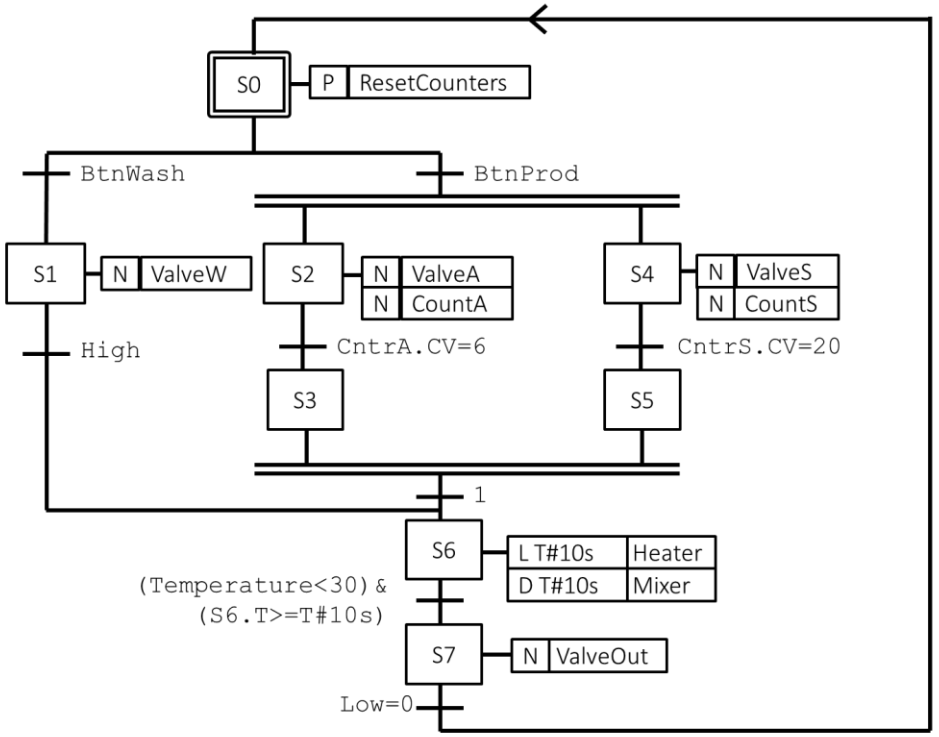
# Sequential Function Charts

Industrial control

KOVÁCS Gábor
gkovacs@iit.bme.hu

# Sequential Function Charts (SFC)

- Aim: decomposition of a complex program to smaller units and description of the program flow
- Origin
  - flow charts
  - Petri nets
  - Grafcet

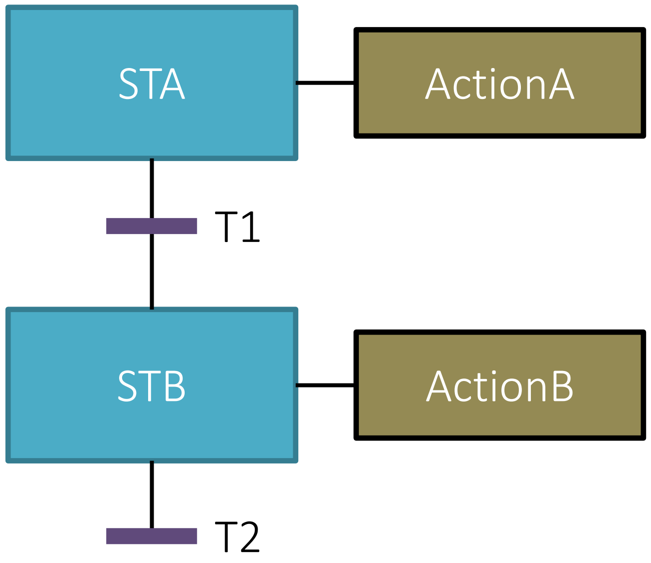


## Implementing POUs

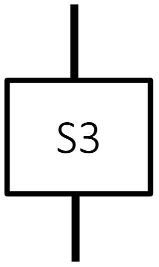
- SFC describes the program flow
- It needs to store the current state: **functions can not be implemented in SFC**
- There are development environments which only allow the implementation of programs in SFC

# SFC elements

- SFC – bipartite graph
  - nodes:
    - steps ≈ states
    - transitions : Boolean expressions
  - directed edges (from top to bottom by default)
- Actions associated to steps

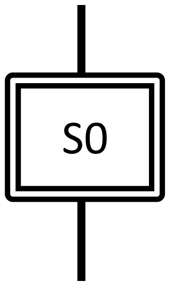


# Steps



- Box and step identifier: `StepName`
- Step flag: `StepName.X`
  - Boolean variable, true if the given step is active
- Step time: `StepName.T`
  - duration-type variable, contains the time elapsed since the last activation of the step
  - value of step time is frozen when the step is deactivated, reset to `T#0s` when the step is activated again
- Step flag and step time are read-only variables

# Initial step



- Notation: double line
- Any arbitrary identifier might be used
- Initial value of its step flag is `TRUE`
- Each network must contain one and only one initial step

# Textual definition of steps

- Step

```
STEP StepName
   (* step body *)
END_STEP
```

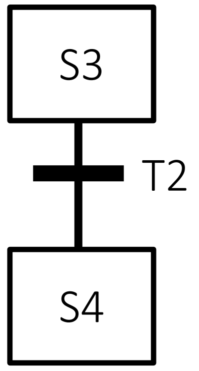
- Initial step

```
INITIAL_STEP StepName
   (* step body *)
END_STEP
```

- Textual description is part of the standard but generally not supported by the development environments

# Transitions

- Horizontal bar (line) crossing a vertical connection between two steps
- Each transition is assigned one single condition
  - condition is a Boolean expression
  - the transition might fire if the condition evaluates to true
  - unconditional transitions can be implemented by assigning a constant `TRUE` (`1`) condition



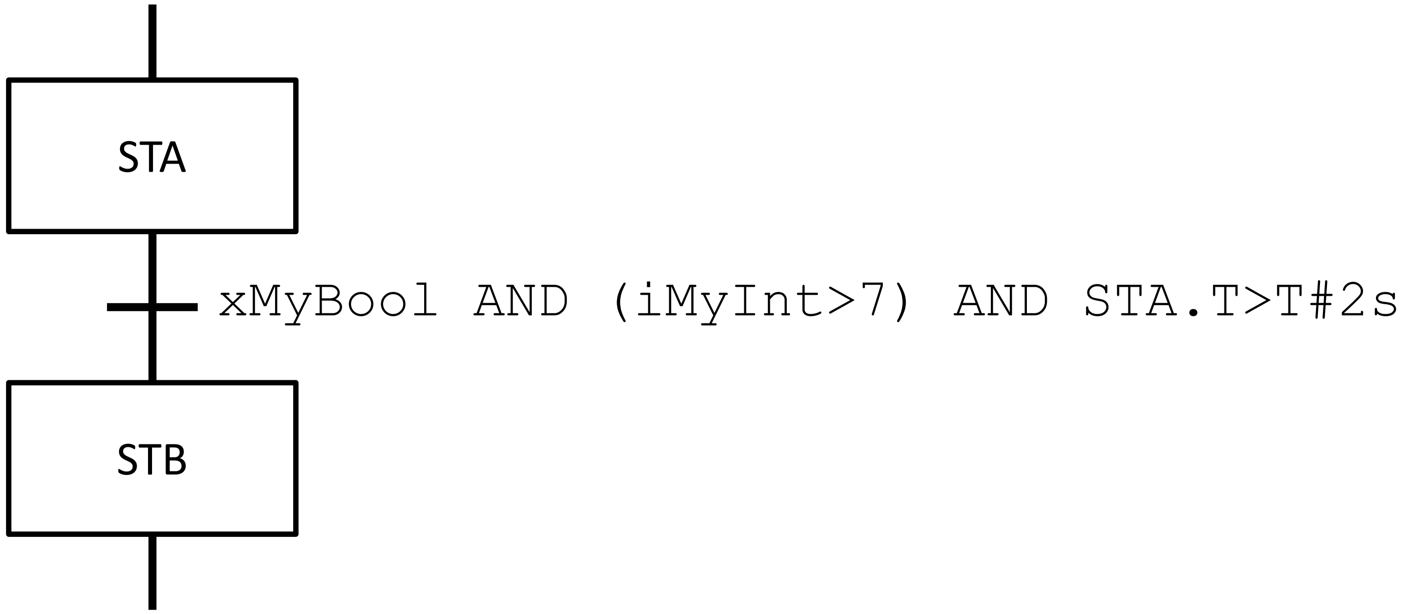
# Textual definition of transitions

```
TRANSITION TranName FROM Step1 TO Step2
     (* body *)
END_TRANSITION
```

- Textual description is part of the standard but generally not supported by most development environments in this form

# Definition of transition conditions

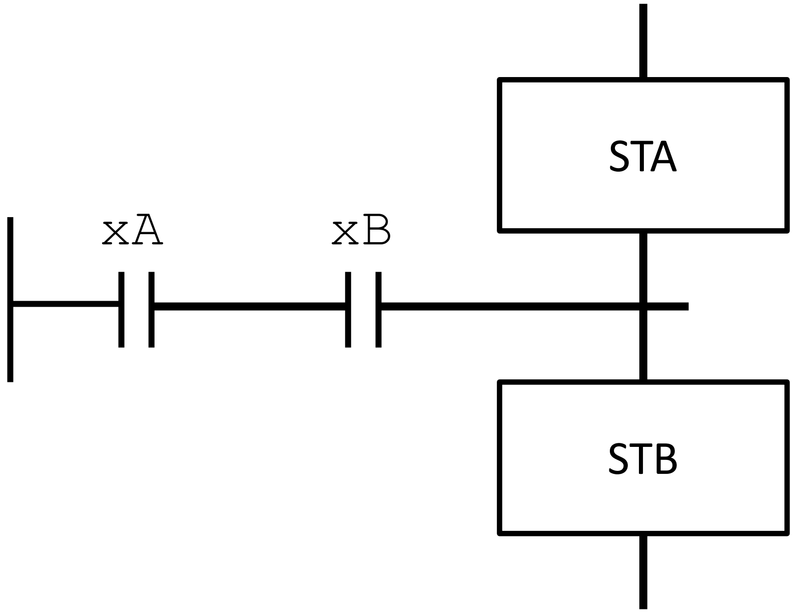
- Direct definition using an ST expression

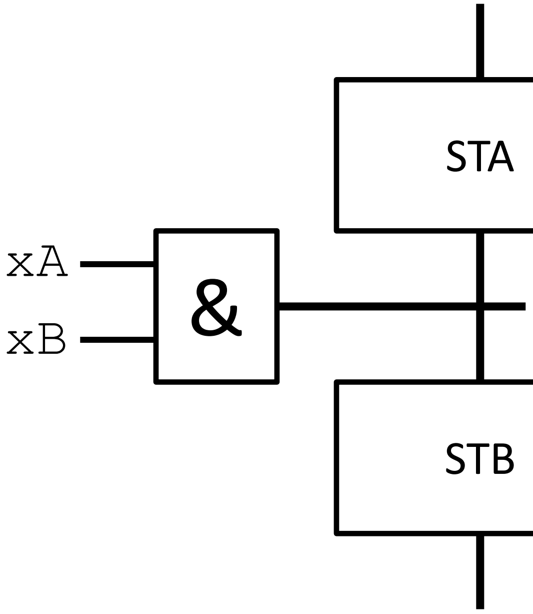


CODESYS allows direct definition of transitions as ST expressions only

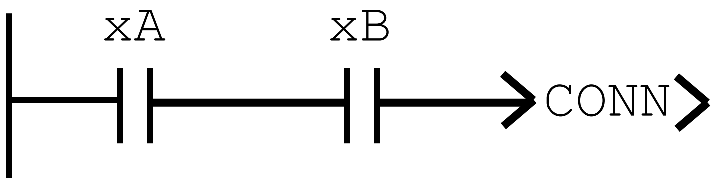
# Definition of transition conditions

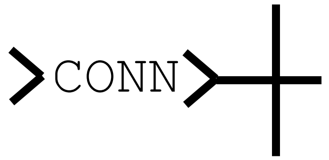
- Direct definition in LD or FBD language



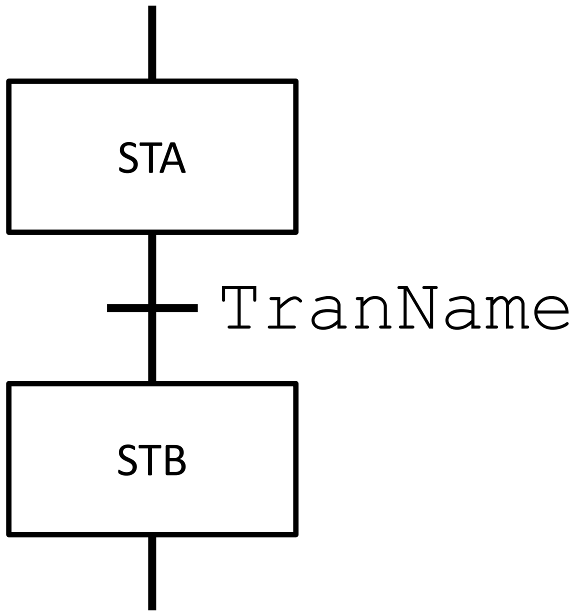


- Connectors might be used





# Indirect definition of transition conditions



- Condition can be defined in a `TRANSITION` block associated to the transition
- The condition might be associated to multiple transitions

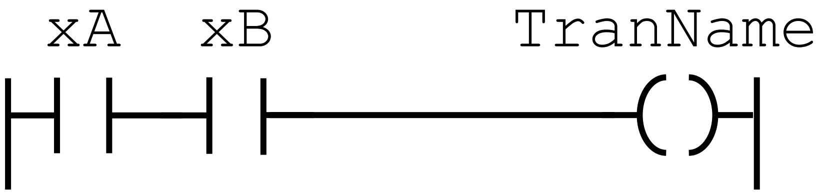
```
TRANSITION TranName:
(* LD, IL, FB, ST *)
END_TRANSITION
```

# Indirect definition of transition conditions

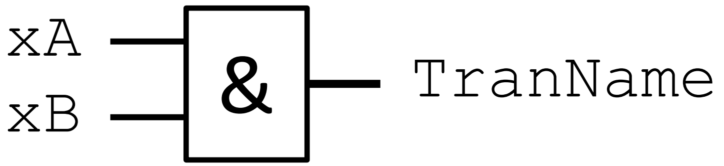
- ST: assignment of an expression (left-hand side is omitted or an implicitly declared variable with the same identifier as the `TRANSITION` block)

```
TranName:=xA AND xB;
```

- LD



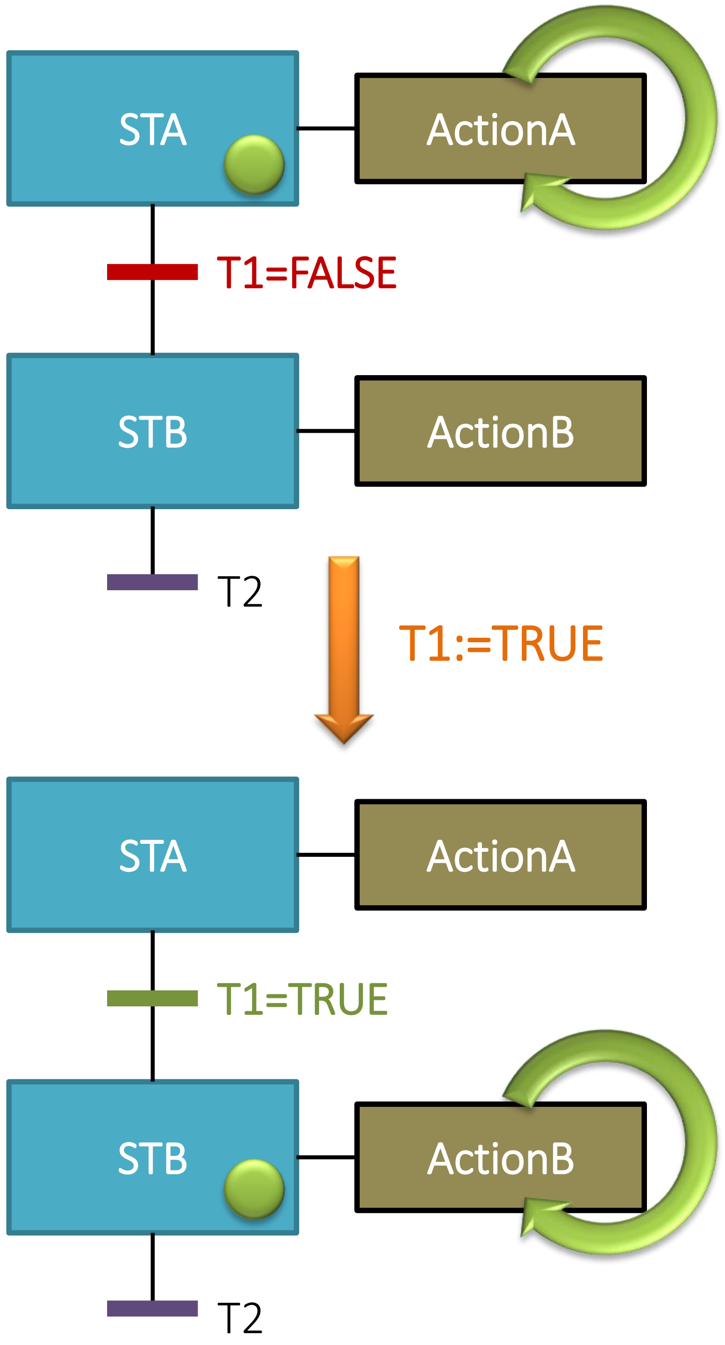
- FBD



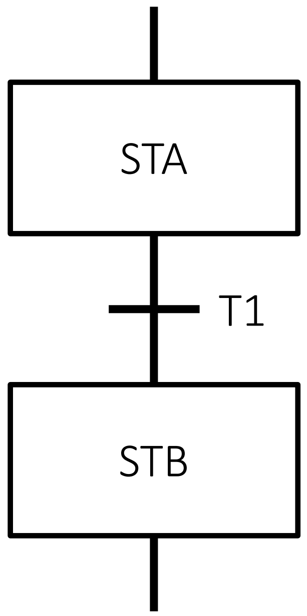
Variable with the same identifier as the `TRANSITION` block

