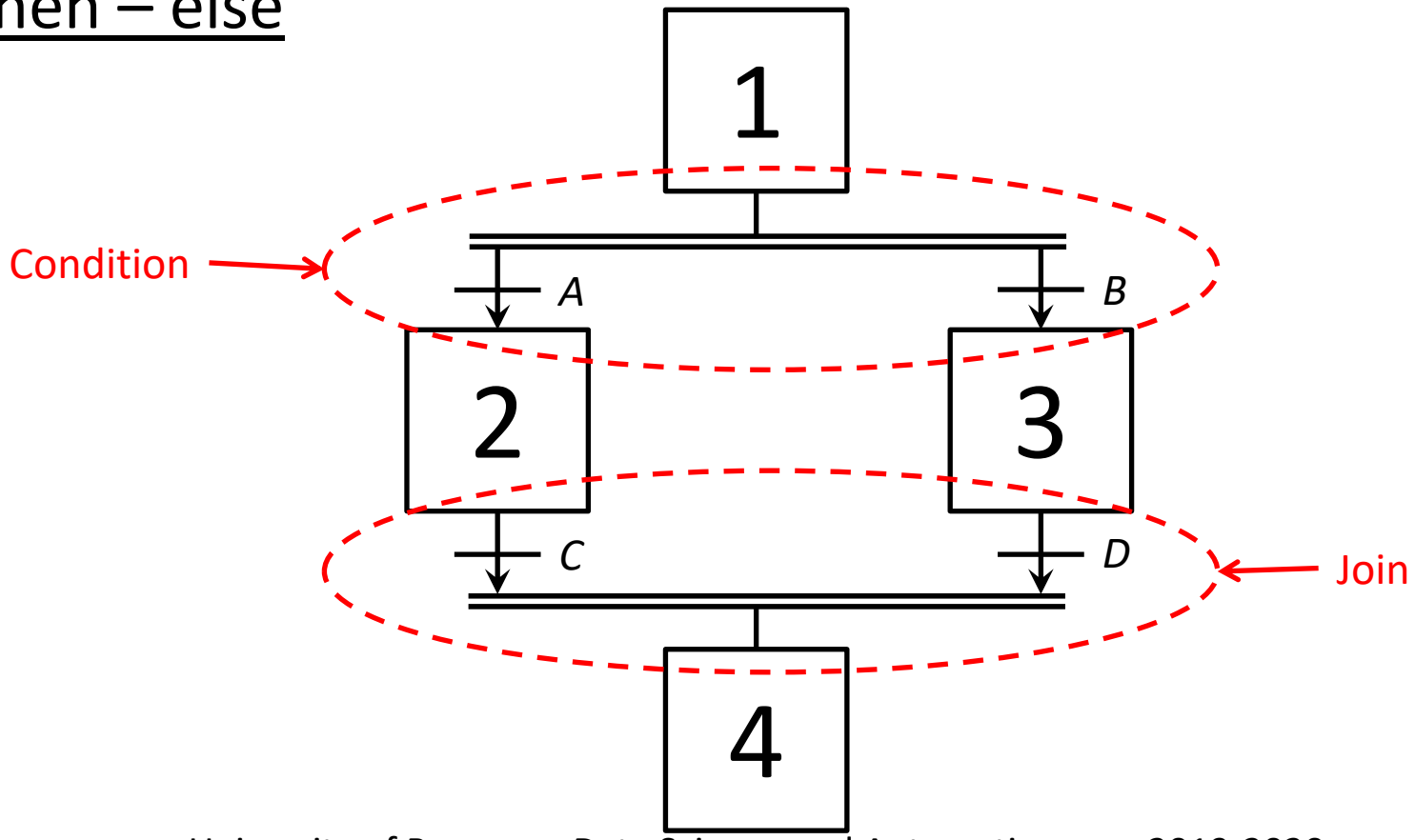
Example



Basic Structures

By using the previous rules, we can create the following basic structures:

<u>If – then – else</u>

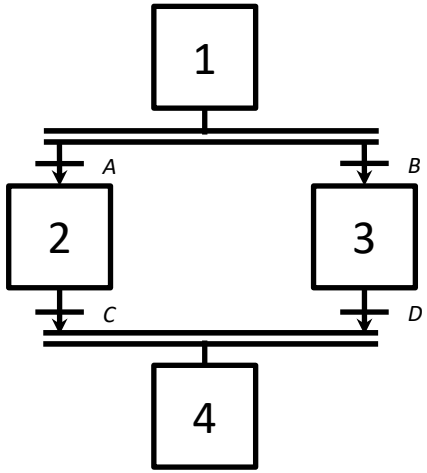


Basic Structures

Remarks

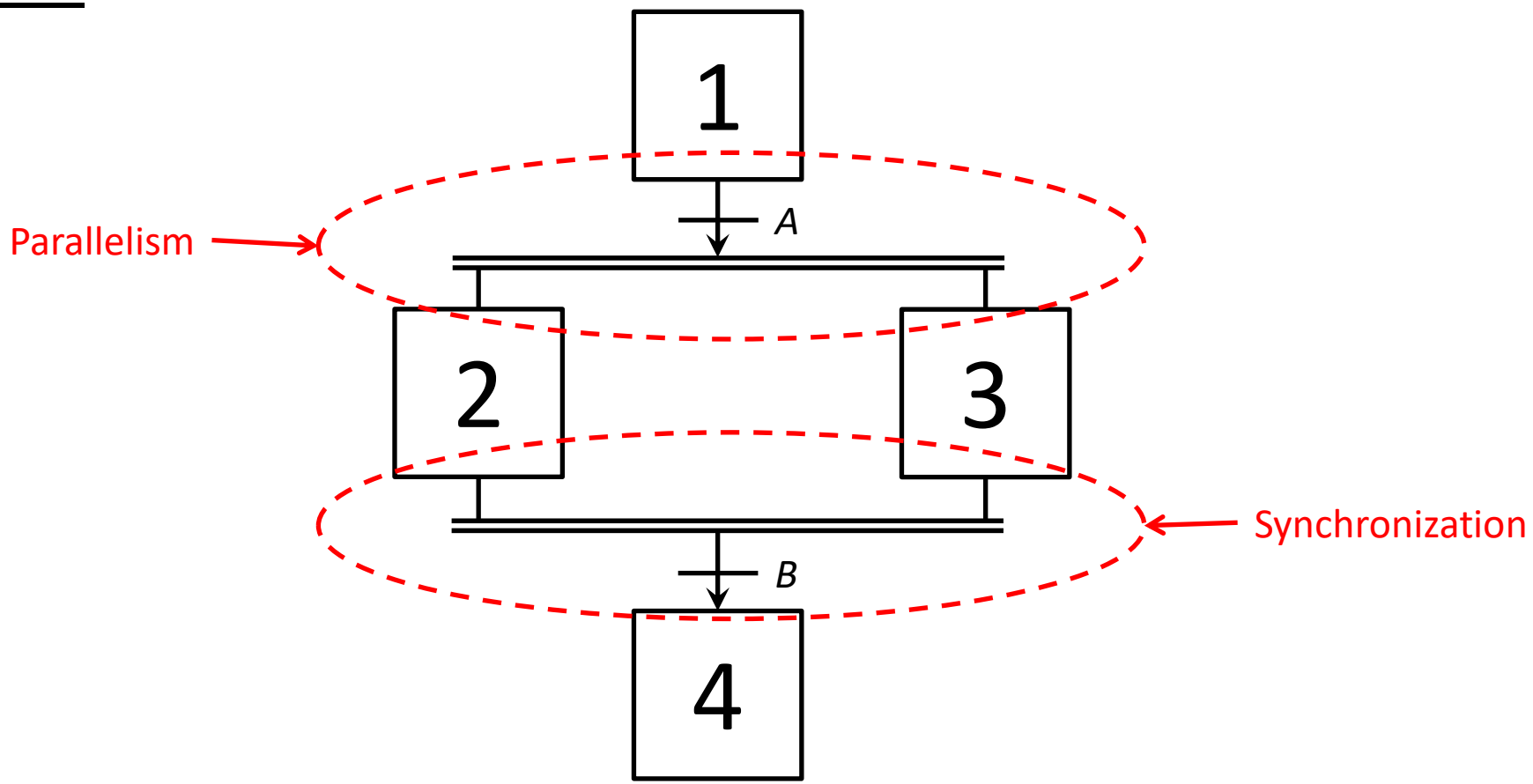
The conditions named *A* and *B* must be mutually exclusive.

If they weren't, you would risk having it phase 4 activated «twice».



Basic Structures

Parallel

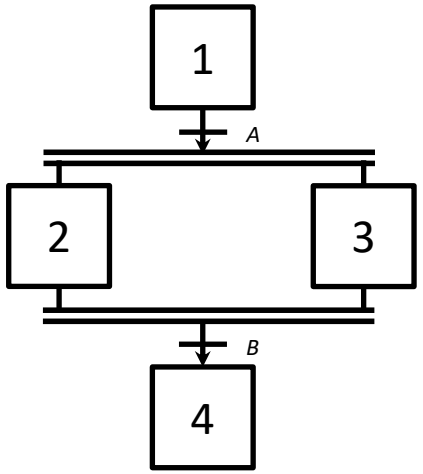


Basic Structures

Remarks

The states 2 and 3 are activated at the same time.

If one of the branches has more than a state, the check of the condition B is executed when all the branches have finished the last phase.



SFC in an actual case study

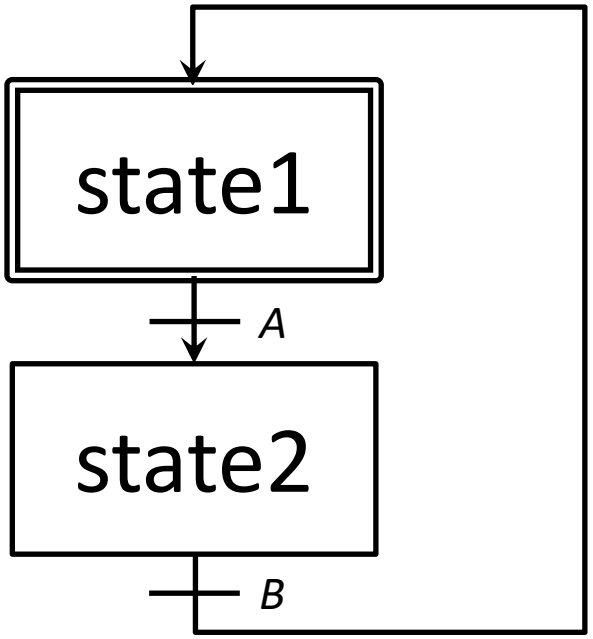
In many development environment (such as for B&R Automation Studio) the SFC language is integrated with the other languages of the norm IEC 61131.

This means that, for each state, it is possible to associate the execution of a piece of code written in one of the other language.

For this reason, in B&R Automation Studio some of the actions described in the previous slides are not present.

Example

Basic example



What happens if A and B are both true?