# Token game

- Active step is denoted by the presence of a token
- There might be multiple active steps within a network
- The token is placed to an other step if the condition of a transition between them evaluates to true
- Actions associated to the active step are executed cyclically by default

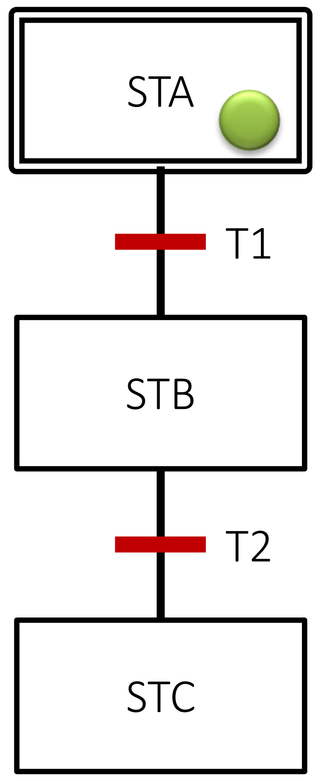


# Simple sequence

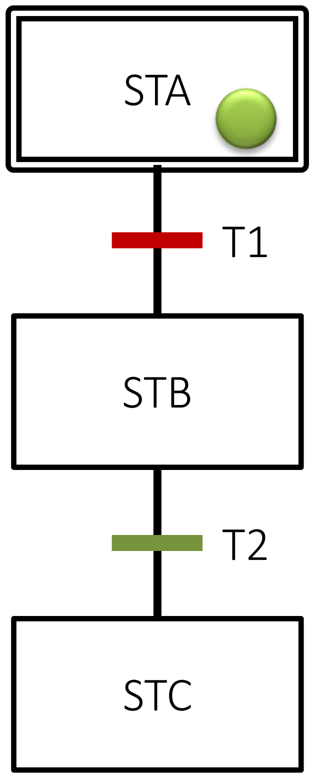


- If STA is active and T1 evaluates to true
  - STA is deactivated
  - STB is activated

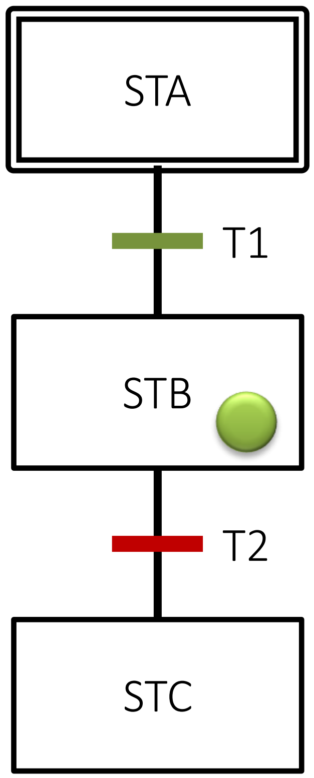
## Simple sequence



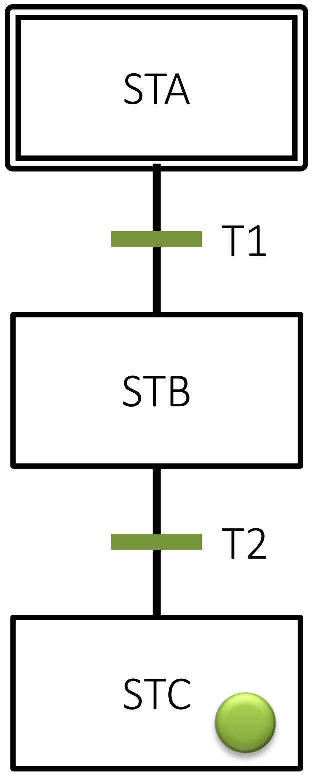
# Simple sequence



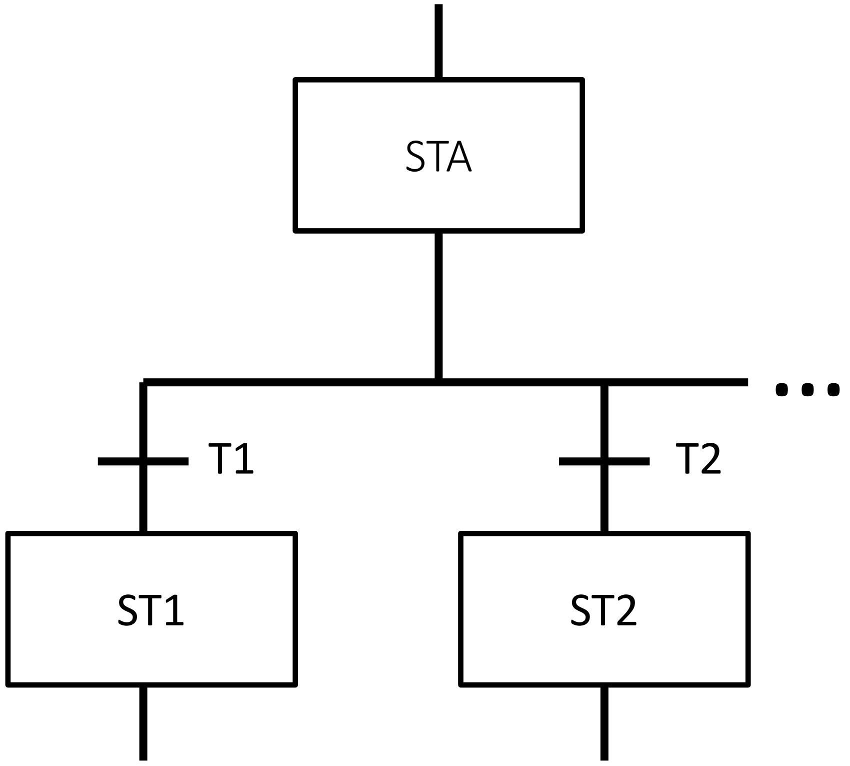
# Simple sequence



# Simple sequence



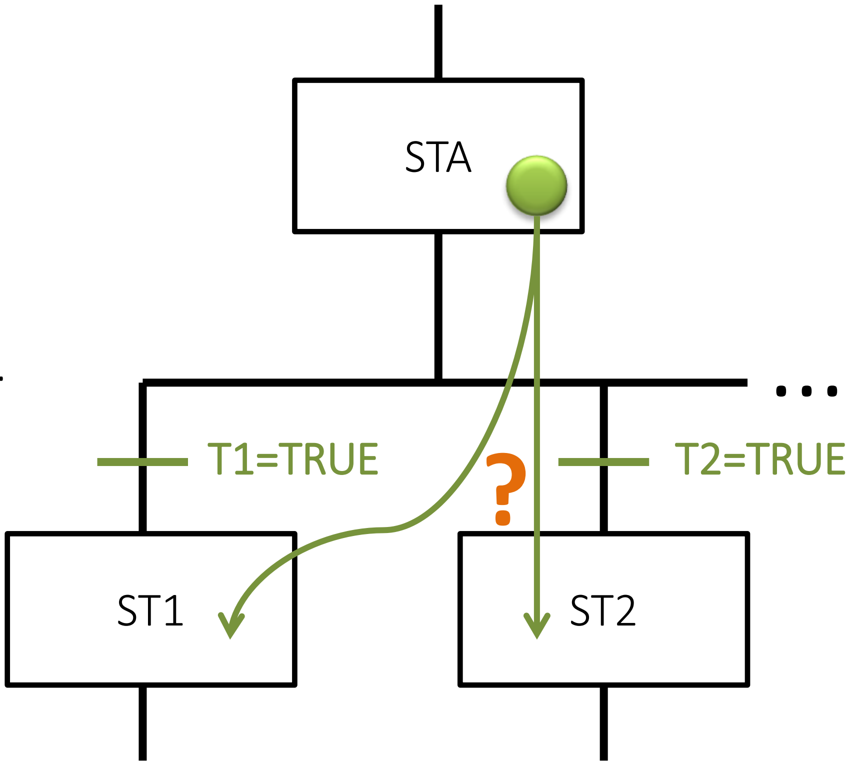
# Sequence selection (divergent paths)



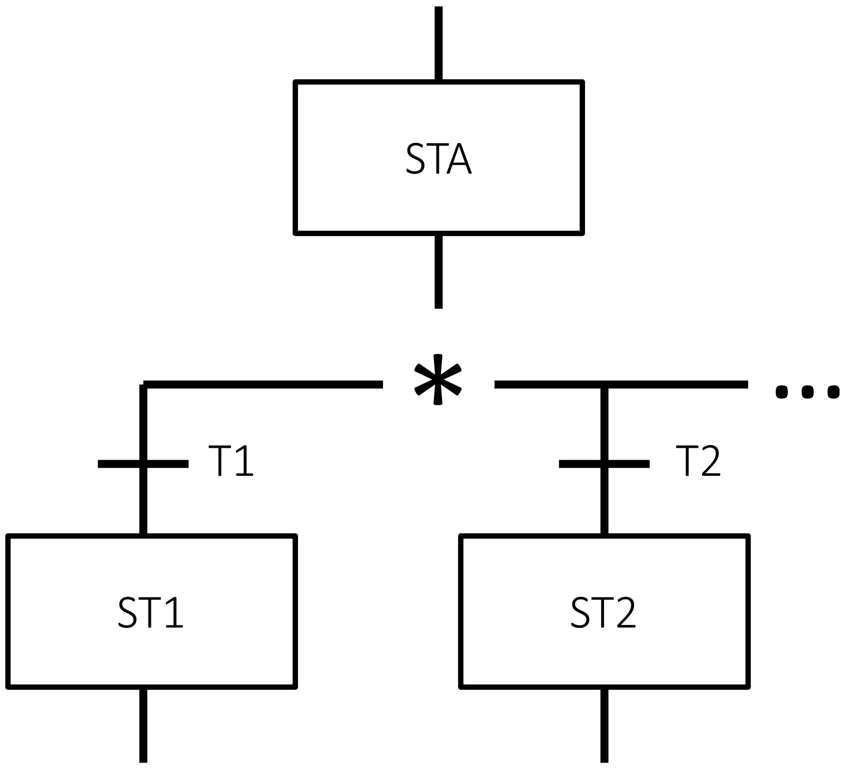
- A step is followed by multiple transitions
- If STA is active, the first transition T*i* which evaluates to true deactivates STA and activates ST*i*-t
- Only one of the branches becomes active

# Evaluation of divergence of sequence selection

- According to the standard
  - evaluation according to left-to-right priority
  - evaluation according to user-defined priority
  - evaluation with mutual exclusion
- In practice: **evaluation according to left-to-right priority**

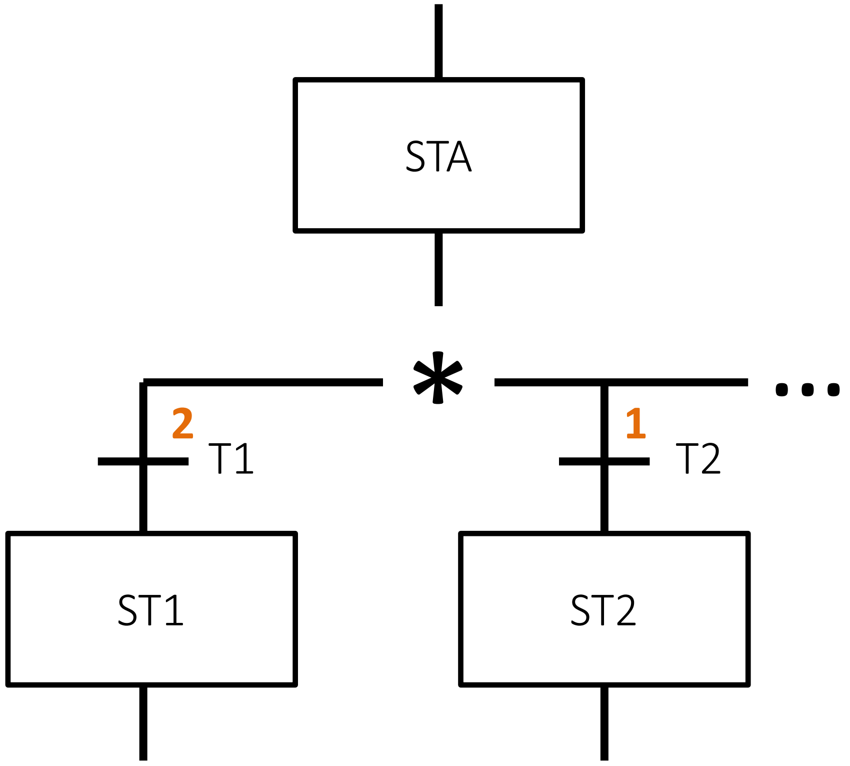


# Left-to-right priority



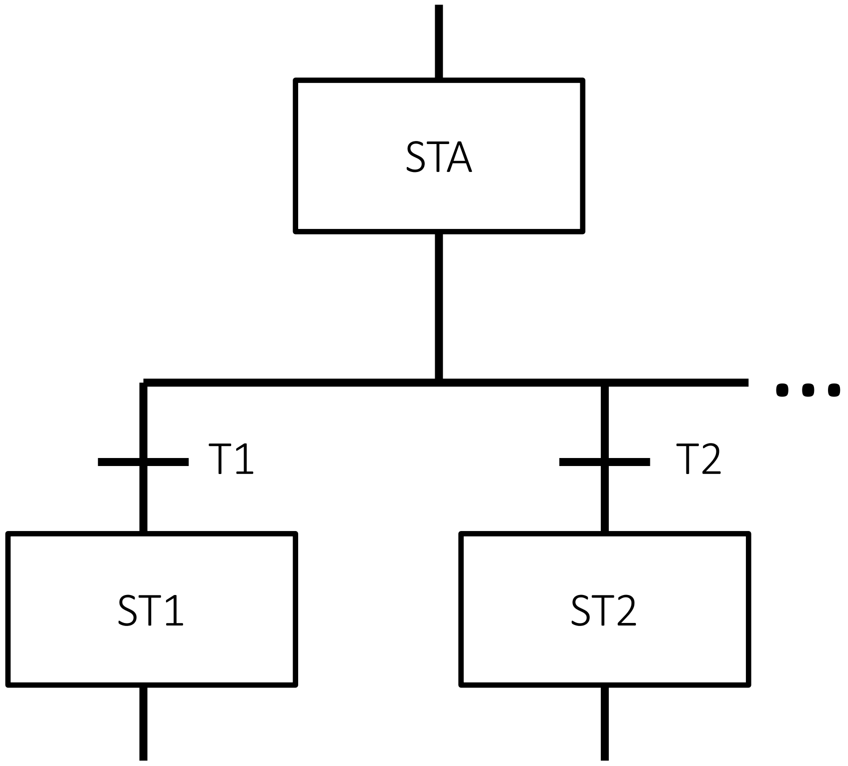
- While STA is active, transitions are evaluated from left to right
- The **first** transition T*i* which evaluates to true deactivates STA and activates STi

# User-defined priority



- Transitions are evaluated in an order defined by the user
- Lower value corresponds to higher priority