The PLC continues to change between state1 and state2

Example

Actions and if – then – else

Develop a software that discard the pieces with weight under a threshold, by the activation of an actuator for 1 second.

Inputs

Cell: photocell

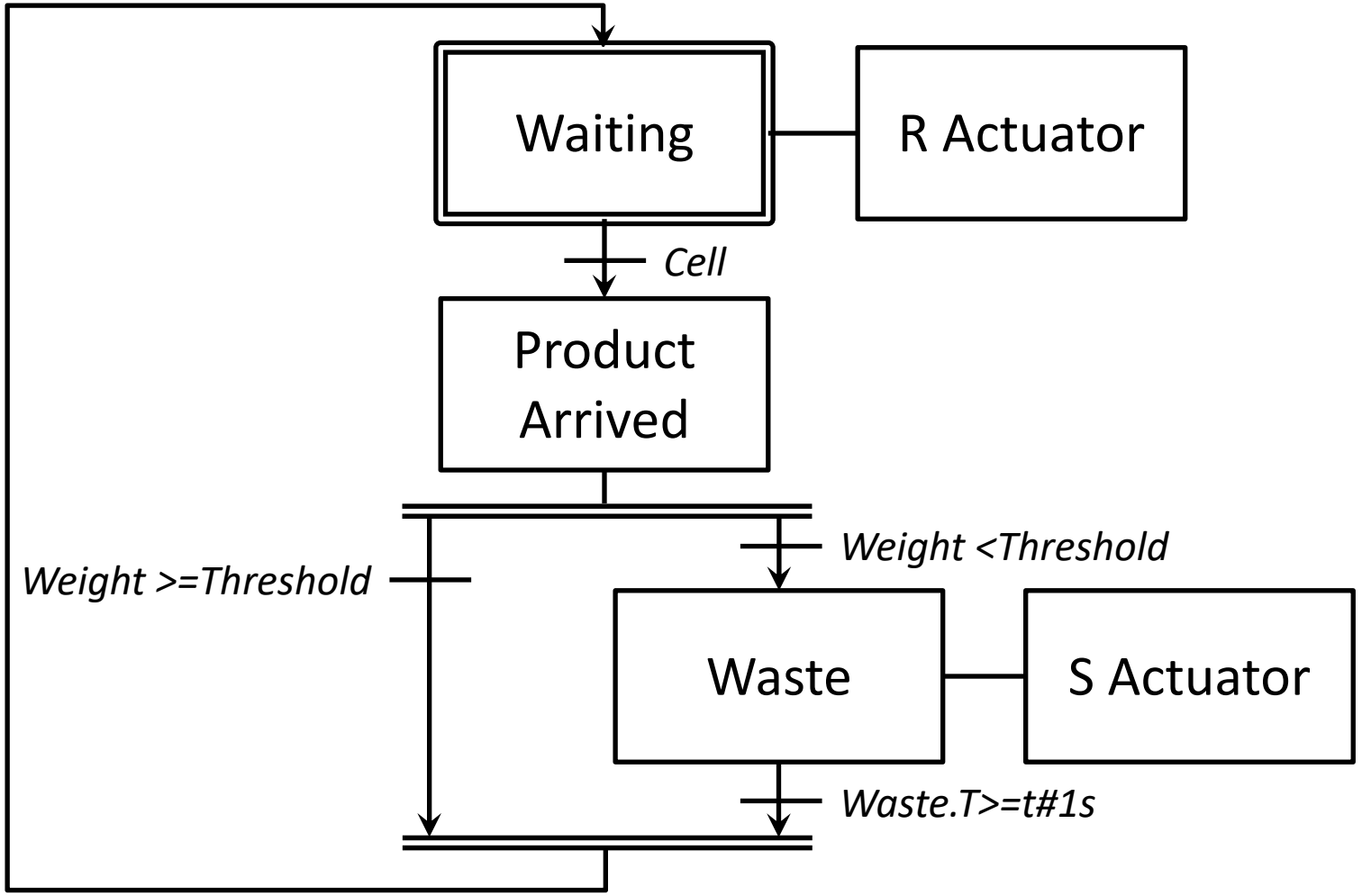
Weight: weight of the product

Outputs

Actuator: activation of the actuator

Example

Actions and if – then – else



Example

Example with states and timers

Develop a software that activates the electro-valve U if a «sensor too-full» remains active for more than 10s and keep U activated for 30 s.

Inputs

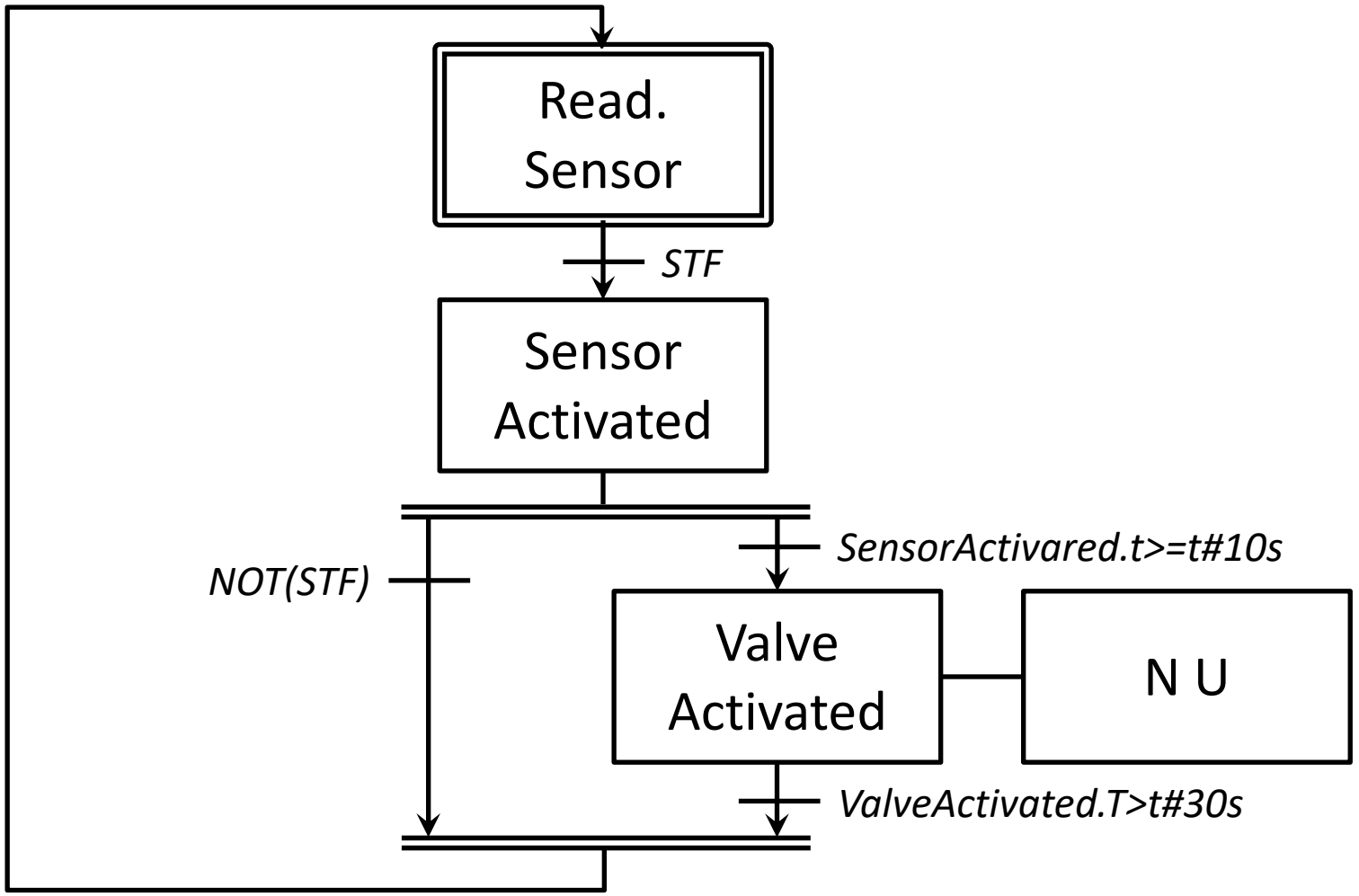
STF: Sensor too-full

Outputs

U: Electro-valve activation

Example

Example with states and timers



Example

Actions written in other languages

Develop a software that signals the tenth customer entering a shop. The customers are counted using a photocell at the entry of the shop.

Input

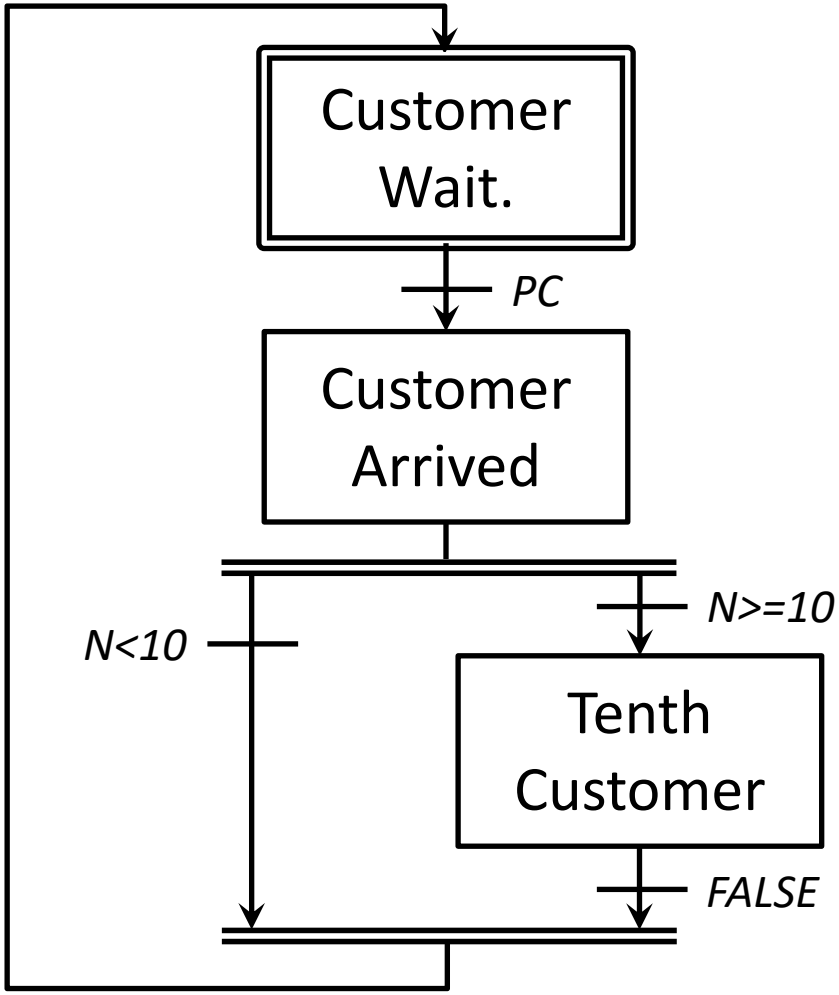
PC: photocell

Outputs

Tenth: ten customers entered the shop

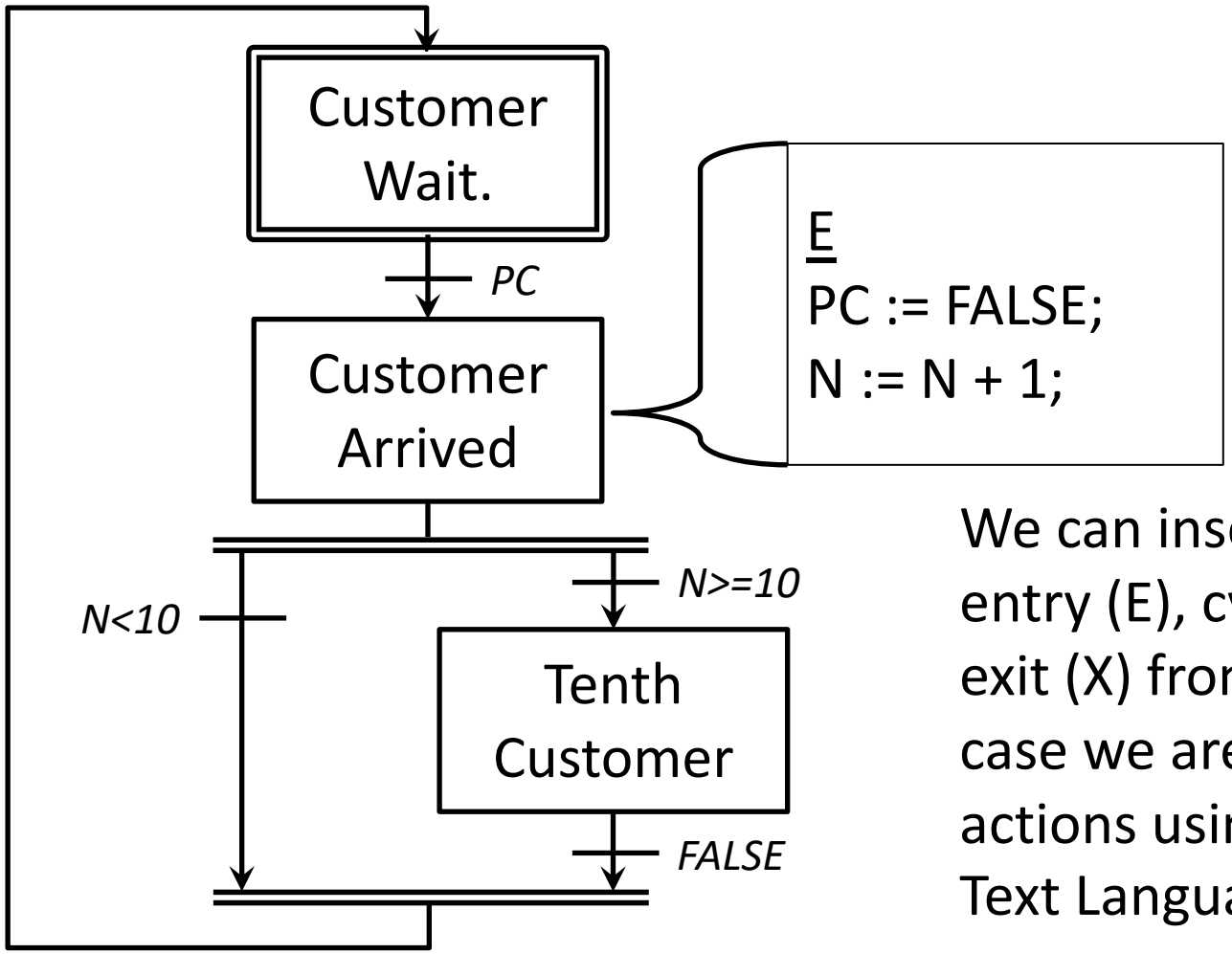
Example

Actions written in other languages



Example

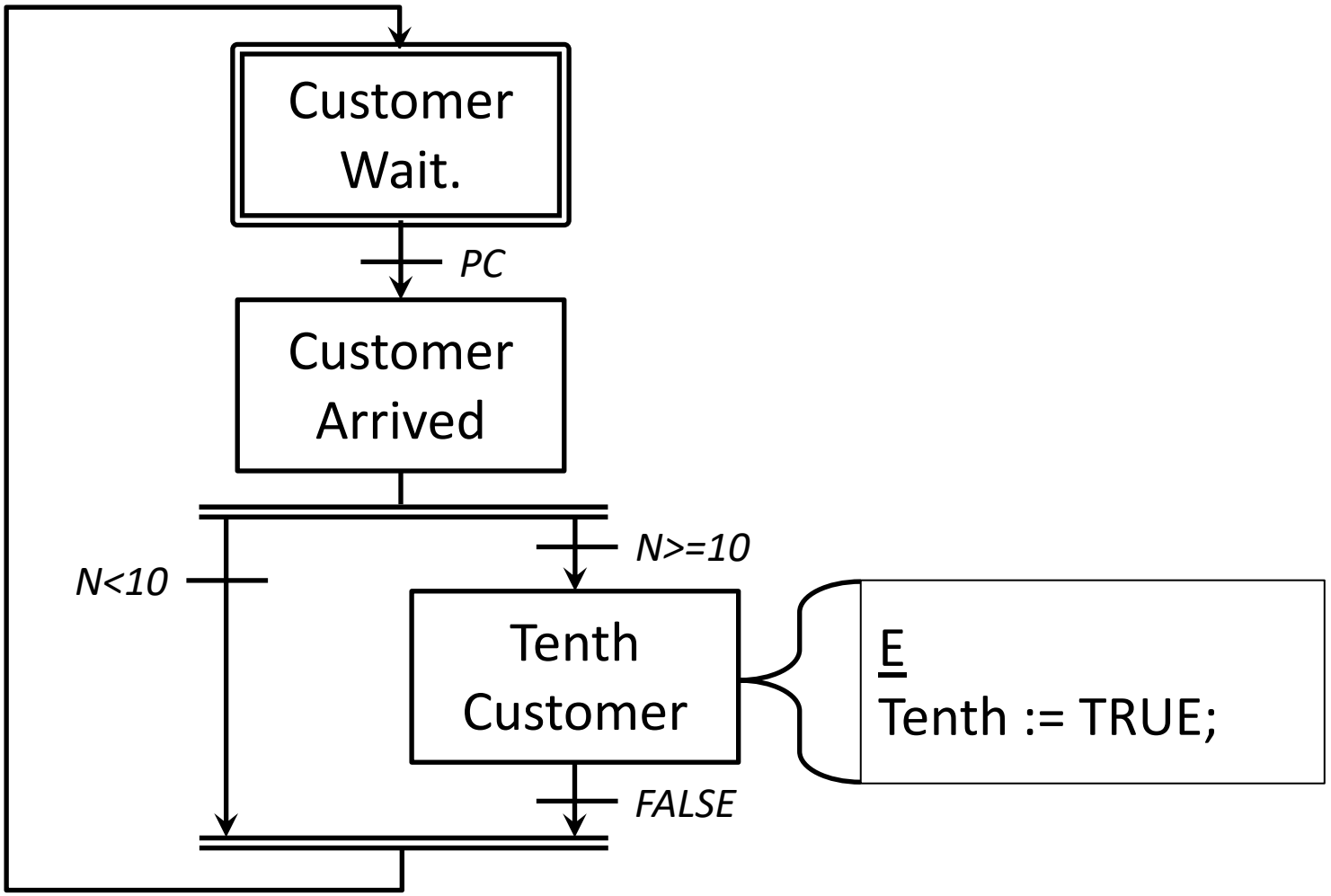
Actions written in other languages



We can insert actions on the entry (E), cyclic (C) or at the exit (X) from the state. In this case we are writing these actions using the Sructured Text Language

Example

Actions written in other languages



From SFC to Ladder

There is the possibility to translate a SFC program to a Ladder Program, by following some simple rules that are independent with respect to the program structure.

During this course we will see only the algorithm “without search for stability”.

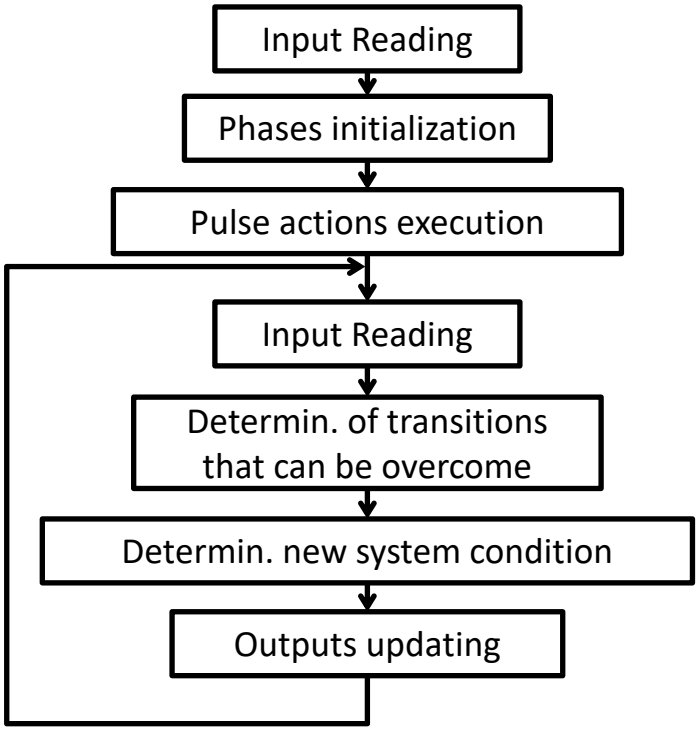
N.B.: All the rules we will see are “standard-rules”, so they can be applied to all the SFC programs!

From SFC to Ladder

Let's see this algorithm that is calles «without search for stability».