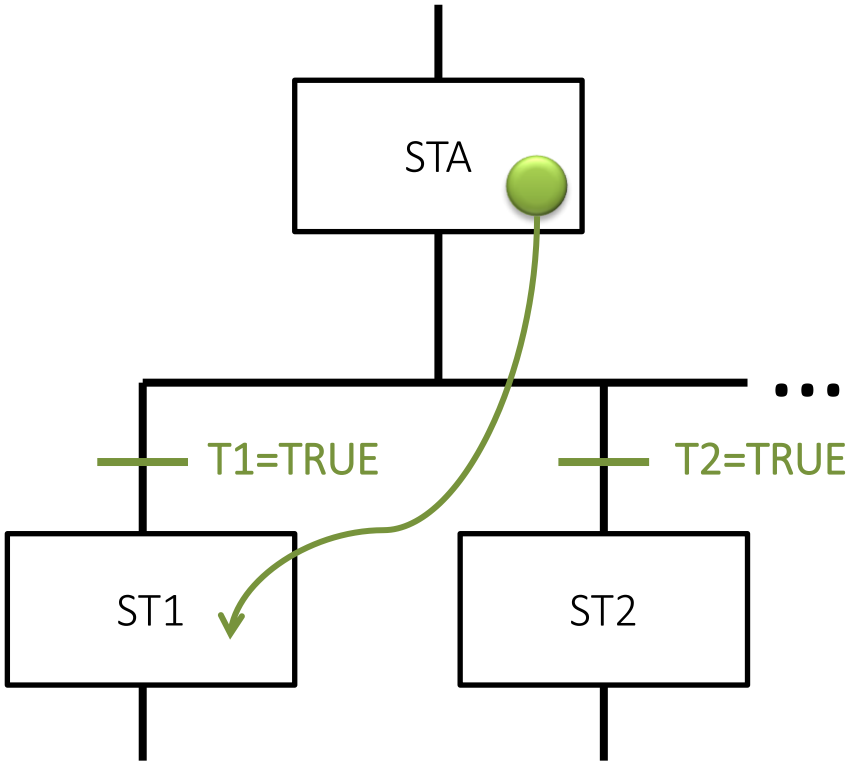
# Mutual exclusion



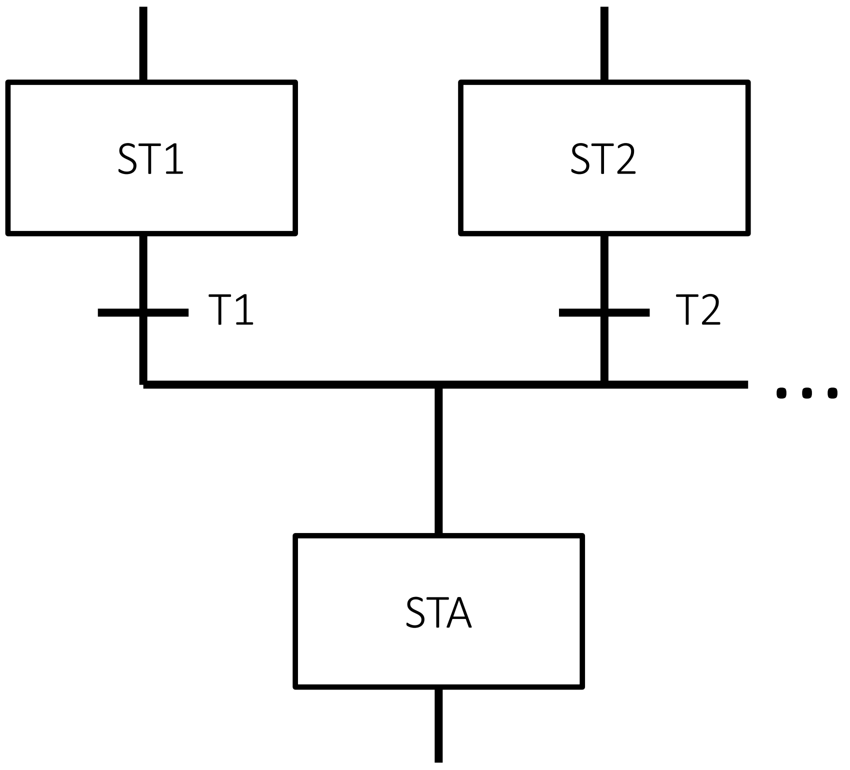
- Evaluation order of transitions is not defined
- The user must assure that transition conditions are mutually exclusives (i.e. at most one of them might evaluate to true)

# Divergence of sequence selection - note

- Most development environments support left-to-right priority only
- In that case the asterisk (*) notation is omitted
- We assume left-to-right priority and omit the asterisk in the followings

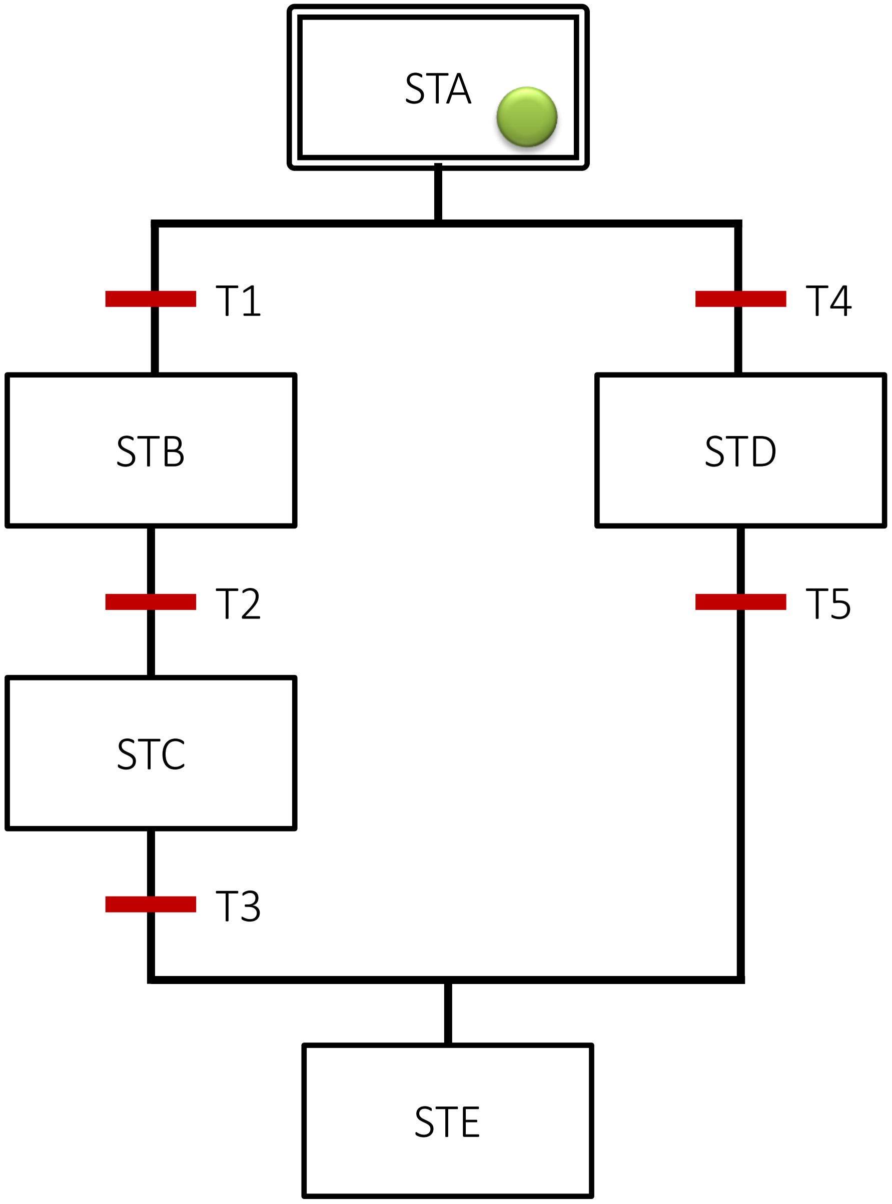


# Convergence of sequence selection

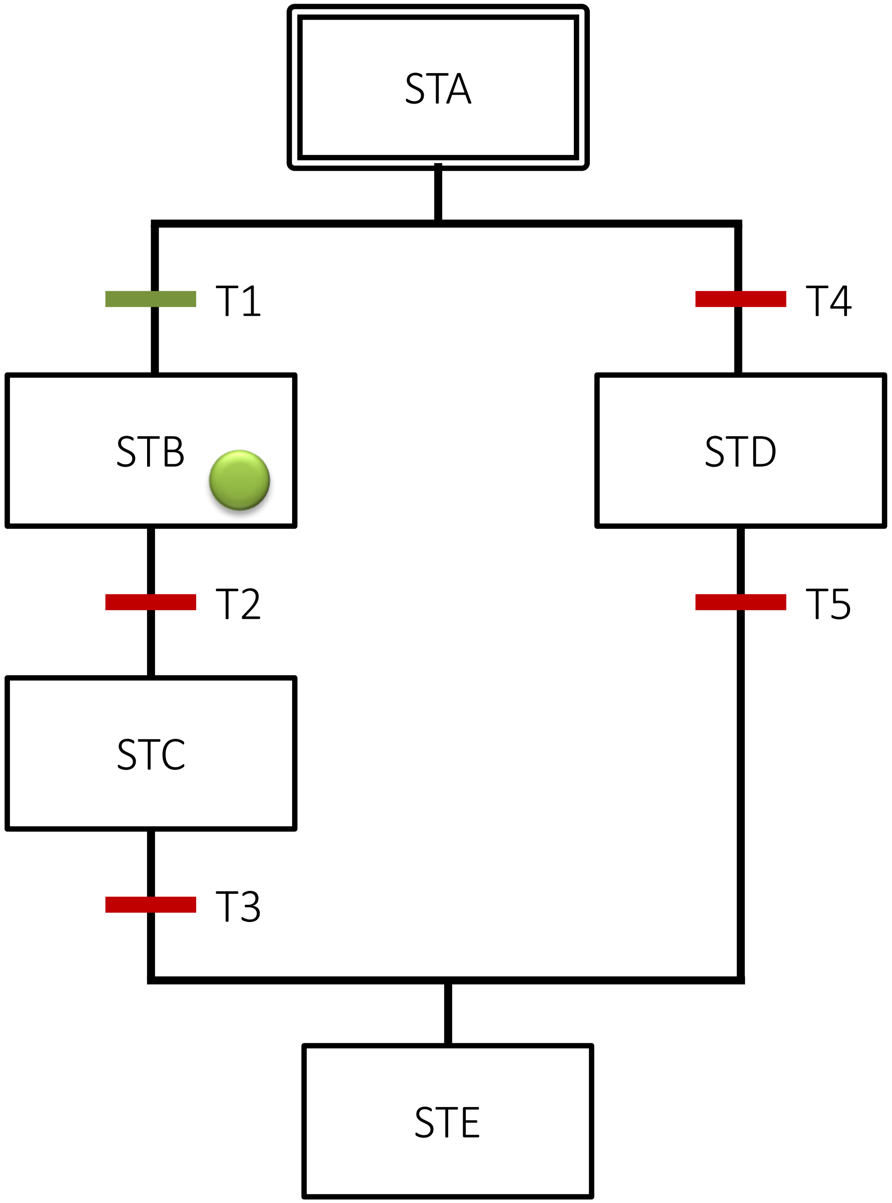


- STA is activated if STi is active and Ti evaluates to true
- STi is deactivated at the same time