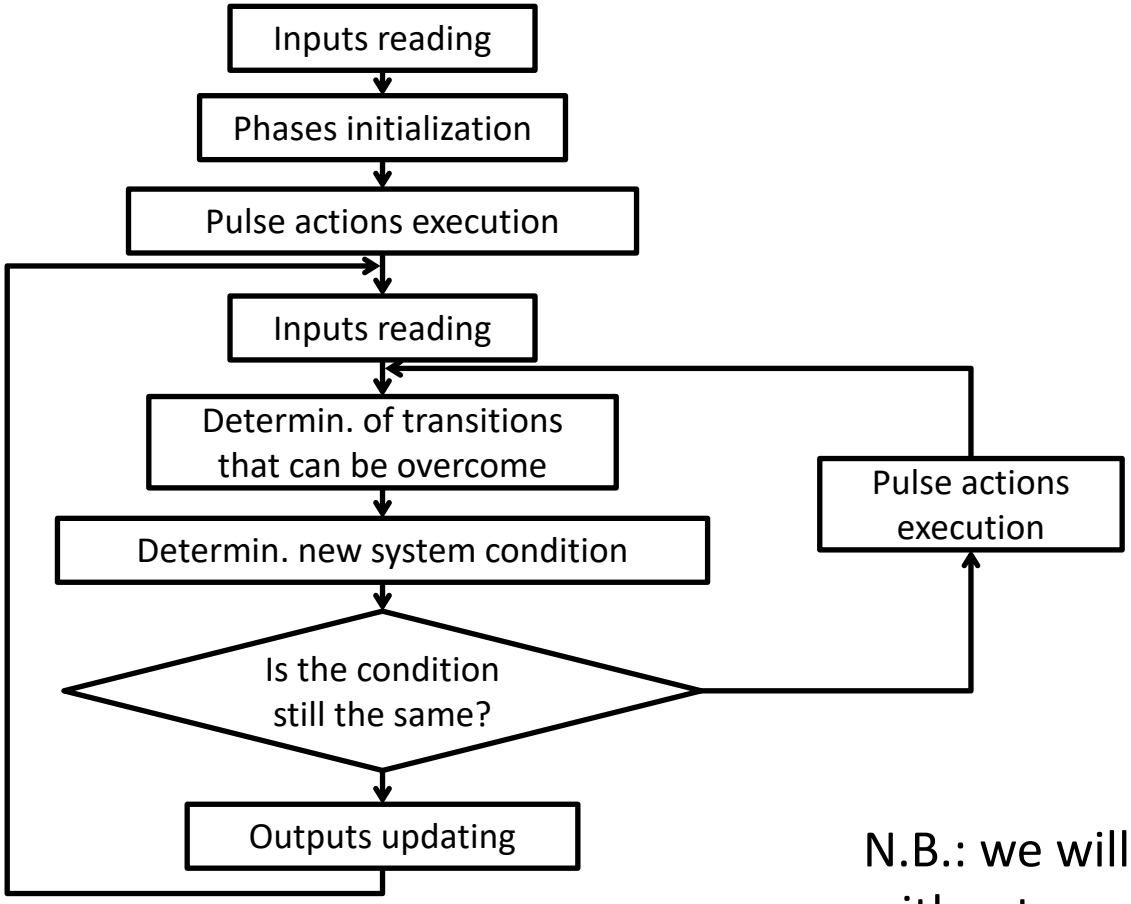
The name given to the algorithm means that if we have a non-stable phase (i.e. a phase with the exit transition already enabled at the time of the phase activation), this solution is not correct:

- If the action was continue, the action may not be activated
- If the action was impulsive, yes



From SFC to Ladder

For information, here is the algorithm <u>with</u> search for stability



N.B.: we will use the algorithm without search for stability

From SFC to Ladder

The ladder code must be composed of four sections:

- Initialization

  It is executed only during the first step of the program

- Actions execution

  The actions connected to the active phase are enabled

- Evaluation of the conditions

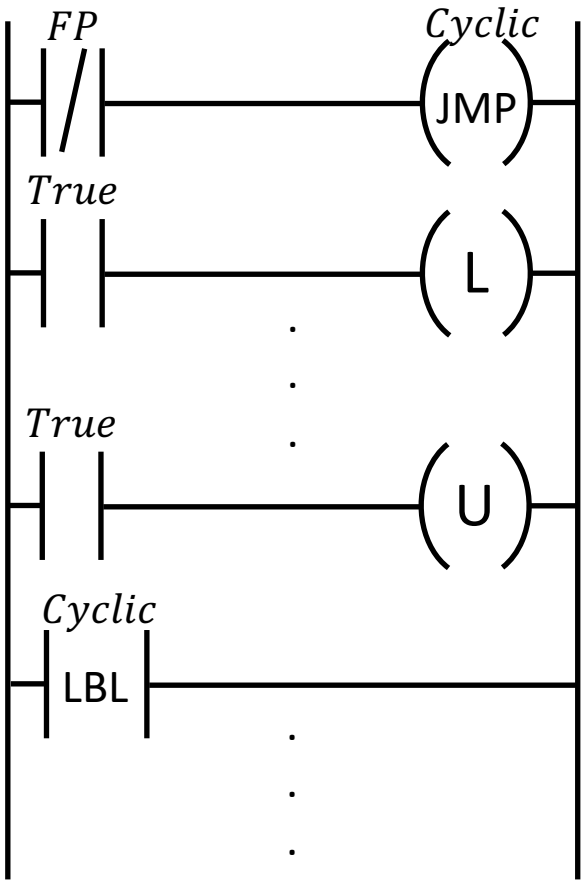
  The transitions connected to the active phases are enabled

- Condition update

  Verification of the transition between the current and the following phases

From SFC to Ladder

Initialization

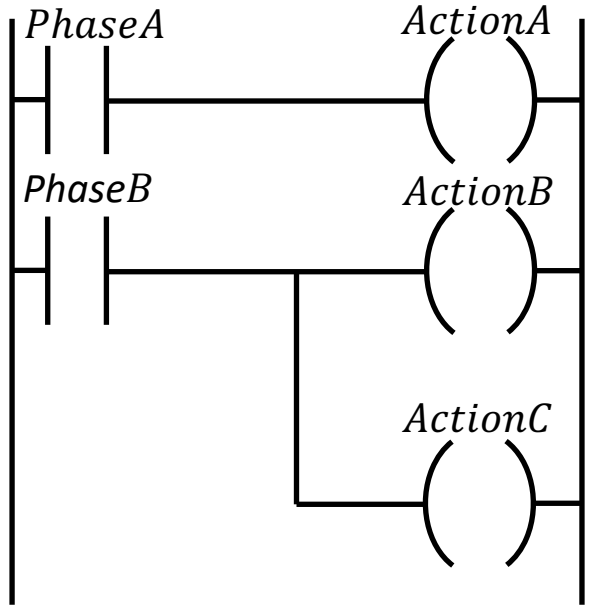


N.B.: This piece of code, in Automation Studio, may be added inside the _INIT file

From SFC to Ladder

Actions execution

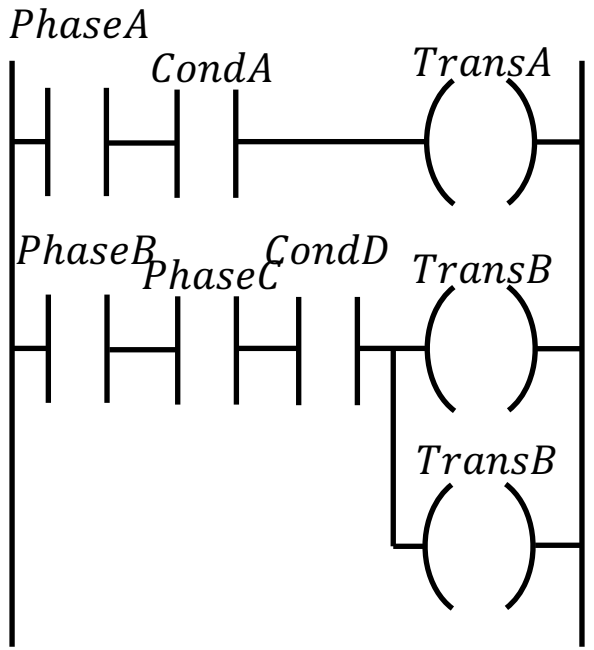
- For each action we write a rung in which, using an OR composition, we put all the phases that enable the considered action
- For the «stored» actions, latch and unletch coils are used
- For the «pulse» actions, we need to create a network that generates the pulse.



From SFC to Ladder

Evaluation of the conditions

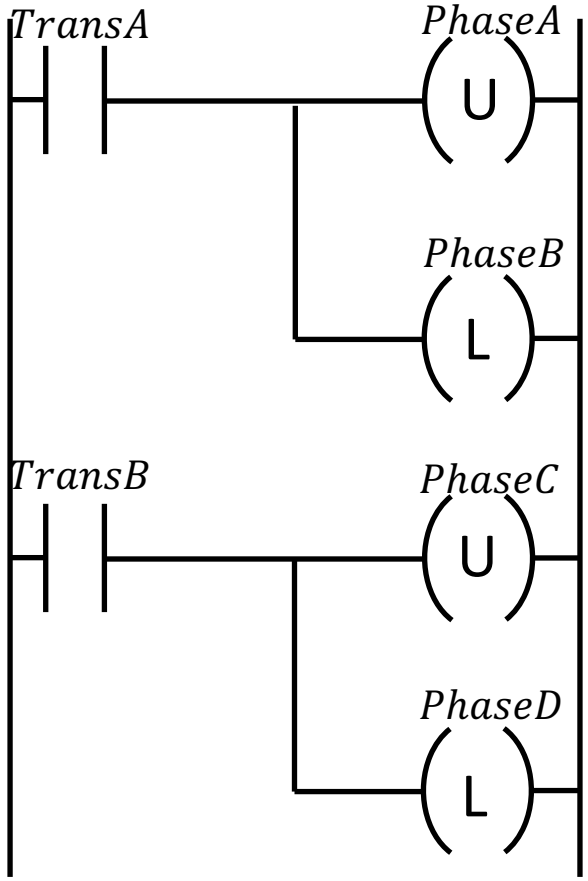
- For each phase, all the exit conditions that enables the transition are checked.
- In the case of transitions that are connected to more then a phase (parallel) we have to put in an AND the boolean signal of the phases.



From SFC to Ladder

Condition update

When we have an active transition, all upstream phases are disabled and all downstream phases are enabled



Exercices

Exercise 1

We want to create a system that allows the transportation of stones with a cart. The operator starts the system by pressing the button START. The cart follows the rail from left to right and stops itself, waiting the loading of the stones.

When the stones are accumulated into a tank, they are transferred into the cart.

After that, the cart has to move automatically down the rail from right to left.

Exercise 1

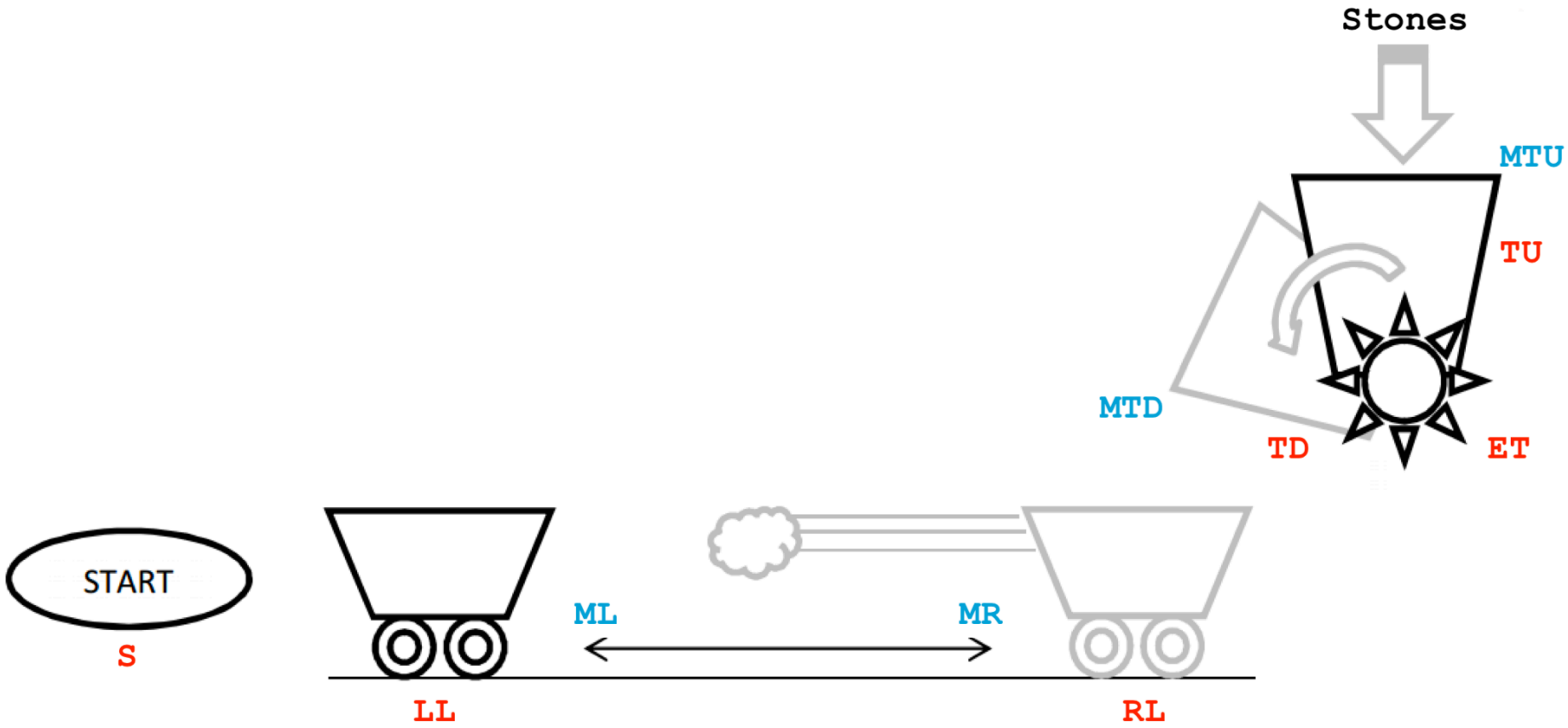
We have six sensors as INPUTS:

- S : Start
- LL : Left Limit-switch
- RL : Right Limit-switch
- ET : Empty tank
- TD : Tank down
- TU : Tank up

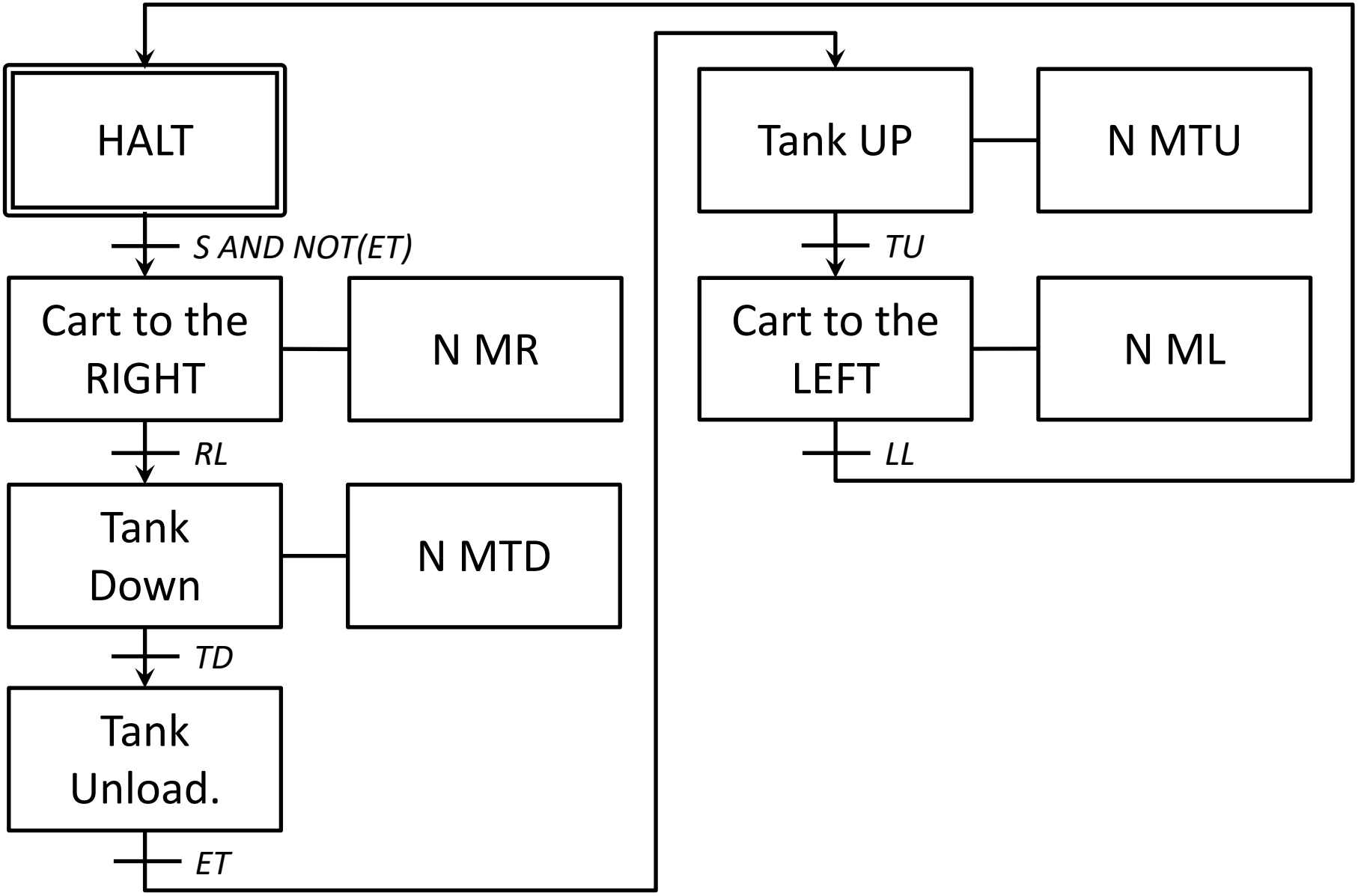
We have the following OUTPUTS:

- MR : Cart motor to the right
- ML : Cart motor to the left
- MTD : Tank's motor down
- MTU : Tank's motor up

Exercise 1



Exercise 1


HALT
S AND NOT(ET)
Cart to the RIGHT
N MR
RL
Tank Down
N MTD
TD
Tank Unload.
ET
Tank UP
N MTU
TU
Cart to the LEFT
N ML
LL

Exercise 1.1

Let’s add to the previous exercise a maintenance stop every 100 cycles.

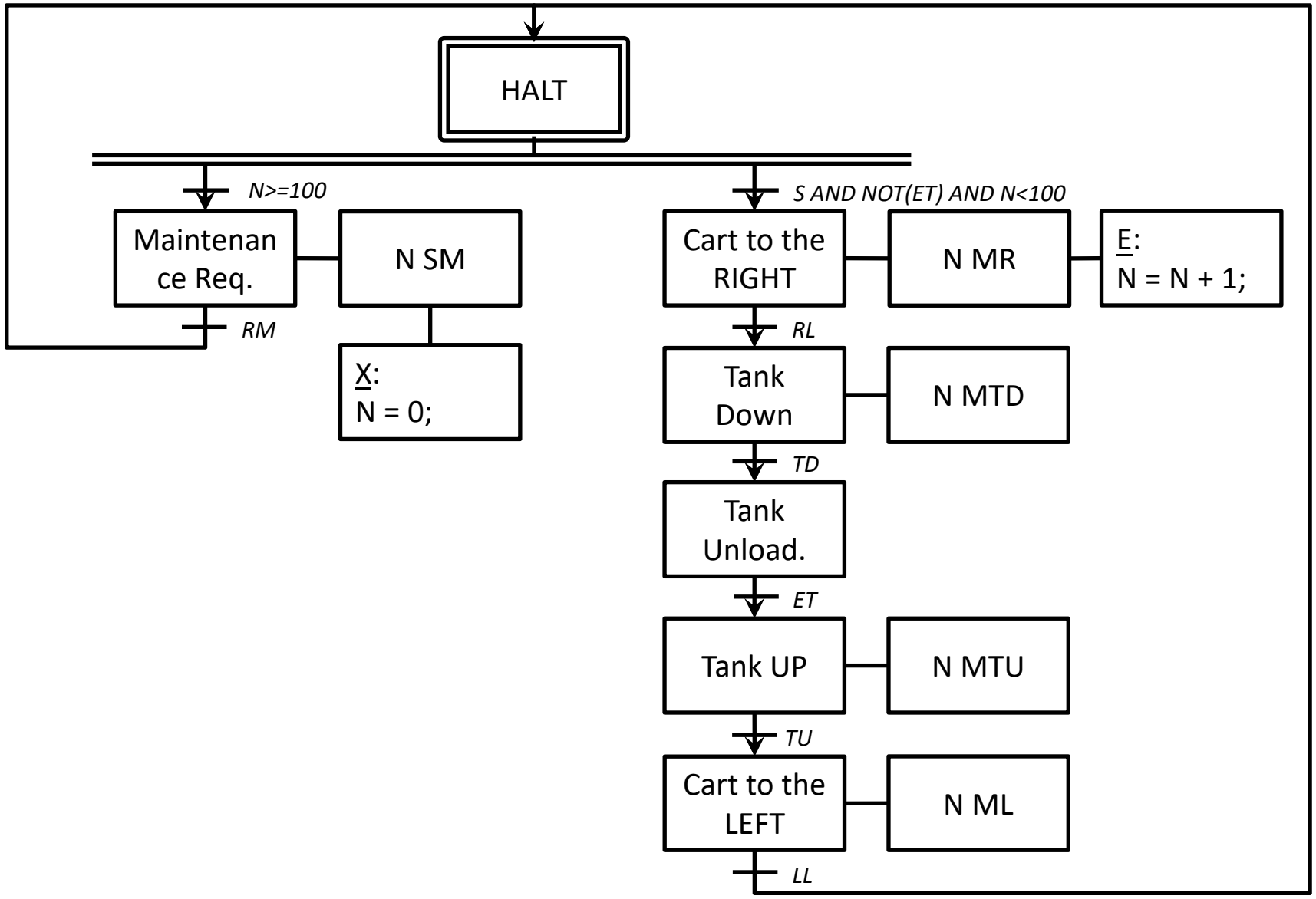
We have to add:

SM (stop for maintenance) as output

RM (maintenance reset) as input

N.B.: We need also an internal counter variable (N) to count how many cycles the system have excuted!