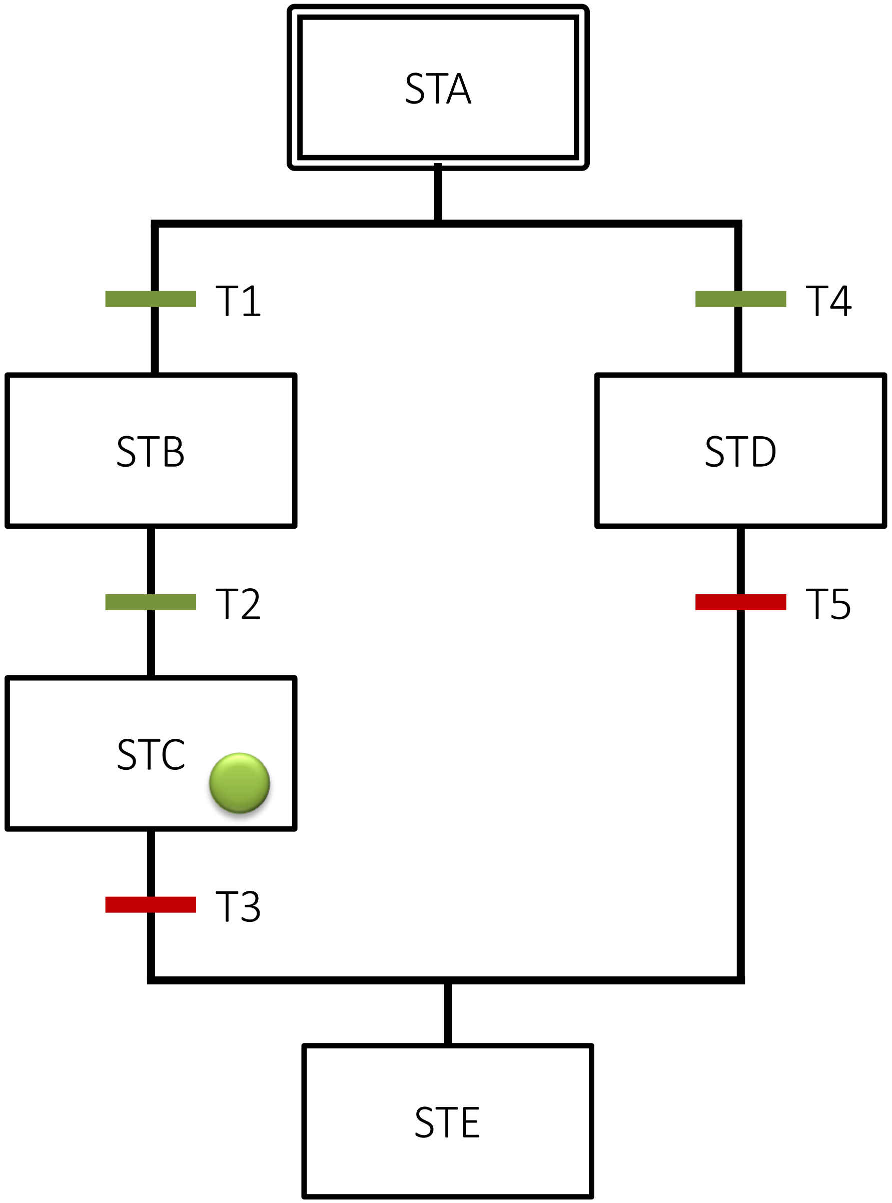
# Sequence Selection



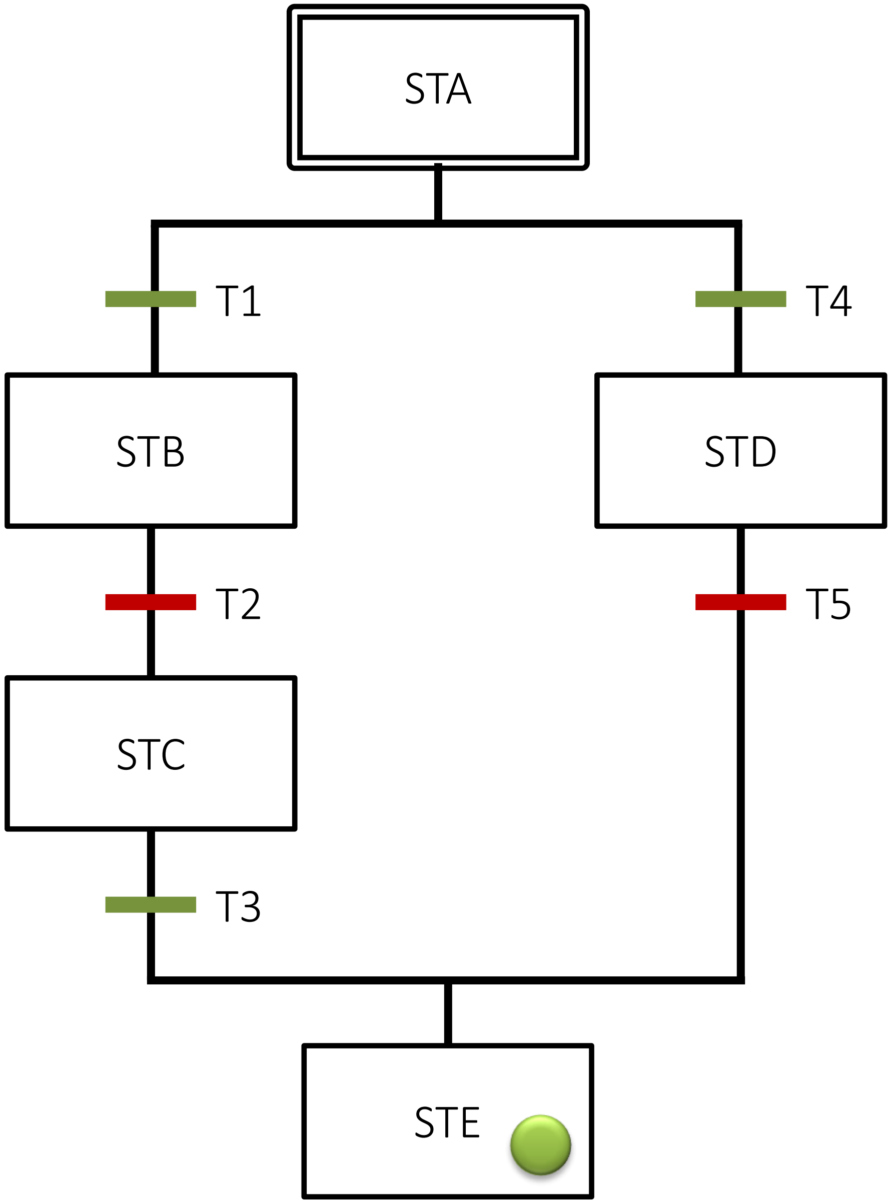
# Sequence Selection



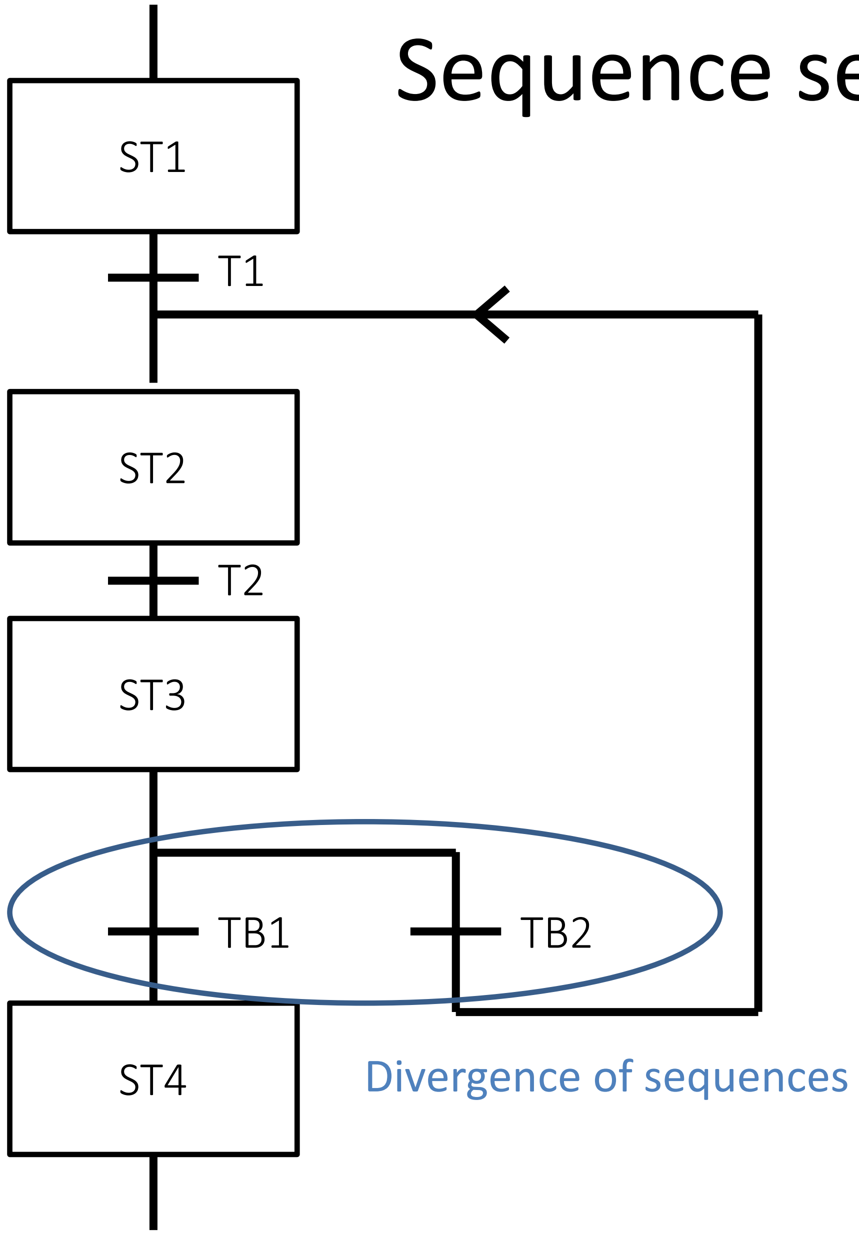
# Sequence Selection



# Sequence Selection



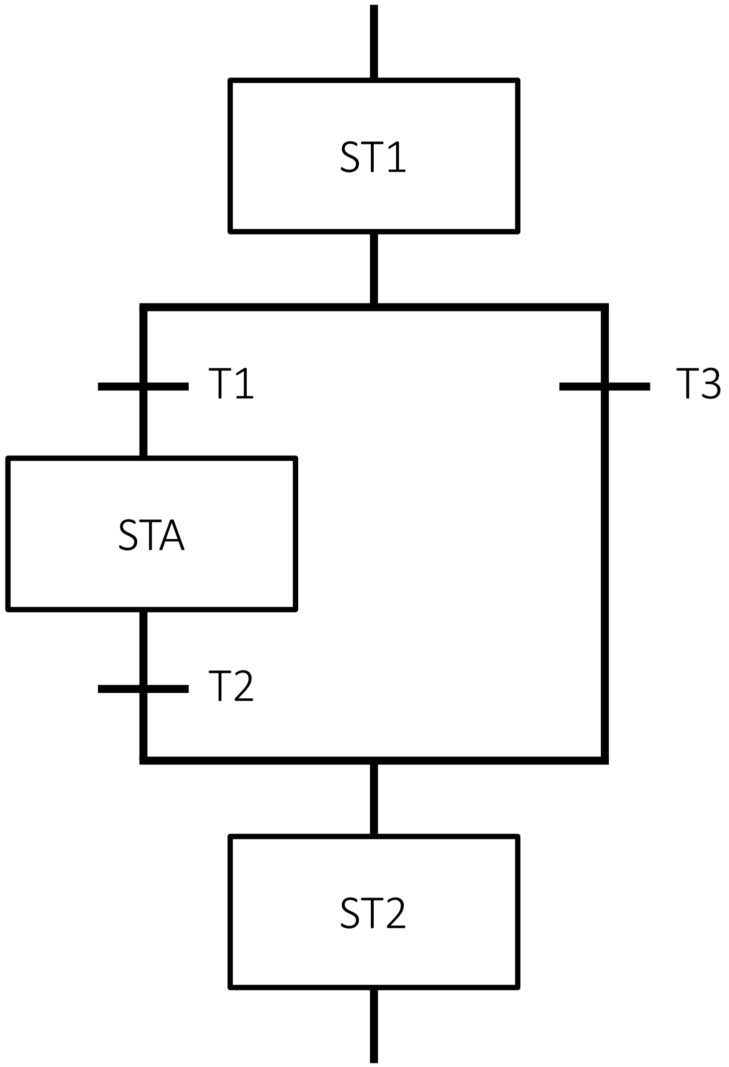
# Sequence selection: sequence loop



- Sequence returning to a predecessor step
- Any supported priority scheme might be used

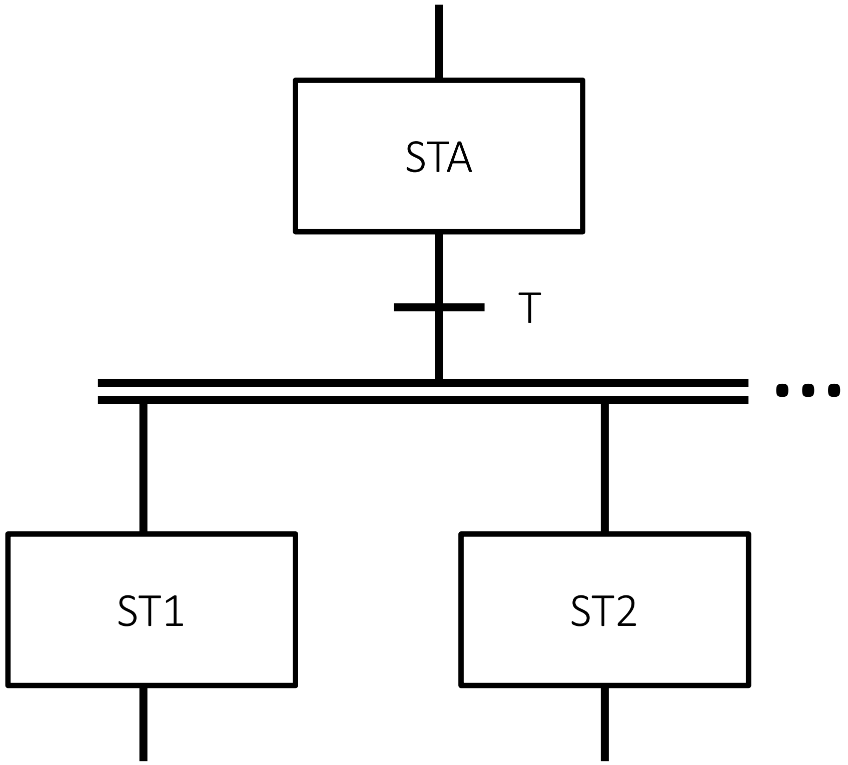
- Defines an iteration
    - TB1: iteration is terminated if condition is met
    - TB2: iteration continues if condition is met

# Sequence selection – sequence skip



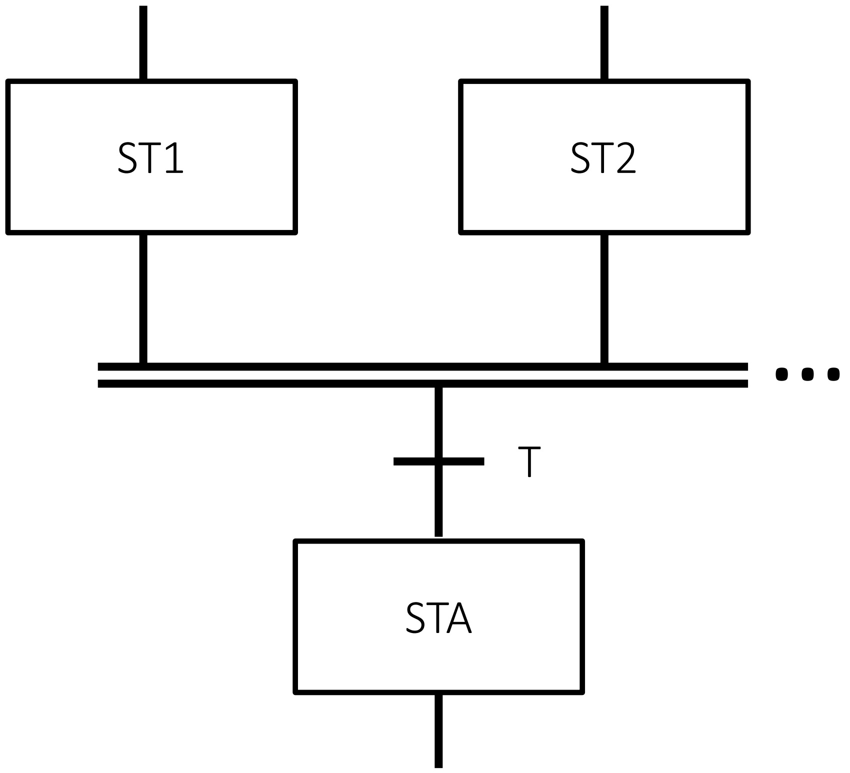
- Step STA is skipped if T3 evaluates to true (while T1 evaluates to false)

## Divergence of simultaneous sequences



- If STA is active and T evaluates to true then STA is deactivated and **every step** STi is activated
- The token is „cloned”

# Convergence of simultaneous sequences

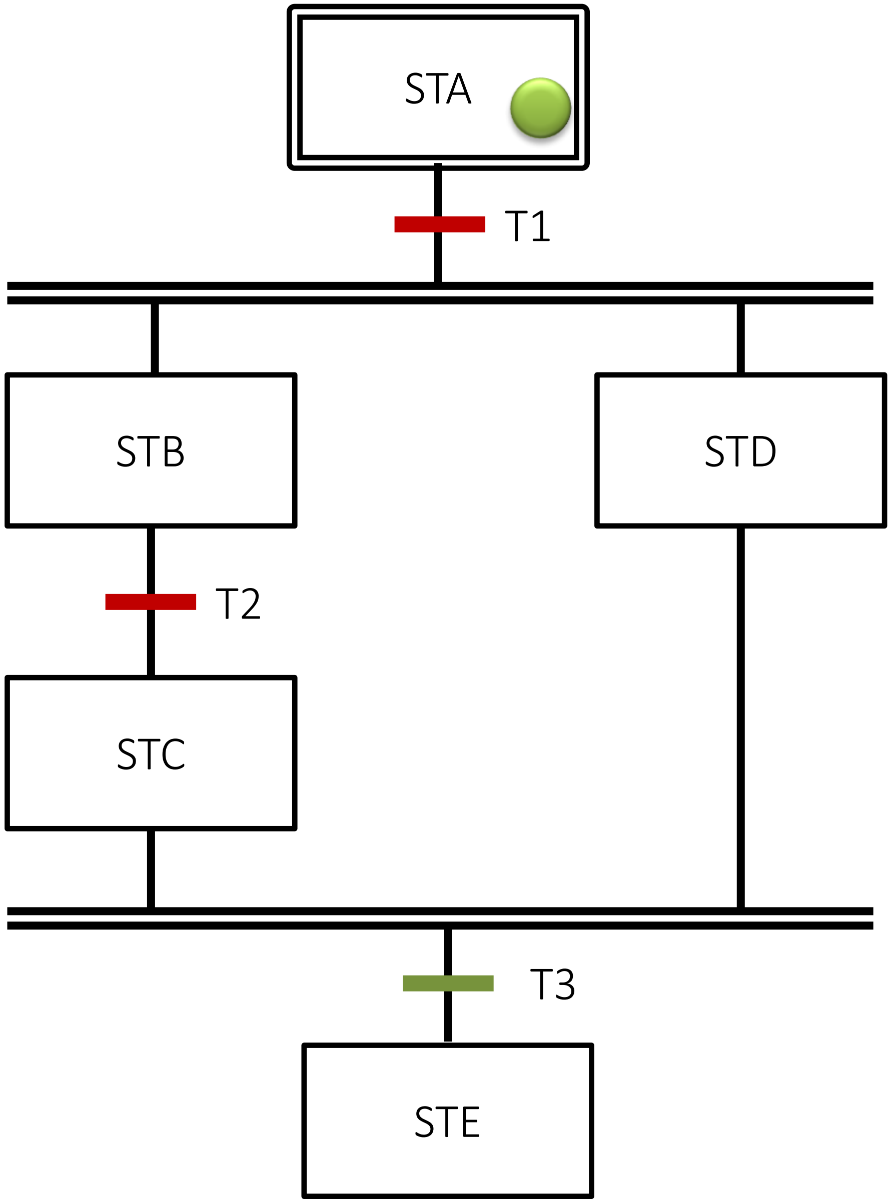


- STA is activated if
  - T evaluates to true

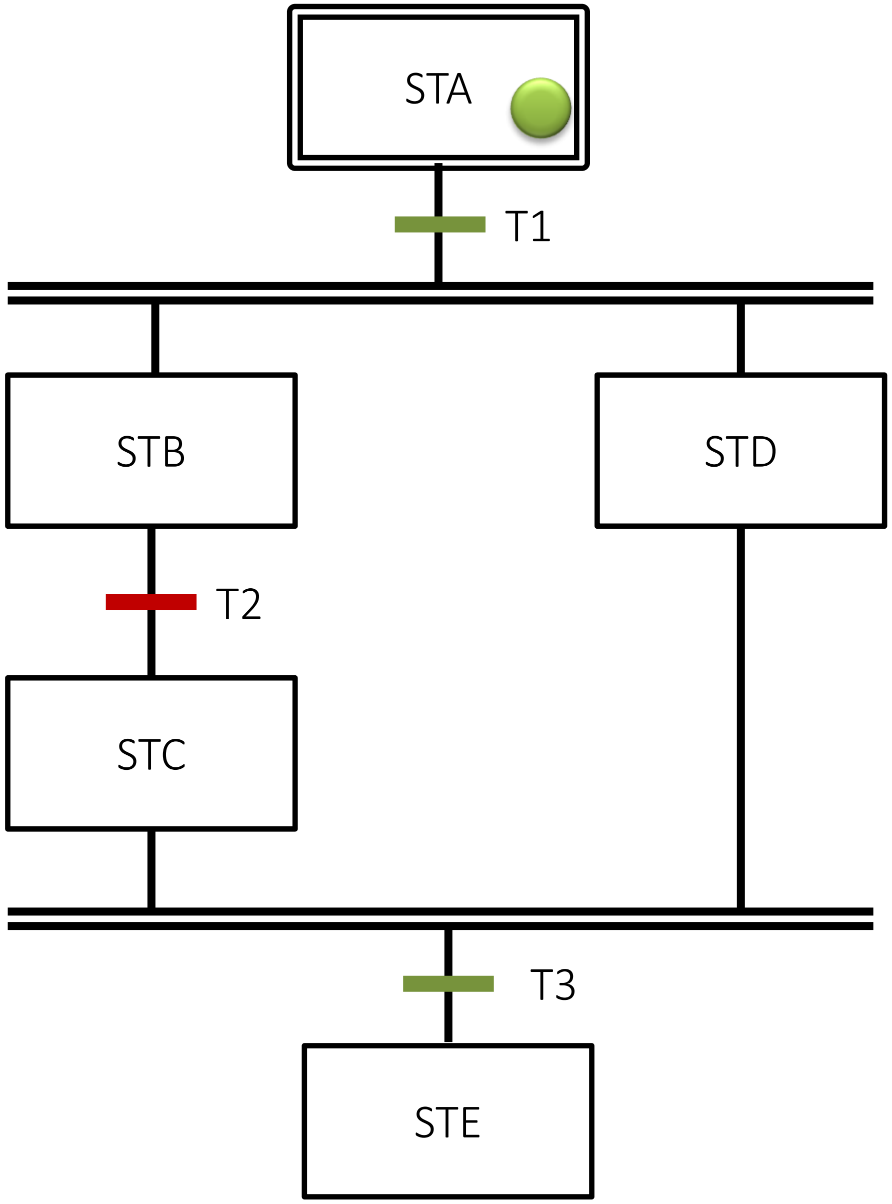
    **AND**
  - each and every step ST1…STn is active