Exercise 1.1


HALT
N>=100
Maintenance Req.
N SM
RM
X:
N = 0;
S AND NOT(ET) AND N<100
Cart to the RIGHT
N MR
E:
N = N + 1;
RL
Tank Down
N MTD
TD
Tank Unload.
ET
Tank UP
N MTU
TU
Cart to the LEFT
N ML
LL

Exercise 2

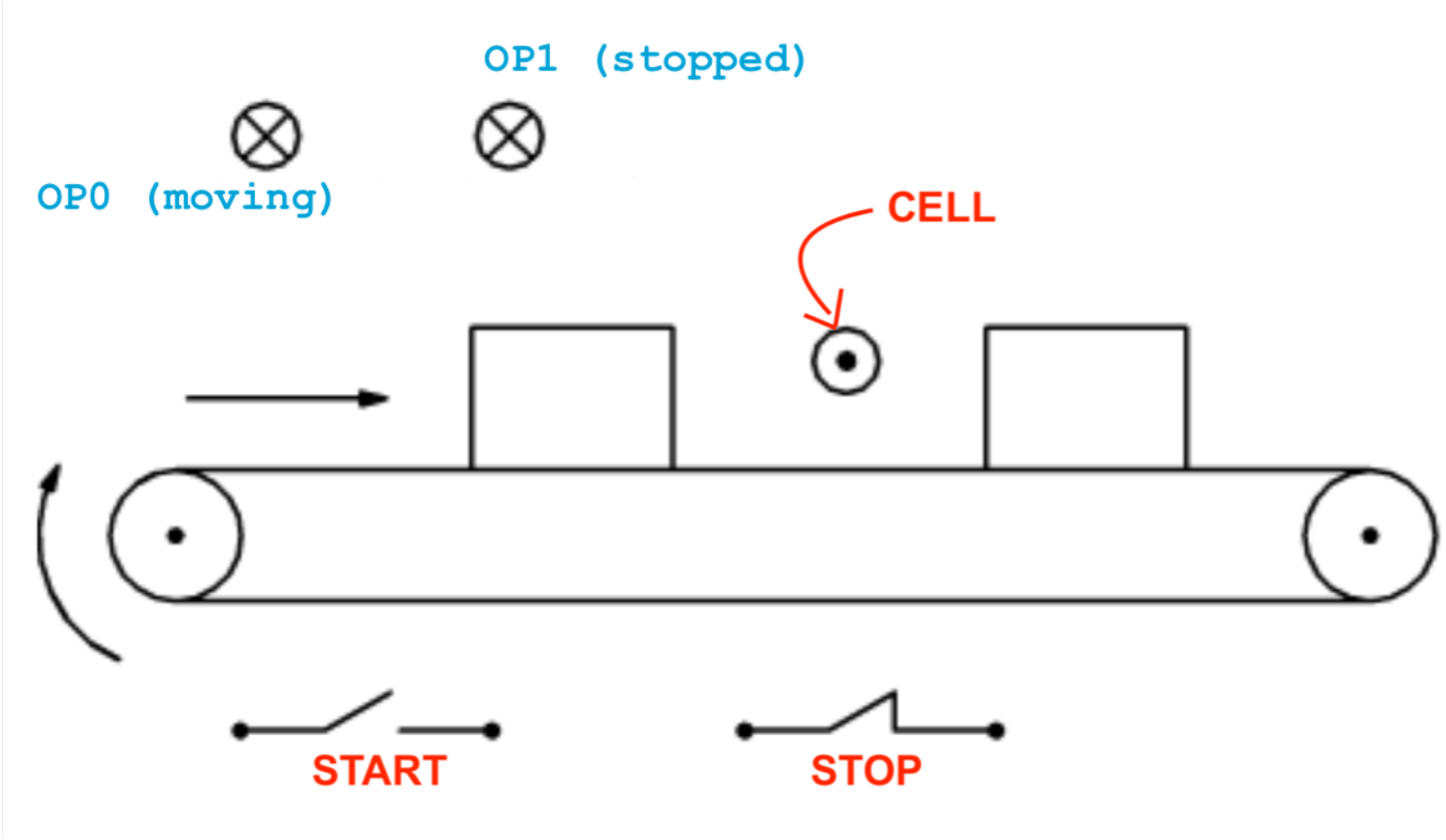
Consider a piece-counter with a conveyor belt.

The conveyor belt is moved by a motor which is controlled by two buttons: START and STOP. There are two lamps that signal the state of the conveyor belt(stopped/moving).

Each piece is placed at the beginning of the conveyor belt and, for each piece that passes in front of a photo-cell, a counter has to be incremented. The system has to automatically stop itself every 50 pieces.

The conveyor belt can be stopped at any time by pressing the STOP button. The restarting of the conveyor belt can occur by means of the pression of the START button, but, in this case, the counter have to retain its value.

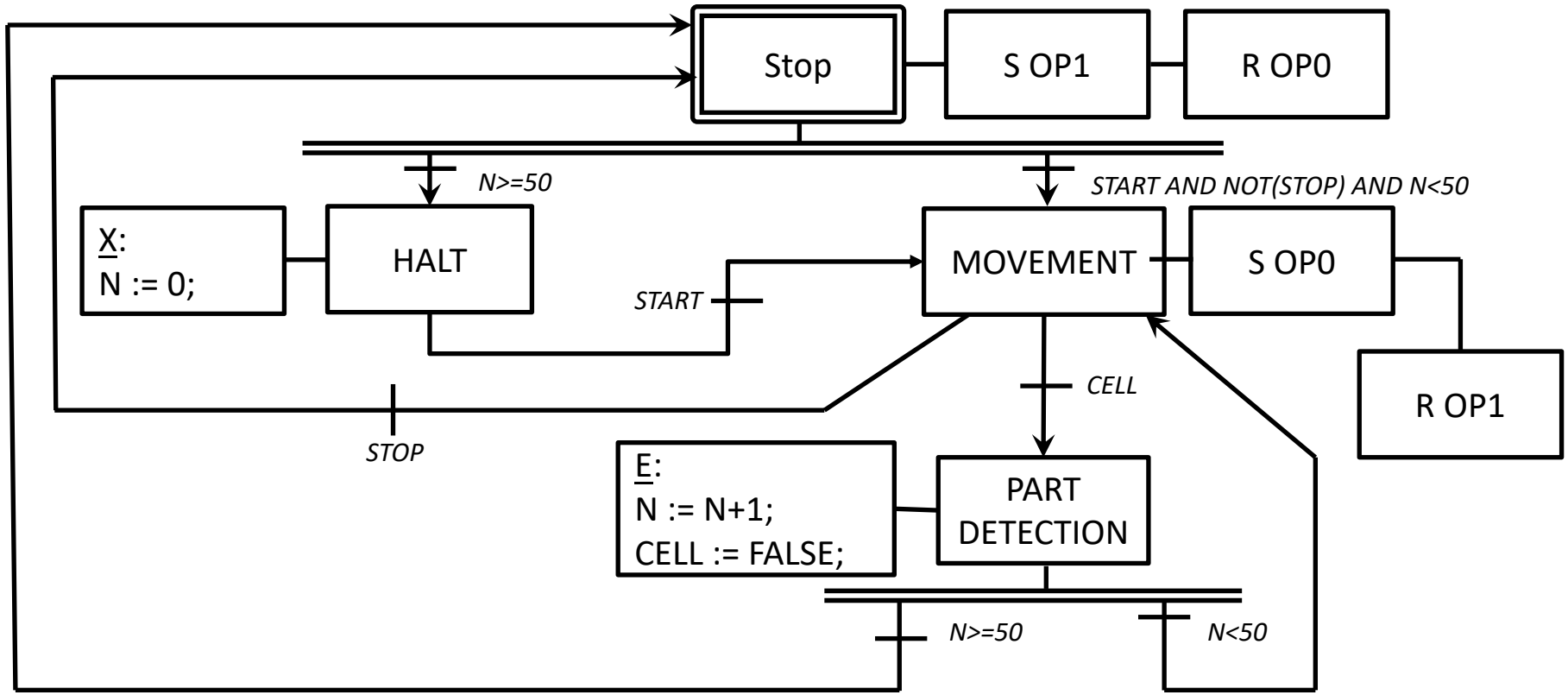
Exercise 2



Exercise 2

- At startup, the motor that drives the conveyor belt must be halted, so only *OP1* has to be on.
- By pressing the *START* button, the motor starts: OP0 has to turn on and OP1 has to turn off.
- Every time the motor is running and a new piece is detected in front of the photocell *CELL*, the counter has to be increased.
- When the counter reach the value 50, the conveyor belt has to be stopped: OP1 has to turn on and OP0 has to turn off. *(in a following restart, the counter must start from the value 0).*
- When the conveyor belt is stopped before the value 50 the counter must retain its value.

Exercise 2


Stop
S OP1
R OP0
N>=50
START AND NOT(STOP) AND N<50
X:
N := 0;
HALT
START
MOVEMENT
S OP0
R OP1
CELL
STOP
E:
N := N+1;
CELL := FALSE;
PART
DETECTION
N>=50
N<50

Exercise 3

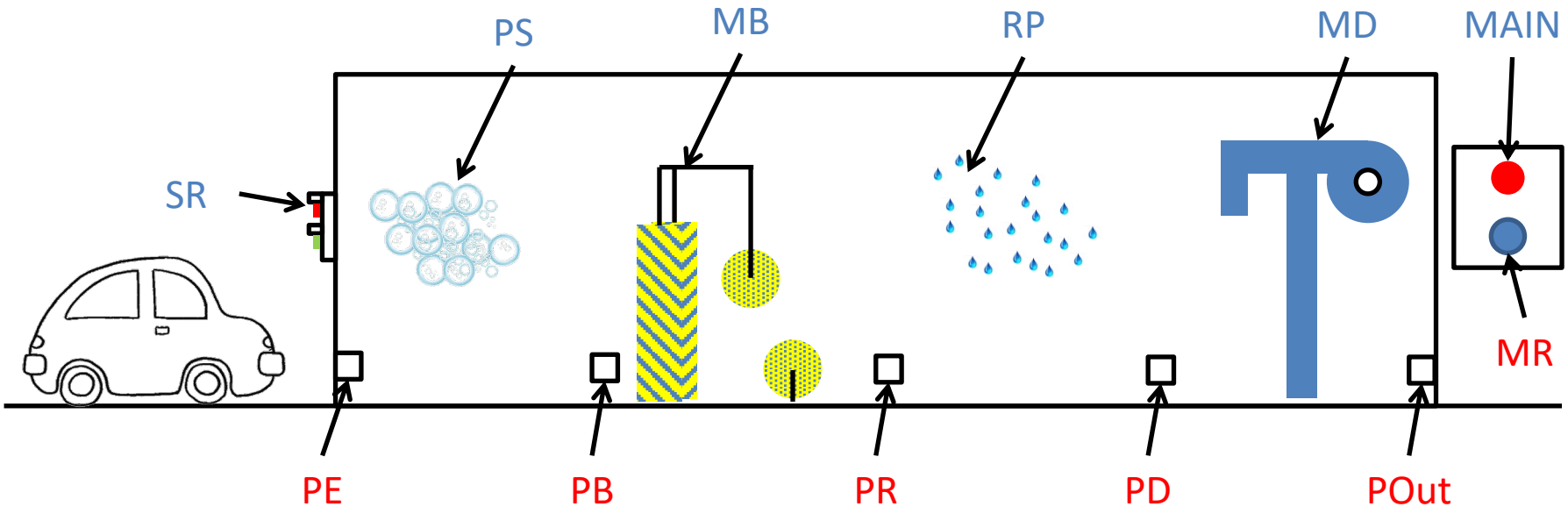
Consider an automatic carwash system.

The customer approaches to the conveyor belt when the semaphore is green. The phases of the washing are: soaping, brushing, rinsing e drying.

All the phases are preceded by a photocell that signals the arrival of the car in that new section of the carwash.

Every 1000 washing, the carwash system have to block itself waiting for the maintenance, that is executed by an operator.

Exercise 3



Input		Output	
PE	Entry photocell	SR	Stop semaphore (0=GREEN, 1=RED)
PB	Brushing photocell	PS	Soaping pump
PR	Rinsing photocell	MB	Brushing motor
PD	Drying photocell	RP	Rinsing pump
POut	Out photocell	MD	Drying motor
MR	Maintenance reset	MAIN	Halt for maintenance

Exercise 3

The system can be developed as a set of sub-systems:

- Soaping
- Brushing
- Rinsing
- Drying

Each of this sub-systems has to start when the previous photocell activates its output and has to stop when the next photocell deactivates its output.

Exercise 3

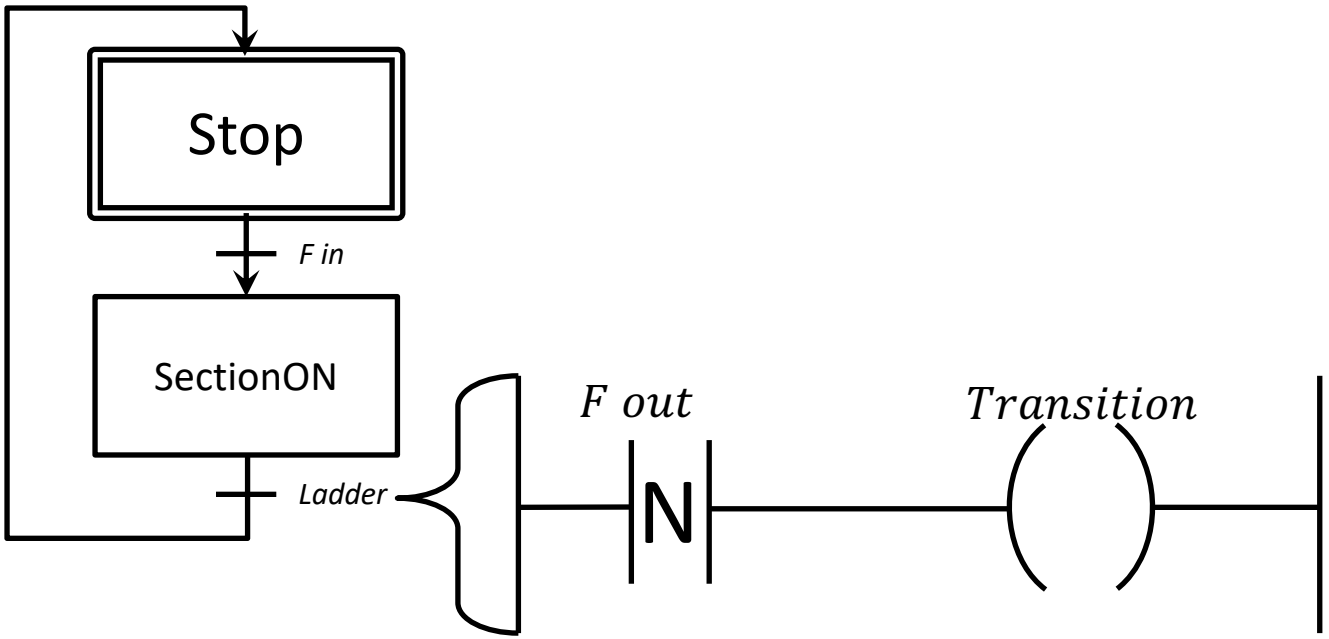
As shown in the previous lesson, this exercise requires a "distributed" solution for each part of the system.

With the SFC language we cannot manage more than a single «execution cycle» with a single code. For this reason we need more than a program: one for each section!

In the program that manages the soaping, we will manage also the semaphore and the mainteinance.

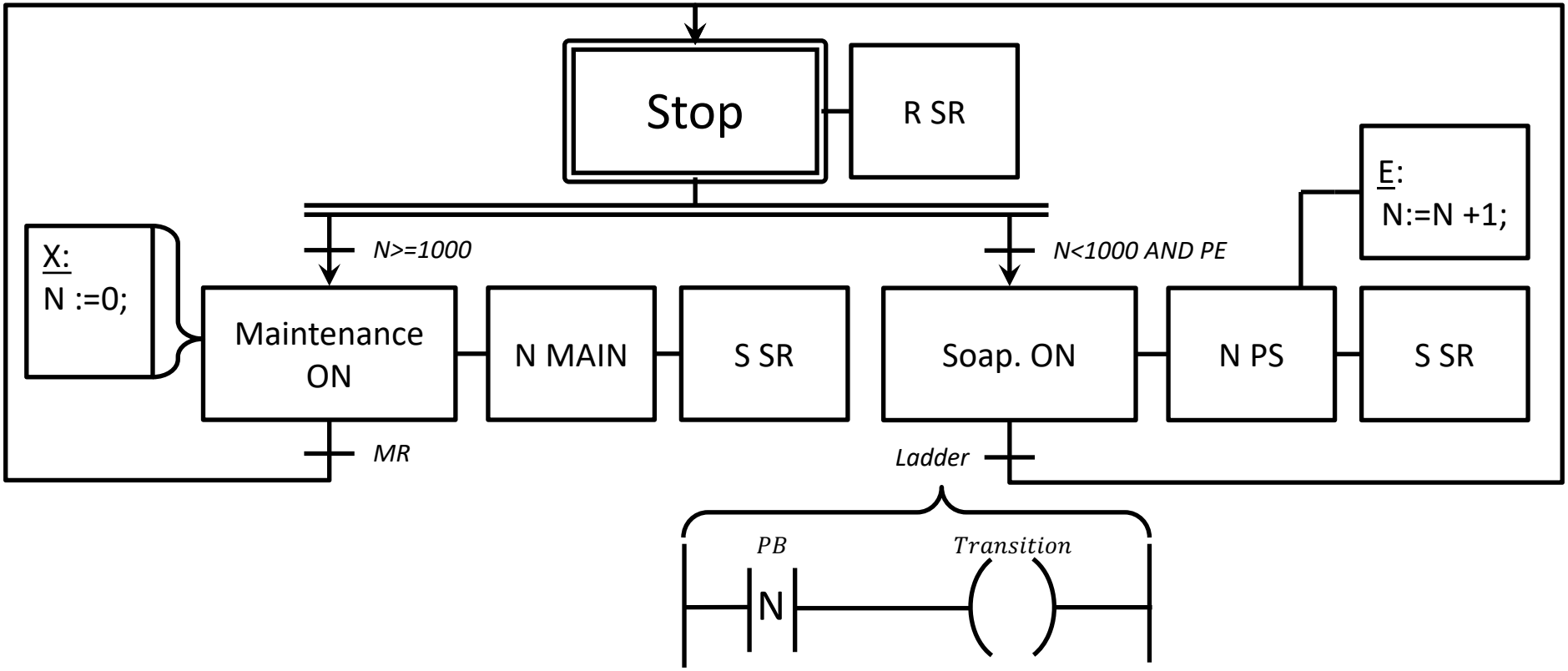
Exercise 3

Each part of the system will have a SFC code like this:



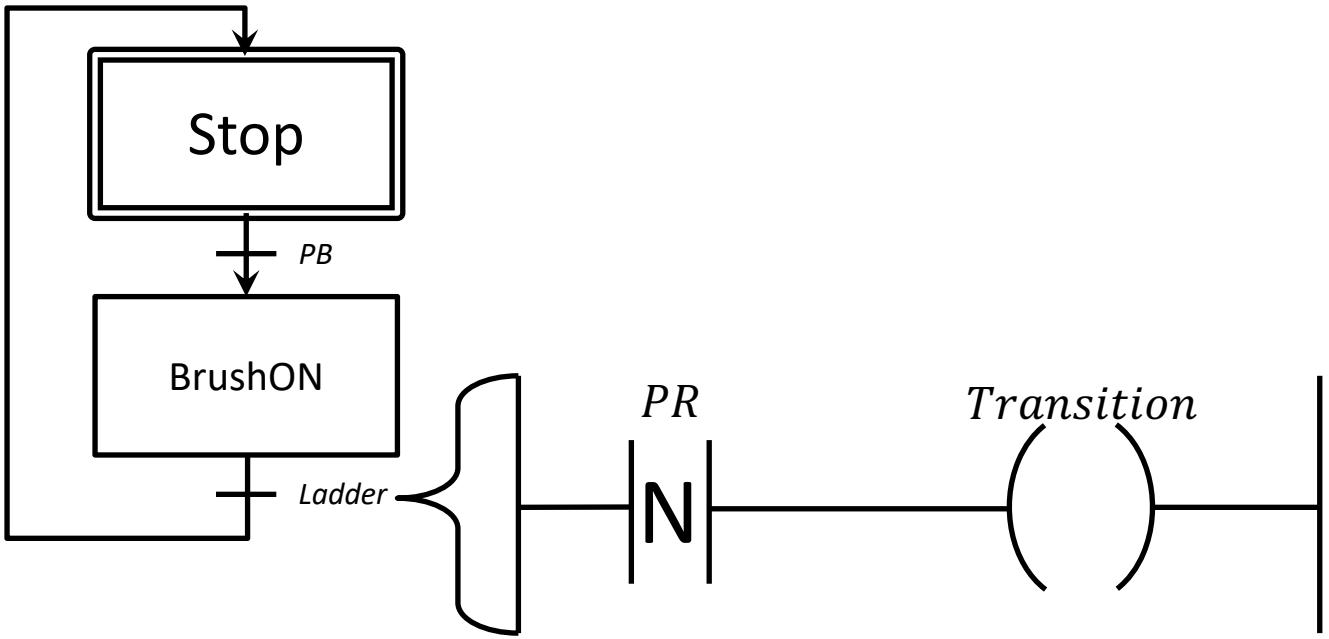
Exercise 3

The software that manage the soaping operation must include the maintenance and the semaphore management.