- Then the „cloned” tokens are united

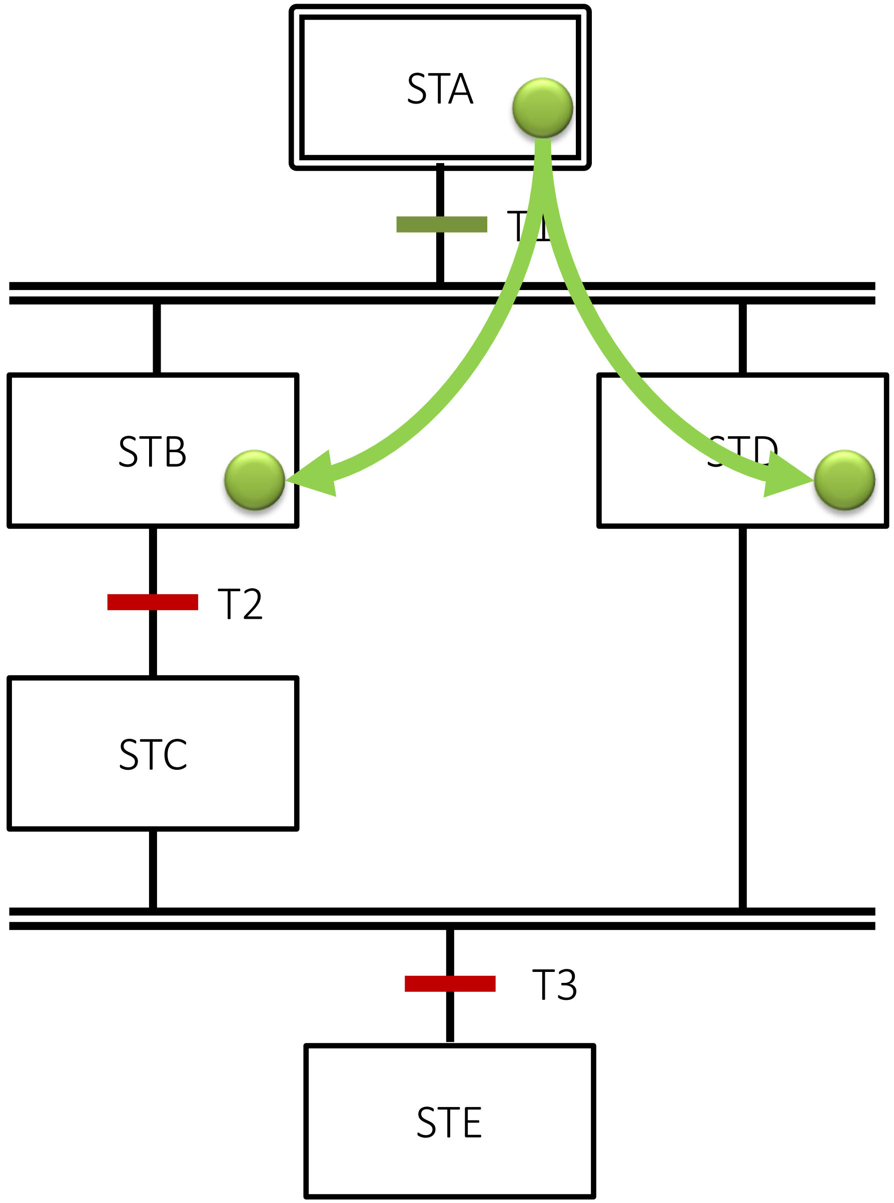
# Simultaneous sequences



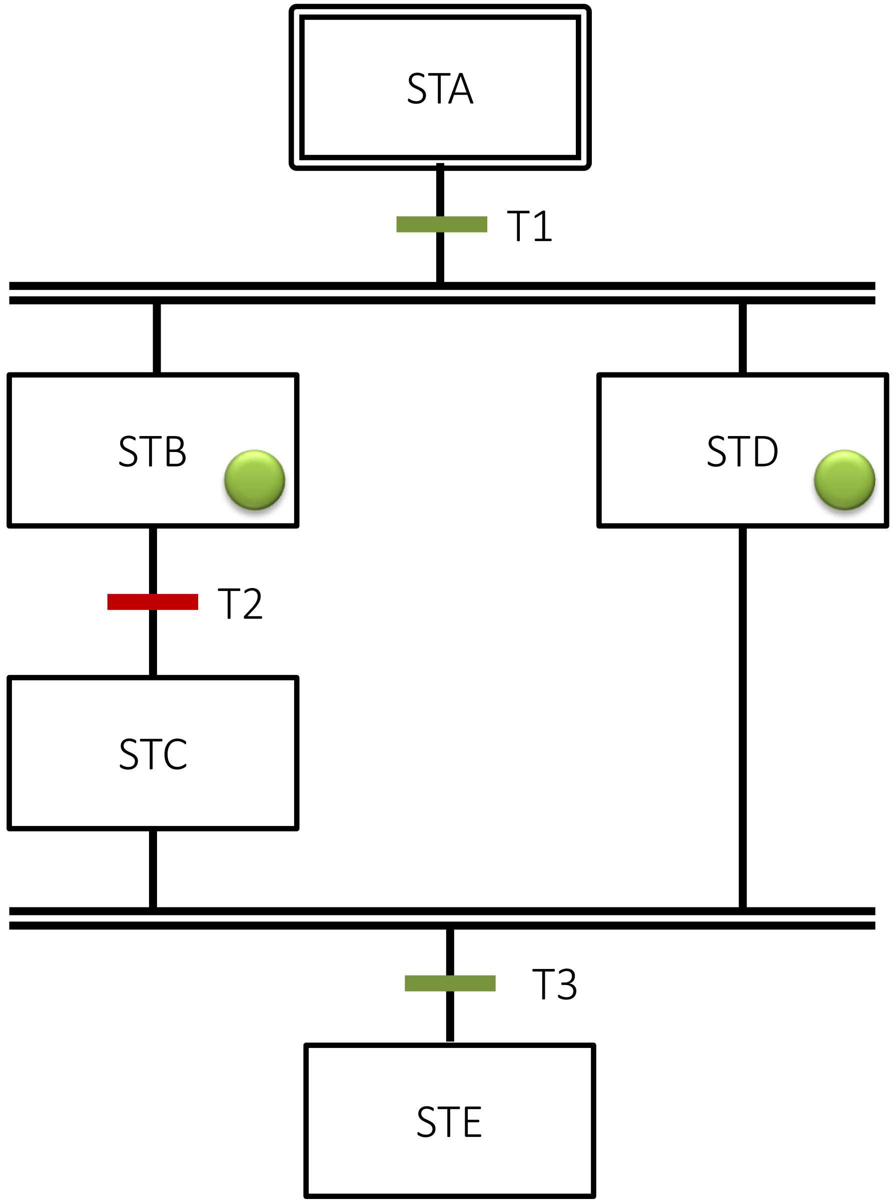
# Simultaneous sequences



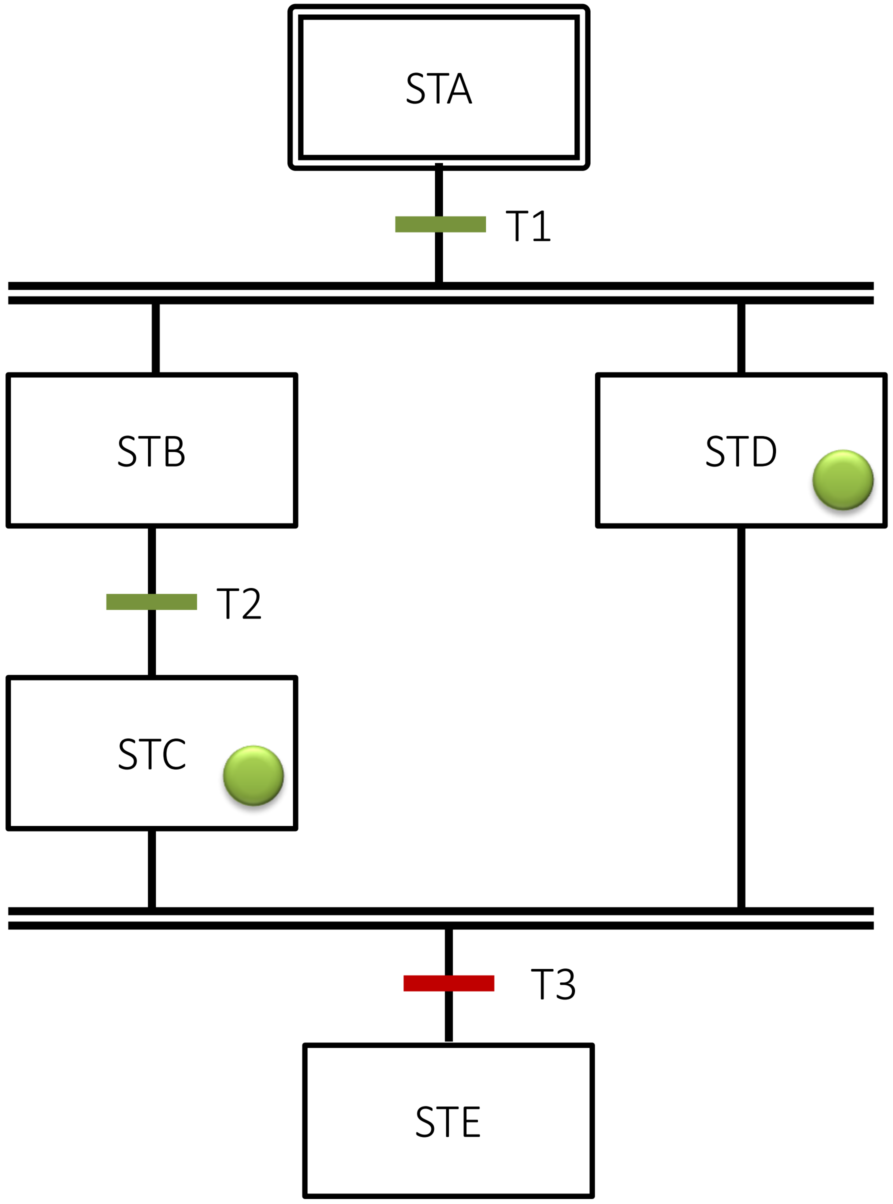
# Simultaneous sequences



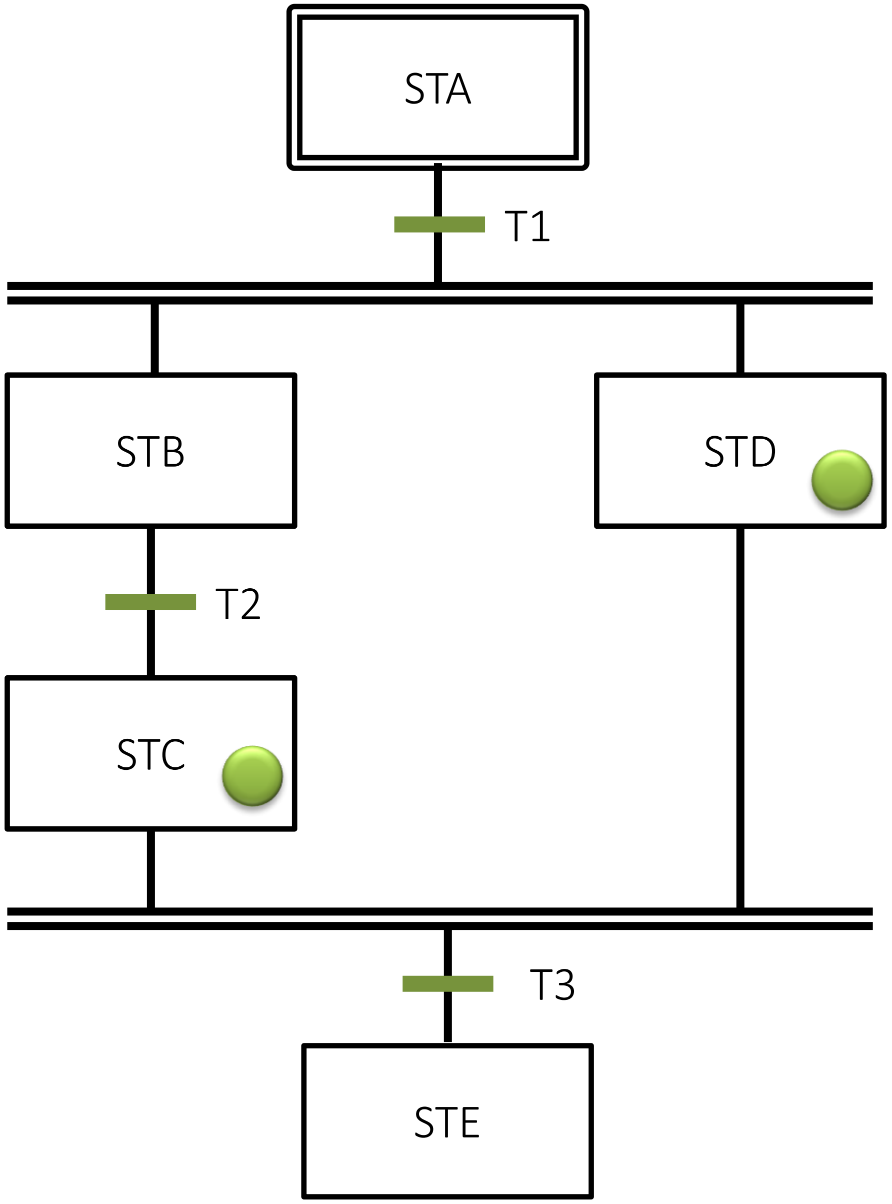
# Simultaneous sequences



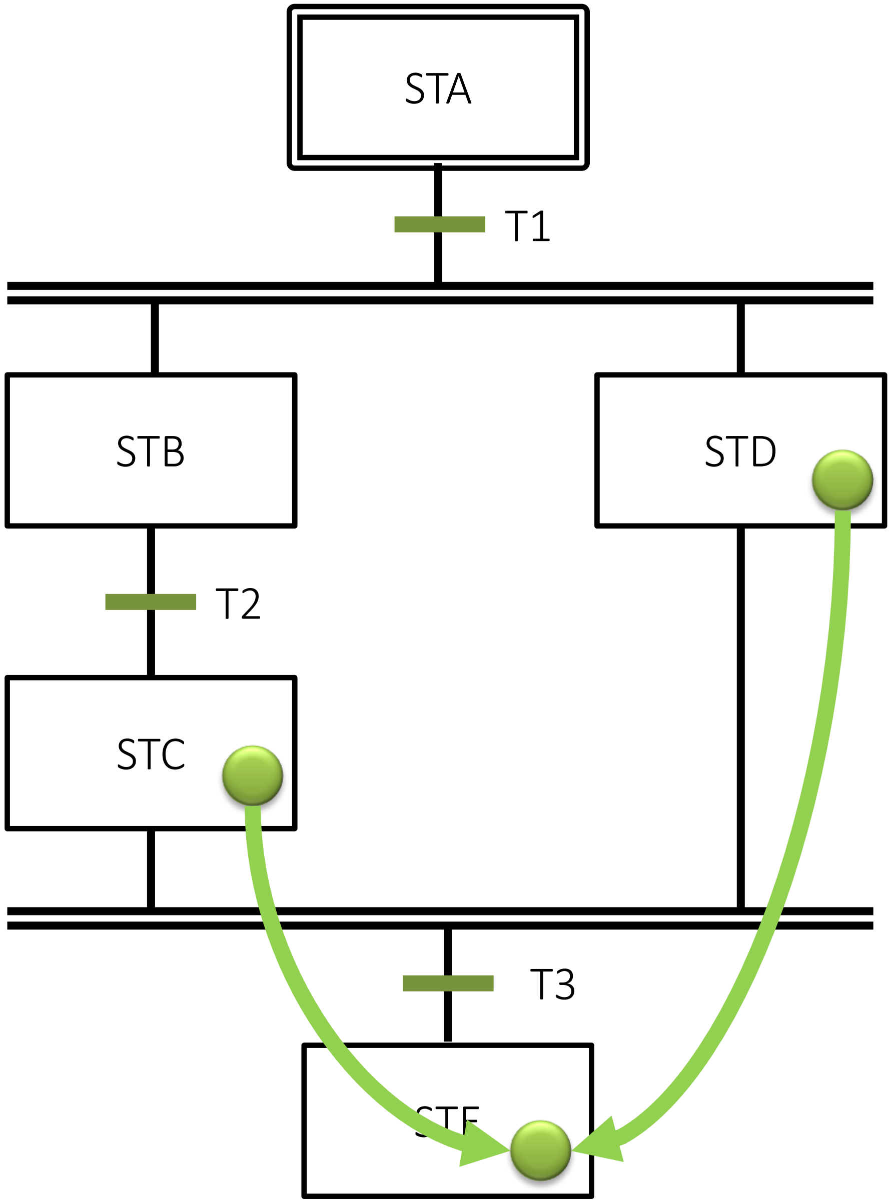
# Simultaneous sequences



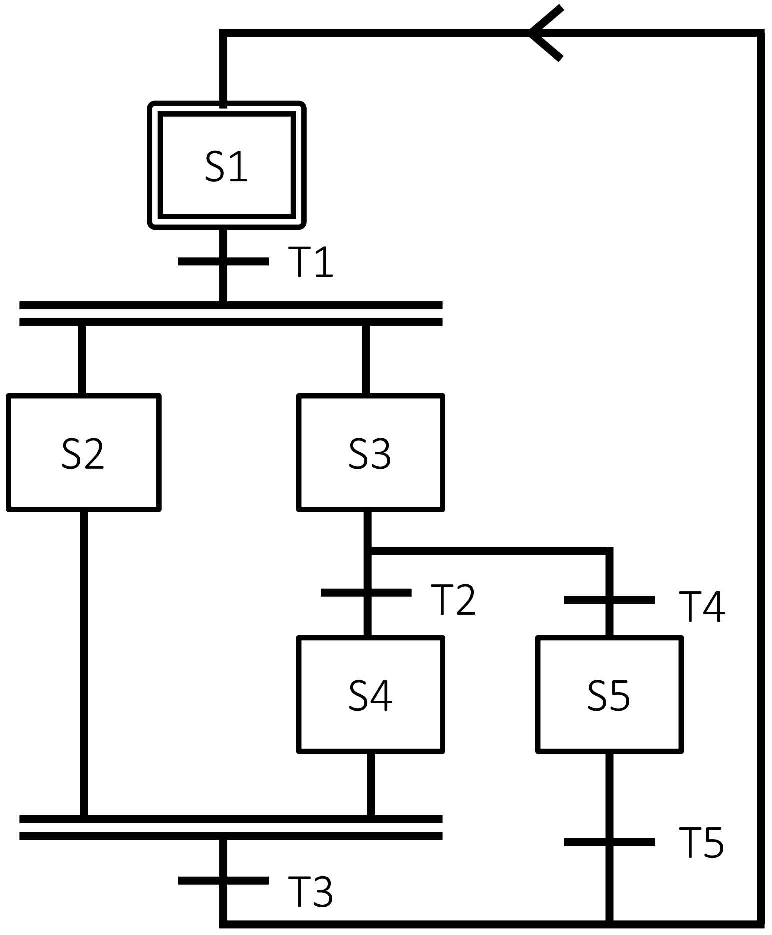
# Simultaneous sequences



# Simultaneous sequences

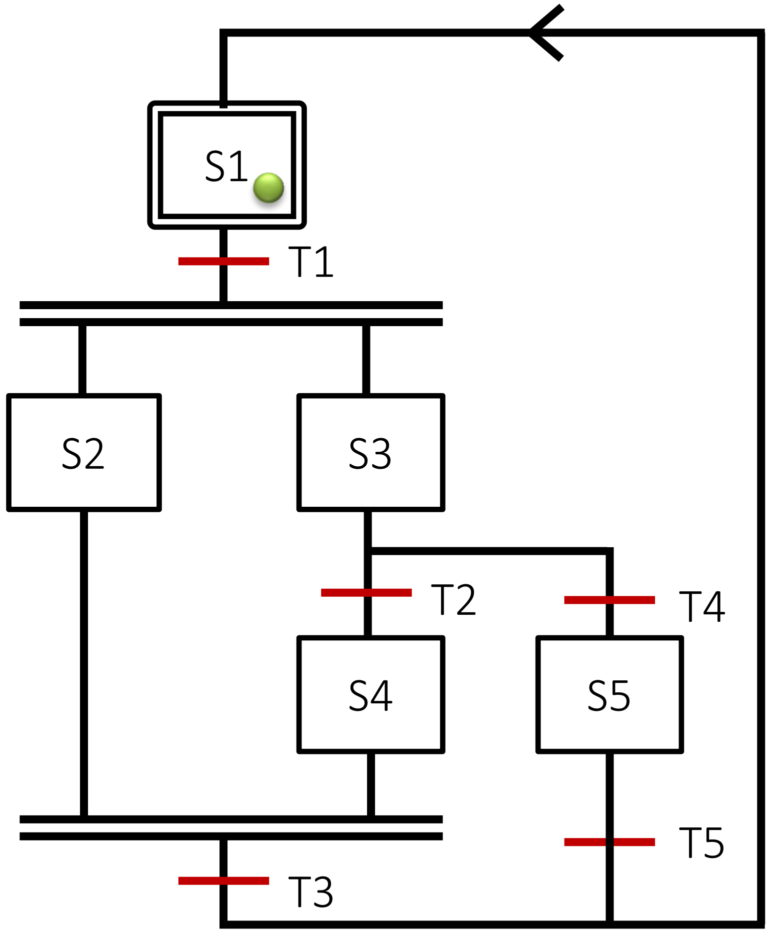


# Unsafe networks

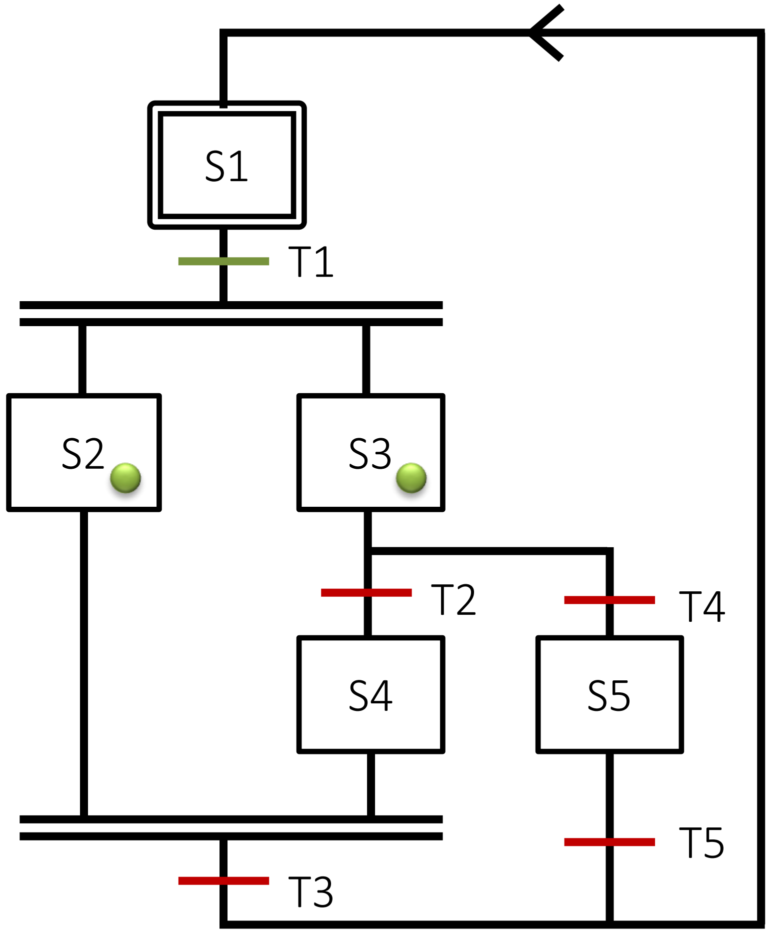


- A network is unsafe if it might exhibit uncontrolled proliferation of tokens, i.e. there might be multiple tokens present fully or partially outside parallel sequences
- Unsafe networks raise a compile-time error

# Unsafe networks



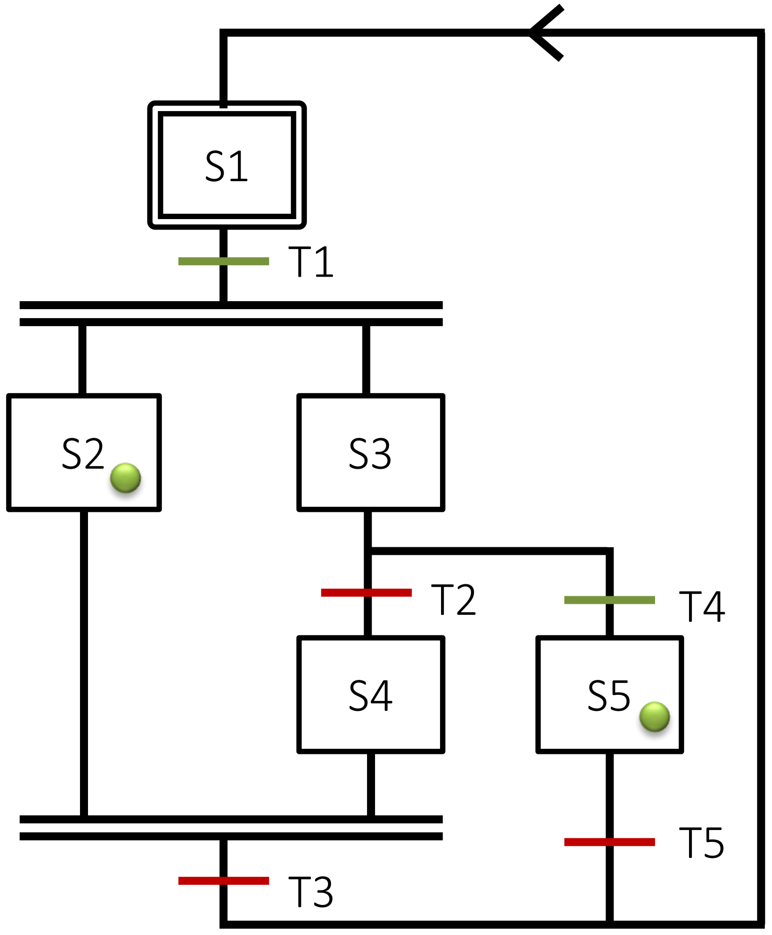
- A network is unsafe if it might exhibit uncontrolled proliferation of tokens, i.e. there might be multiple tokens present fully or partially outside parallel sequences
- Unsafe networks raise a compile-time error



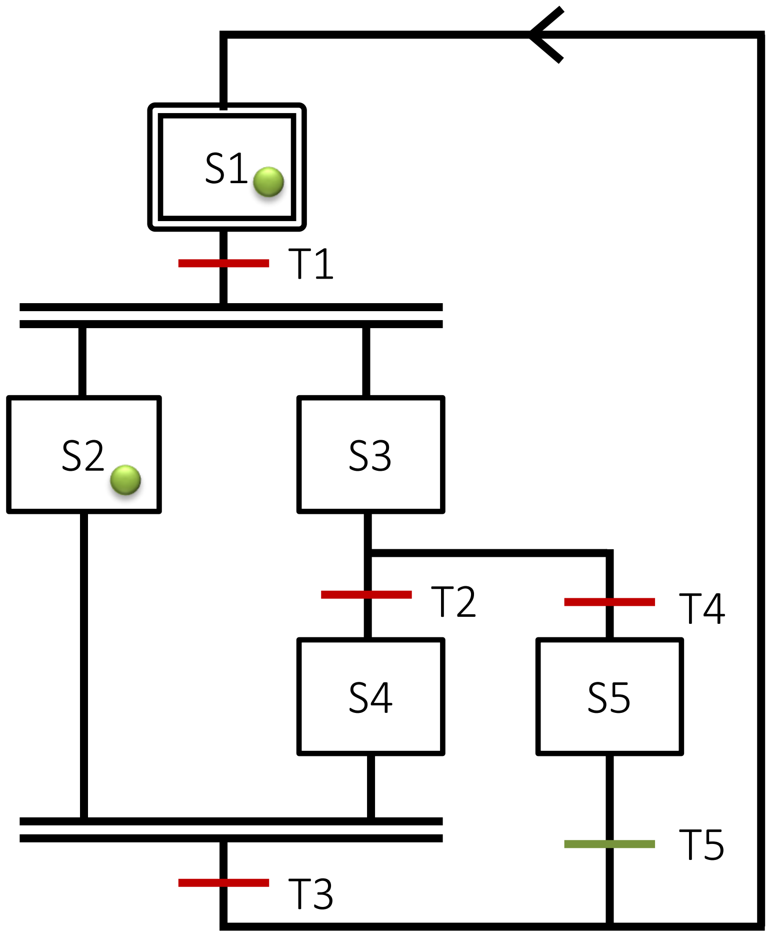
# Unsafe networks

- A network is unsafe if it might exhibit uncontrolled proliferation of tokens, i.e. there might be multiple tokens present fully or partially outside parallel sequences
- Unsafe networks raise a compile-time error

## Unsafe networks



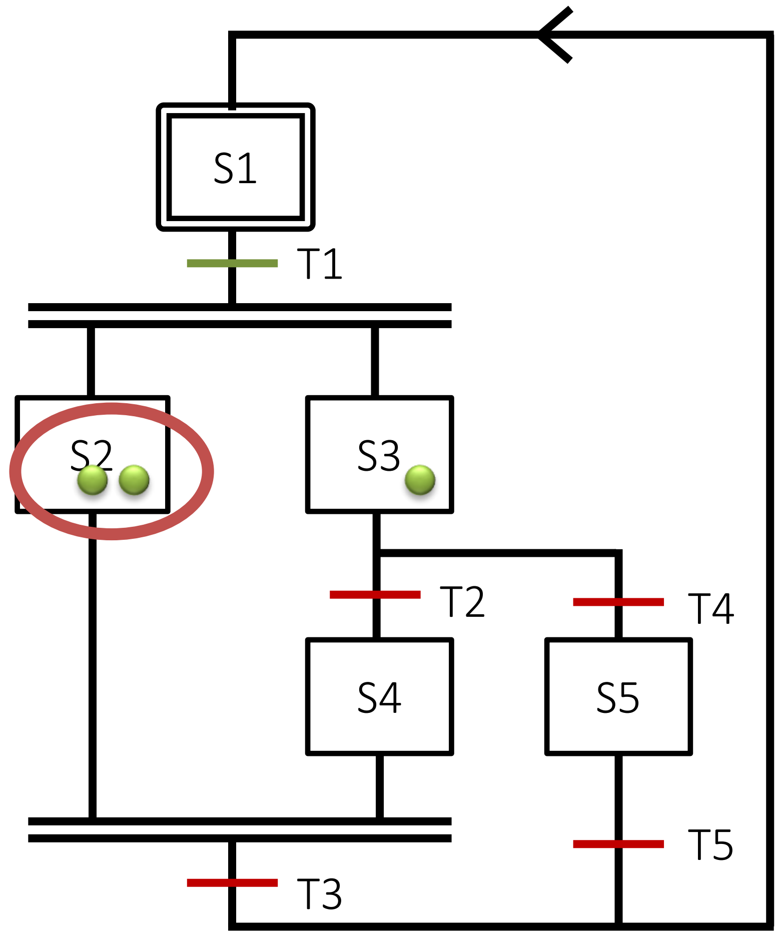
- A network is unsafe if it might exhibit uncontrolled proliferation of tokens, i.e. there might be multiple tokens present fully or partially outside parallel sequences
- Unsafe networks raise a compile-time error