Exercise 3

Example of SFC code for the Brushing section:



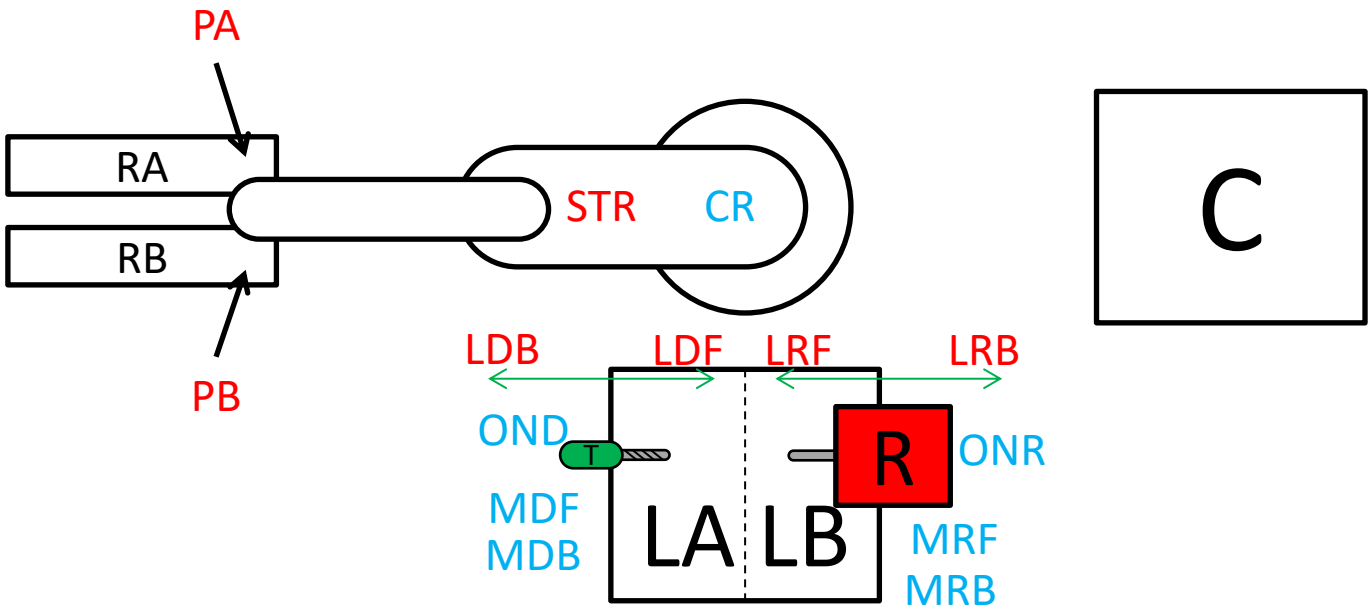
Exercise 4

Consider a system used for automatic drilling and riveting metal sheets.

When the two sheets are available, a robot takes them (one by one) and places them over an assembly mask. After the end of the placing, the pieces are perforated by an automatic drill (5 seconds) and rivetted by a riveter (10 seconds).

At the end of the process, the robot move the produced product in a container.

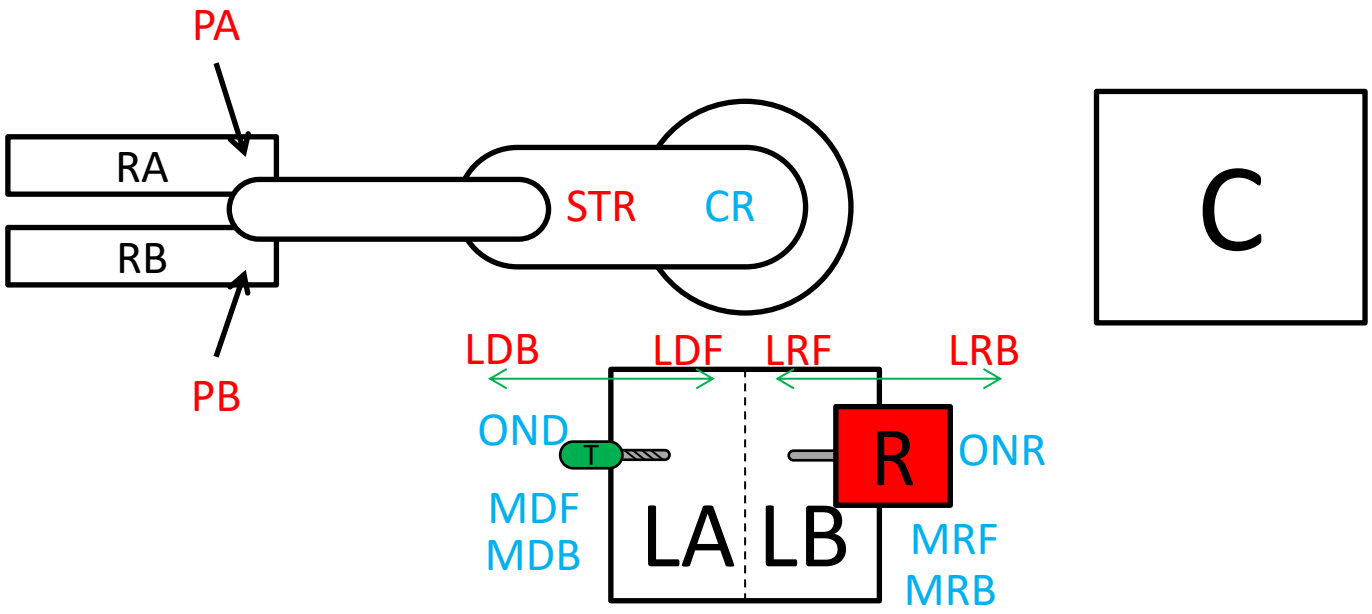
Exercise 4



Input

PA	Piece A available	LDF	Drill forward limit
PB	Piece B available	LRB	Riveter backward limit
STR	Status robot (0=END, 1=EXECUTING)	LRF	Riveter forward limit
LDB	Drill backward limit		

Exercise 4



Output

CR	Robot command (0=STOP, 1=take piece A, 2=take piece B, 3=deposit final product in C)	OND	Drill ON
		MRF	Riveter motor forward
MDF	Drill motor forward	MRB	Riveter motor backward
MDB	Drill motor backward	ONR	Riveter ON

Exercise 4

The steps to be executed to create a finite product are:

1)Wait until PA=1 and PB=1
2)Send command 1 to the robot and wait the end of its execution
3)Send command 2 to the robot and wait the end of its execution
4)Move the drill forward until the forward-limit is reached
5)Operate the drill for 5 seconds
6)Move the drill backward until the backward-limit is reached
7)Move the riveter forward until the forward-limit is reached
8)Operate the riveter for 10 seconds
9)Move the riveter backward until the backward-limit is reached
10)Send command 3 to the robot and wait the end of its execution
11)Send command 0 to the robot

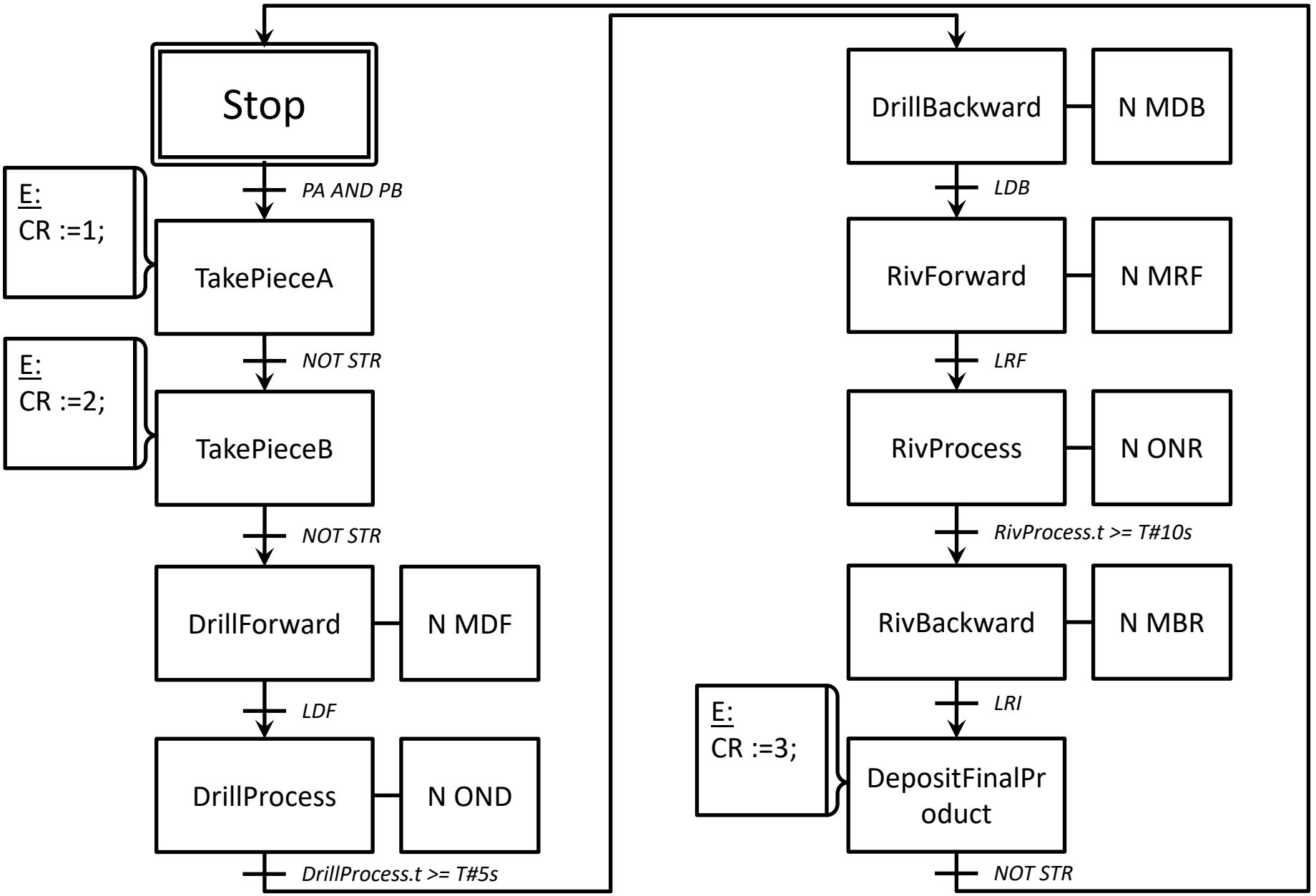
Exercise 4

This exercise, that in the previous lesson was solved using ladder and fake-states, is more adaptable to the logic on which SFC is based.

We have a list of states that have to be sequentially executed, with some simple transitions between them.

The solution is really easy, because we only have to insert all the states that are reported into the previous slide.

Exercise 4


Stop
PA AND PB
E:
CR :=1;
TakePieceA
NOT STR
E:
CR :=2;
TakePieceB
NOT STR
DrillForward
N MDF
LDF
DrillProcess
N OND
DrillProcess.t >= T#5s
DrillBackward
N MDB
LDB
RivForward
N MRF
LRF
RivProcess
N ONR
RivProcess.t >= T#10s
RivBackward
N MBR
LRI
E:
CR :=3;
DepositFinalPr
oduct
NOT STR

Conclusions

Conclusions about SFC

It is a language suitable for the control of systems that can be represented with a finite set of states.

It is not suitable when we want to decentralize the control (all the execution takes place in a single instance).