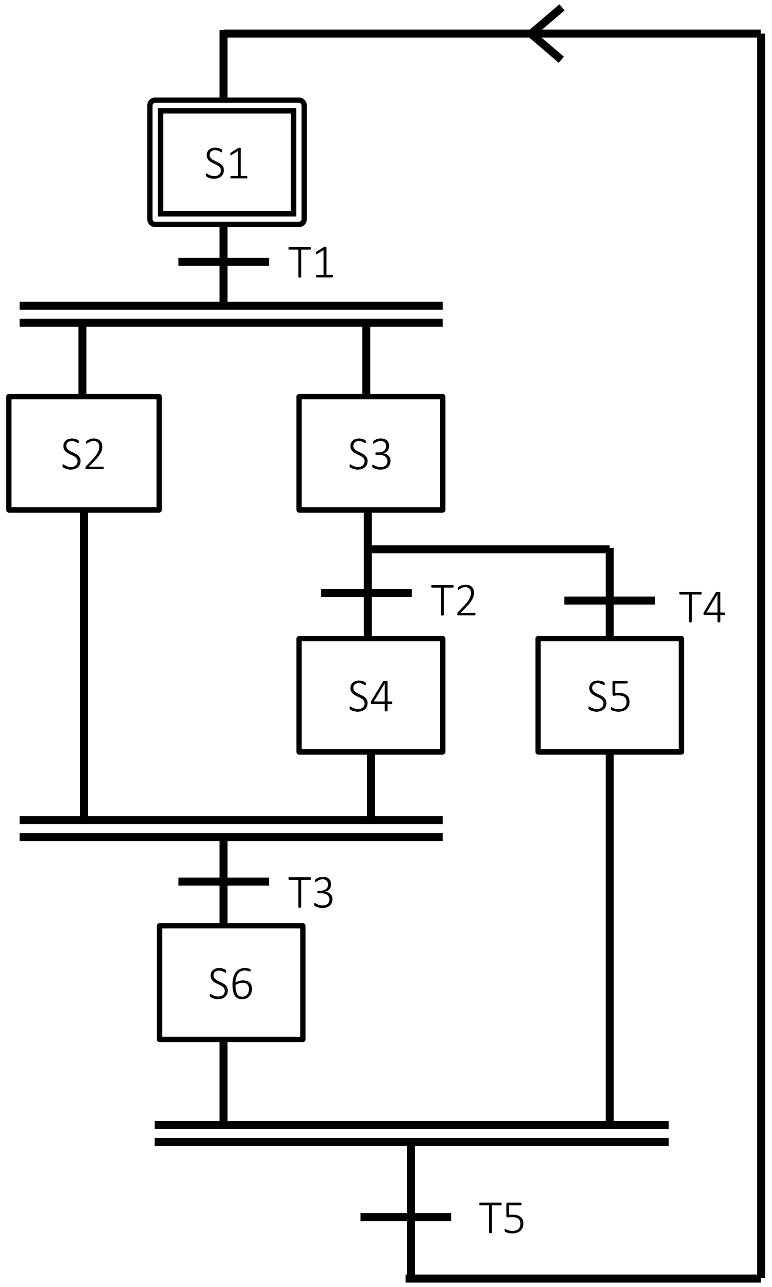
# Unsafe networks

- A network is unsafe if it might exhibit uncontrolled proliferation of tokens, i.e. there might be multiple tokens present fully or partially outside parallel sequences
- Unsafe networks raise a compile-time error

# Unsafe networks

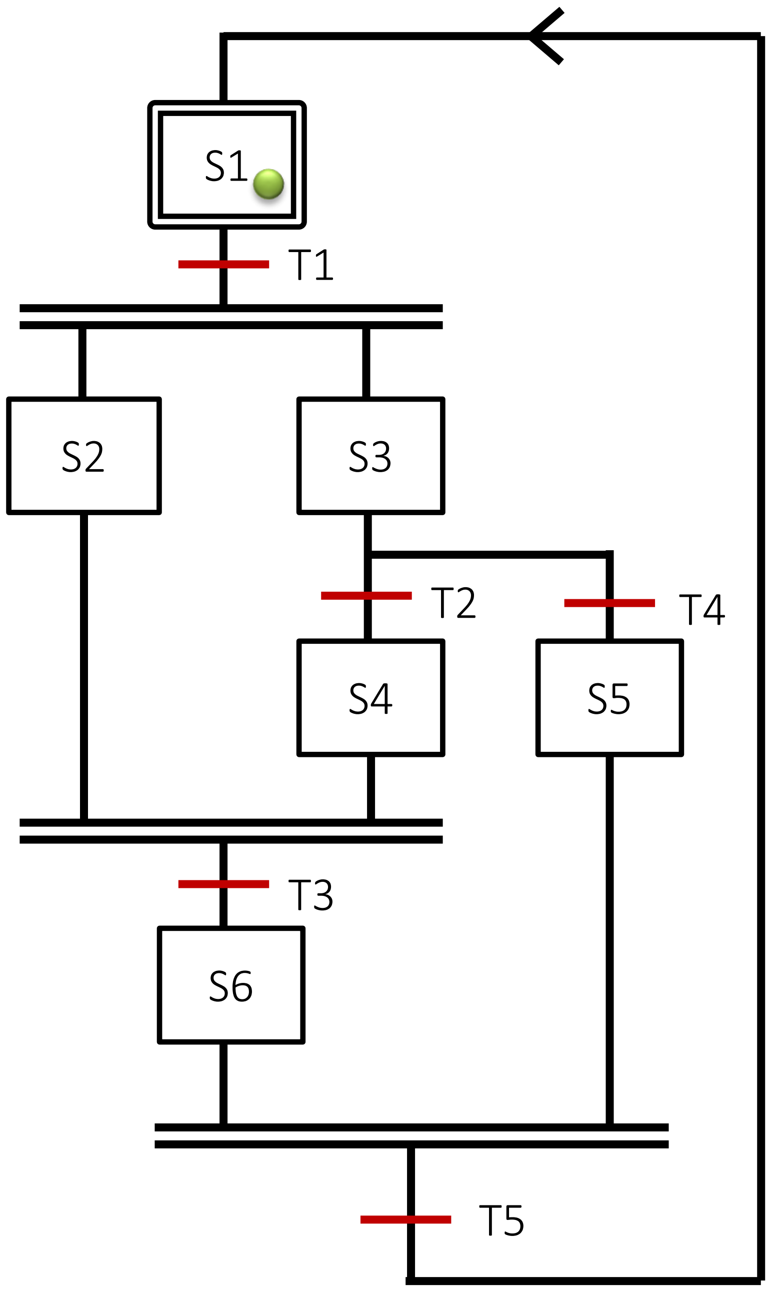


- A network is unsafe if it might exhibit uncontrolled proliferation of tokens, i.e. there might be multiple tokens present fully or partially outside parallel sequences
- Unsafe networks raise a compile-time error



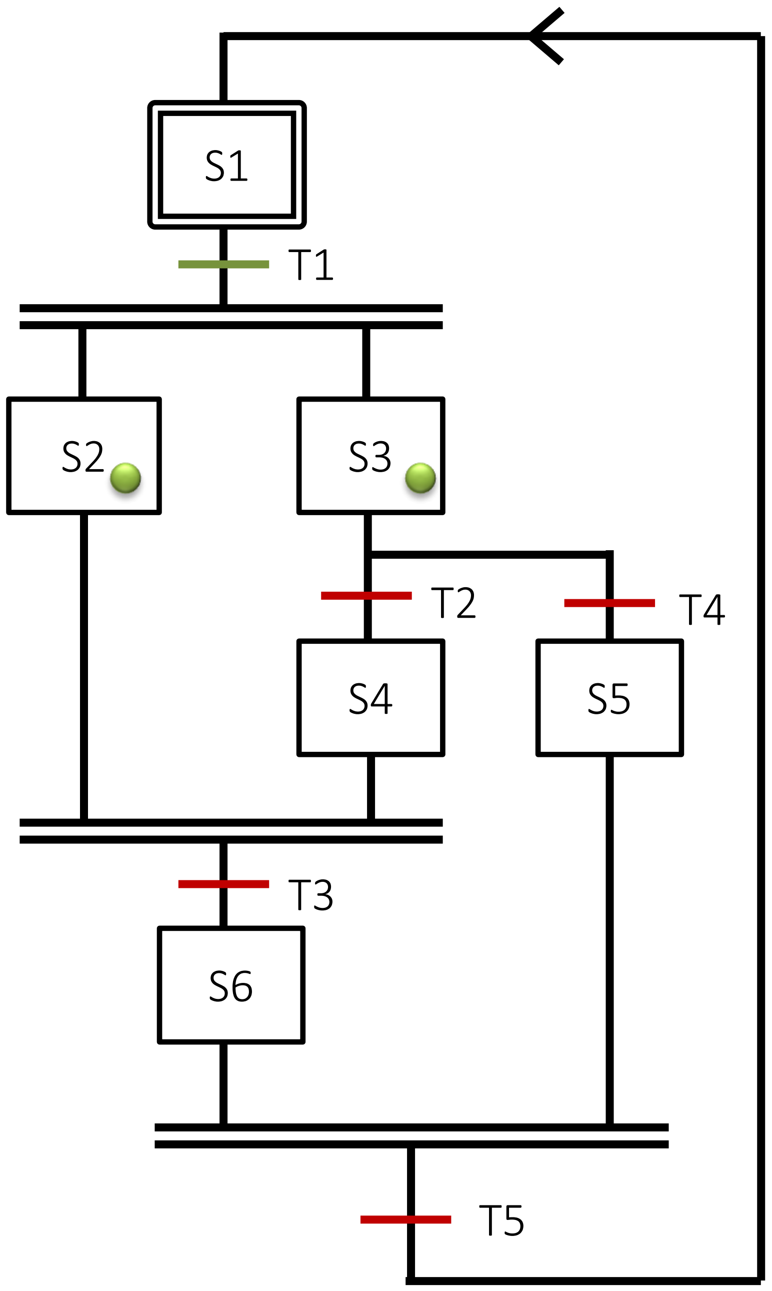
# Unreachable networks

- A networks if unreachable if an unintentional deadlock situation might occur during its evolution
- Unsafe networks raise a compile-time error



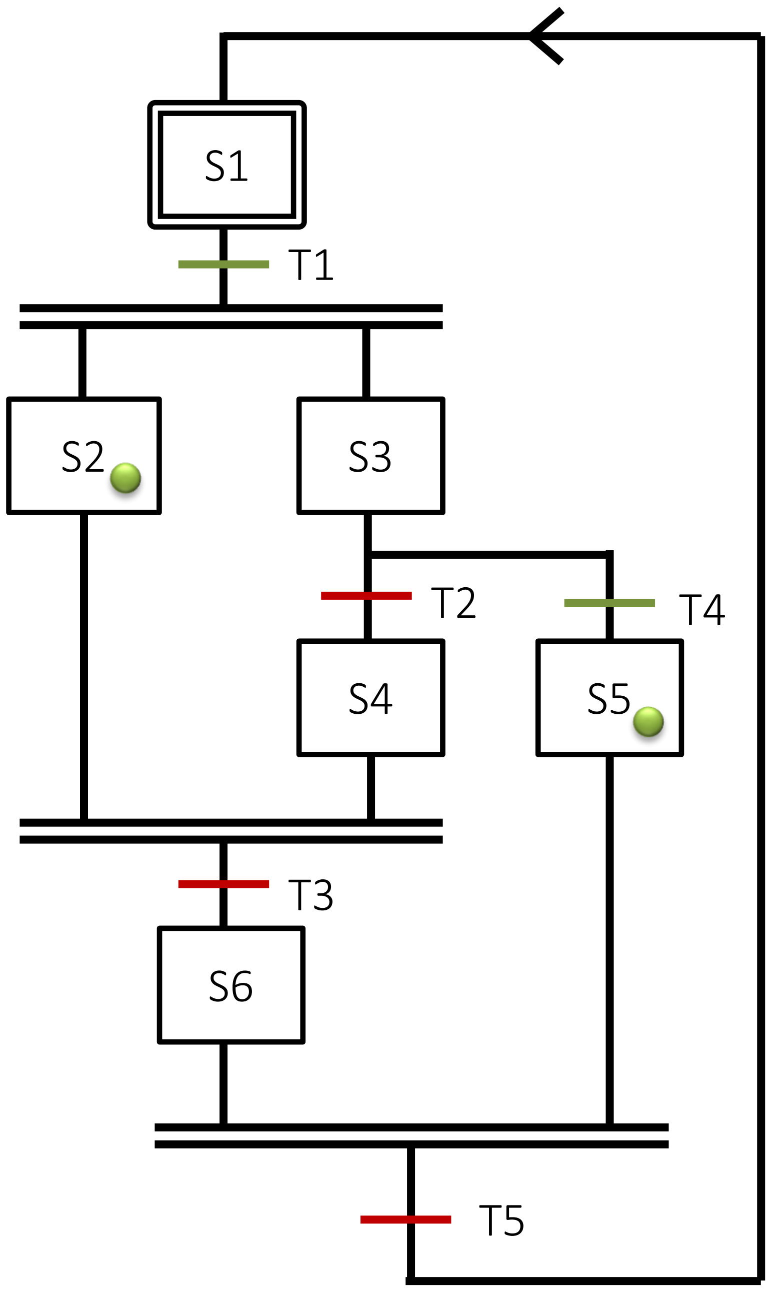
# Unreachable networks

- A networks if unreachable if an unintentional deadlock situation might occur during its evolution
- Unsafe networks raise a compile-time error



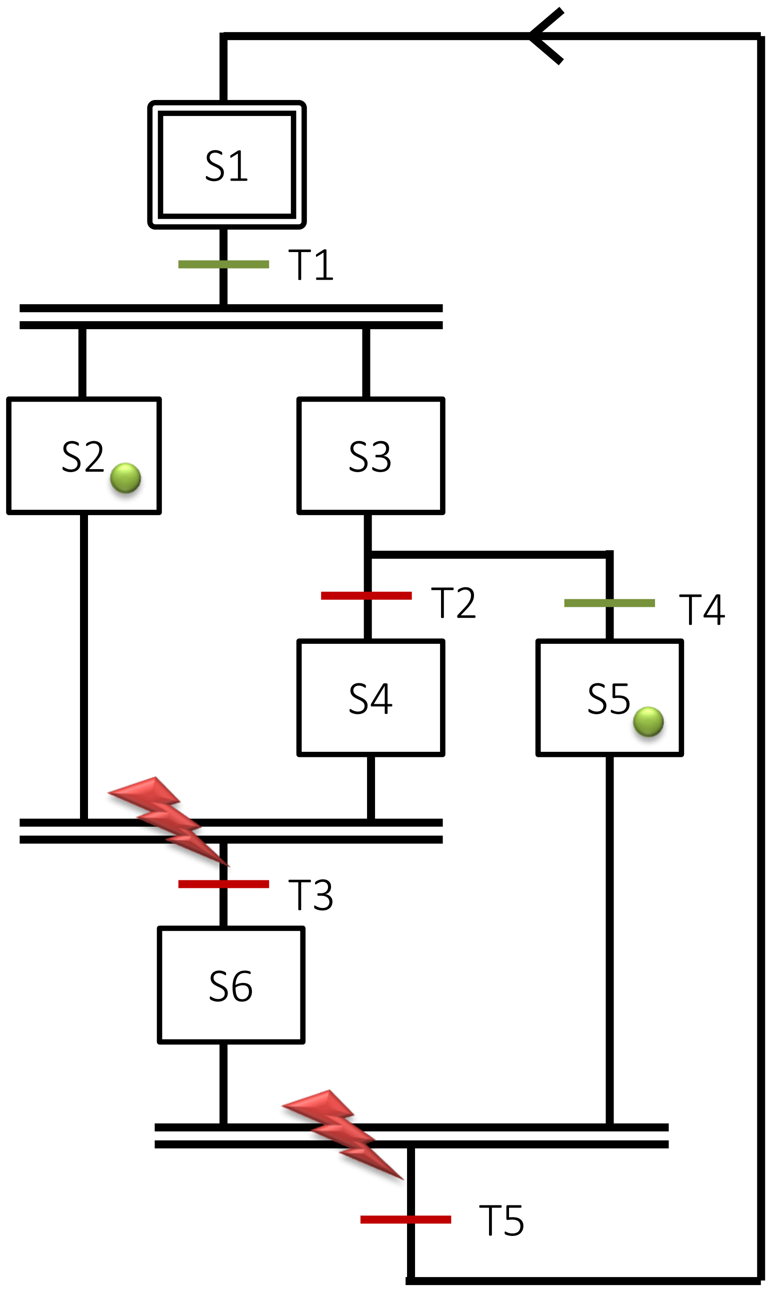
# Unreachable networks

- A networks if unreachable if an unintentional deadlock situation might occur during its evolution
- Unsafe networks raise a compile-time error



# Unreachable networks

- A networks if unreachable if an unintentional deadlock situation might occur during its evolution
- Unsafe networks raise a compile-time error



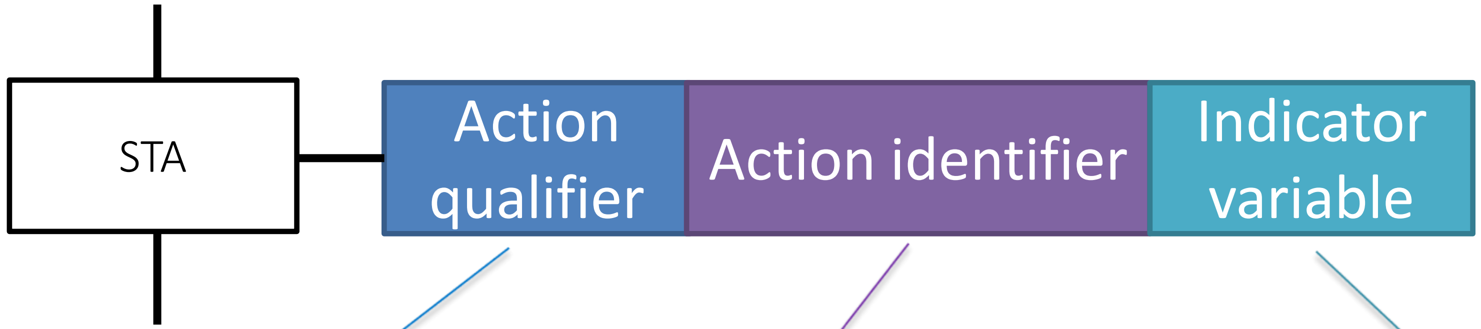
# Unreachable networks

- A networks if unreachable if an unintentional deadlock situation might occur during its evolution
- Unsafe networks raise a compile-time error

# Actions

- Zero or more actions can be associated to each step
- Boolean action: Boolean variable set by the action
- Non-Boolean action:
  - IL instructions
  - ST statements
  - LD networks
  - FBD networks
  - an other SFC network (rare)
- The same action might be associated to multiple steps

# Association of steps and actions – the action block



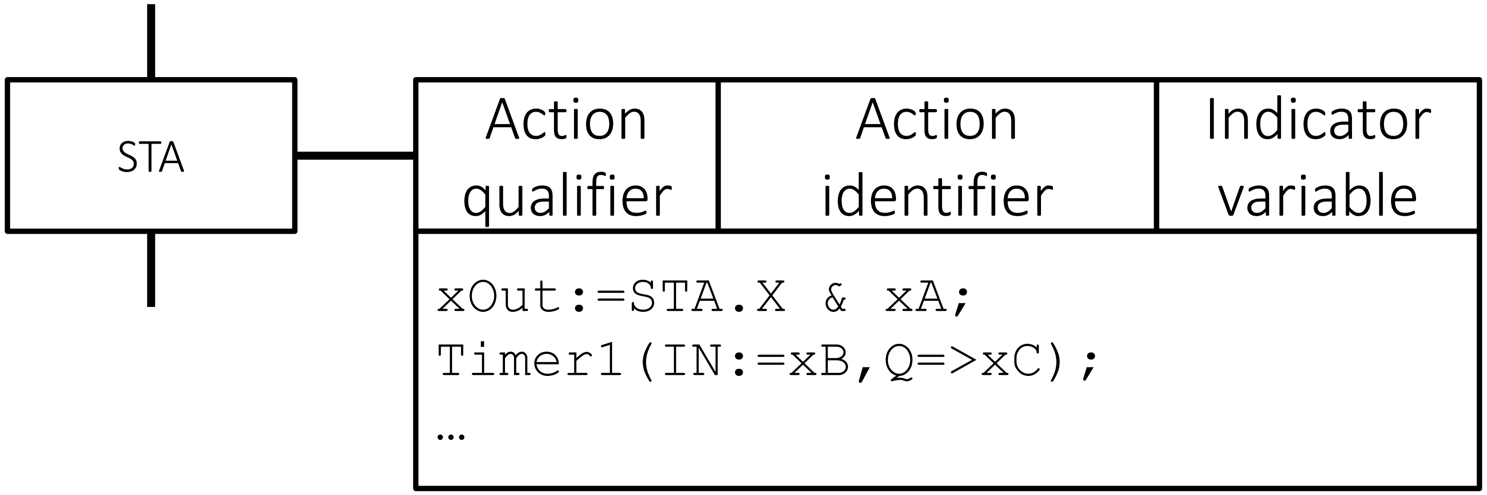
Defines how the action is executed

Optional Boolean variable set by the non-Boolean action (not used in practice, deprecated since 2013)

Action identifier (Boolean `VAR` or `VAR_OUT` variable or identifier of an *Action* block)

# Direct action definition

- Boolean action: a variable declared in a `VAR` or `VAR_OUT` block with the same identifier as the action
- Non-Boolean actions: body of the action might be given inside the action block (not supported in practice)
- Direct action definition is used only for Boolean actions



```
xOut:=STA.X & xA;
Timer1(IN:=xB,Q=>xC);
…
```

## Definition of non-Boolean actions

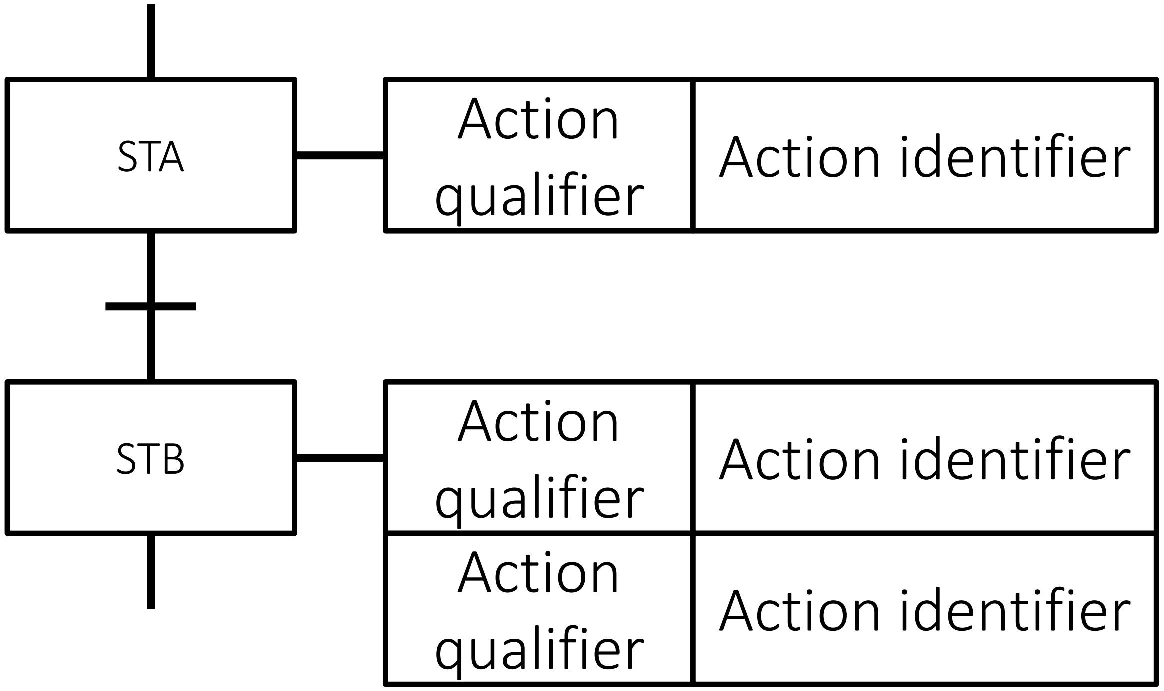
- LD, FBD, SFC: graphical definition (implementer specific, generally the same way as defining POUs)
- ST, IL: `ACTION` keyword

```
ACTION MyAction:
     xMyBool:=xBoolIn & Step8.X;
END_ACTION
```

- Exact syntax of action definition is implementer specific

# Associating actions with steps

- Graphically by connecting an action block to the step



# Associating actions with steps

- Textually inside STEP blocks (not supported in practice)

```
STEP StepName
     ActionName(Qual,QualParameter,IndVar);
END_STEP
```

- Some development environments support associating a non-stored action to a step without using an action block

## Standard action control

- Action flag of an action: $Q$
  - Boolean action: the Boolean variable is a copy of the action flag
  - non-Boolean action: the action is executed cyclically while the action flag is true
- Action flags are set by the action control logic hidden from the user

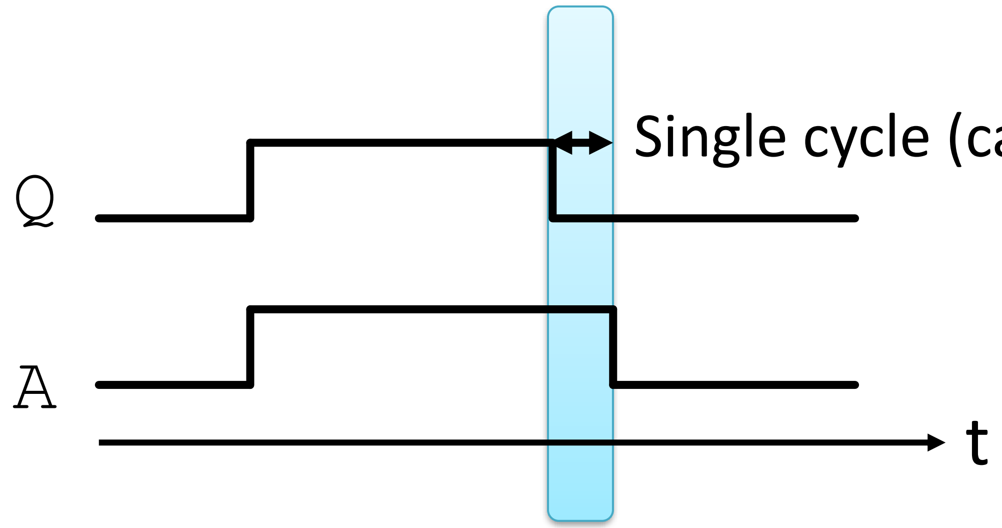
# Action control with final scan logic

- Action flag: `Q`
  - Boolean action: the Boolean variable is a copy of the action flag
  - non-Boolean action: not used
- Activity flag: `A`
  - non-Boolean action: the action is executed cyclically while the activity flag is true
  - Boolean action: not used
- Action and Activity flags are set by the action control hidden from the user

# Activity flag

- The activity flag is set along with the action flag but stays active for one single cycle (call) after resetting the action flag
- Execution of an action is the last one if

```
(Action.Q = FALSE)&(Action.A = TRUE)
```



# Implementation of action control

- A development environment might implement only standard or final scan action control
- Generally standard action control is used (e.g. by CODESYS)
- Additional (non-standard) features
  - entry action – executed once when the step is activated
  - exit action – executed once when the step is deactivated
  - *Last Scan* bit available for each action – true if the current execution is the last one

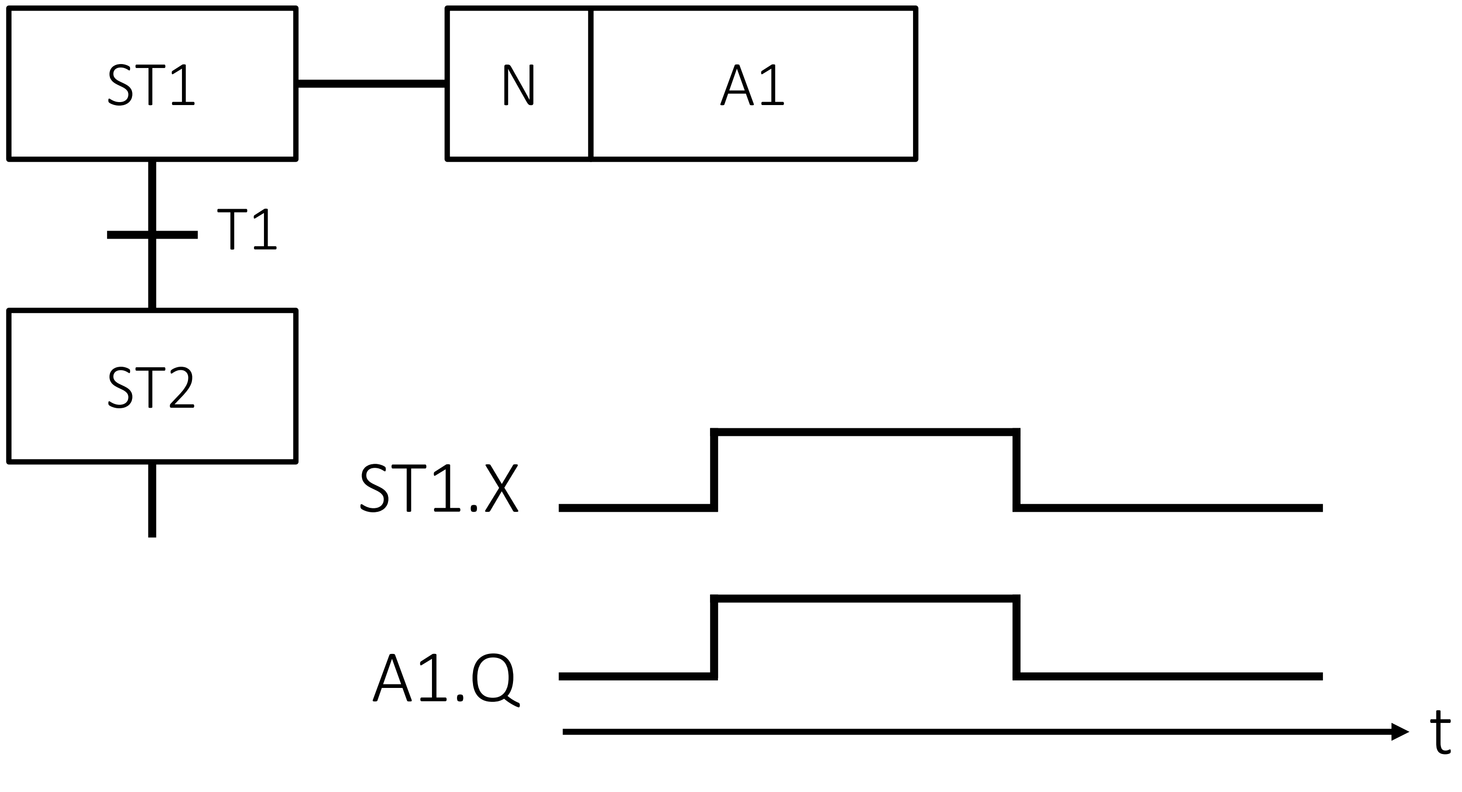
## Action qualifiers

| Qualifier | Meaning |
|---|---|
| N / none | **N**on stored |
| R | **R**eset (*Overriding Reset*) of stored action |
| S | **S**tored (*Set*) |
| L | Time **L**imited |
| D | Time **D**elayed |
| P | **P**ulse |
| SD | **S**tored and **D**elayed |
| DS | **D**elayed and **S**tored |
| SL | **S**tored and **L**imited |
| P1 | Pulse (rising edge) |
| P0 | Pulse (falling edge) |

Not used in practice, not available in most environments

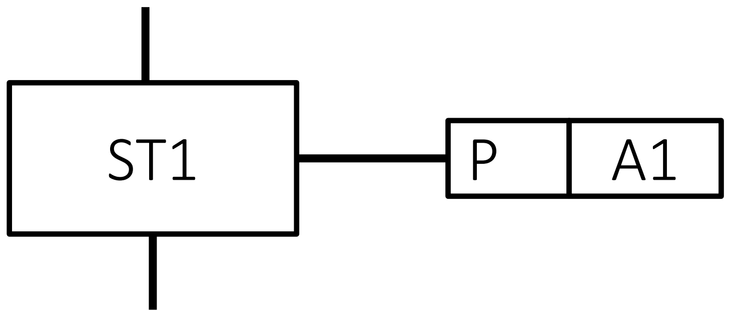
# Non stored action

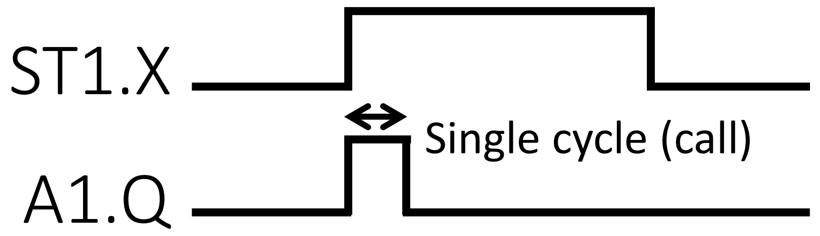
- Action flag Q is the copy of the step flag



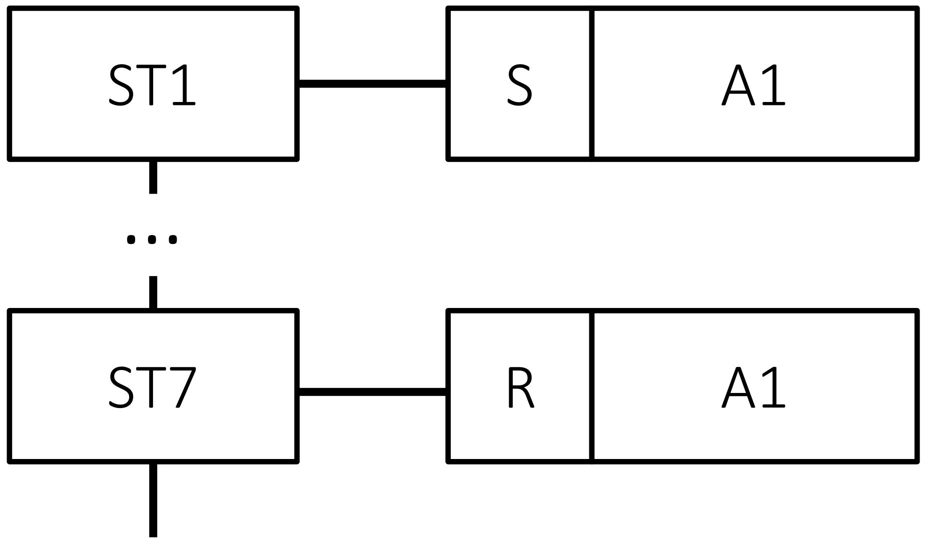
## Pulse action

- Action flag is set upon activating the state but deactivated during the next execution of the chart

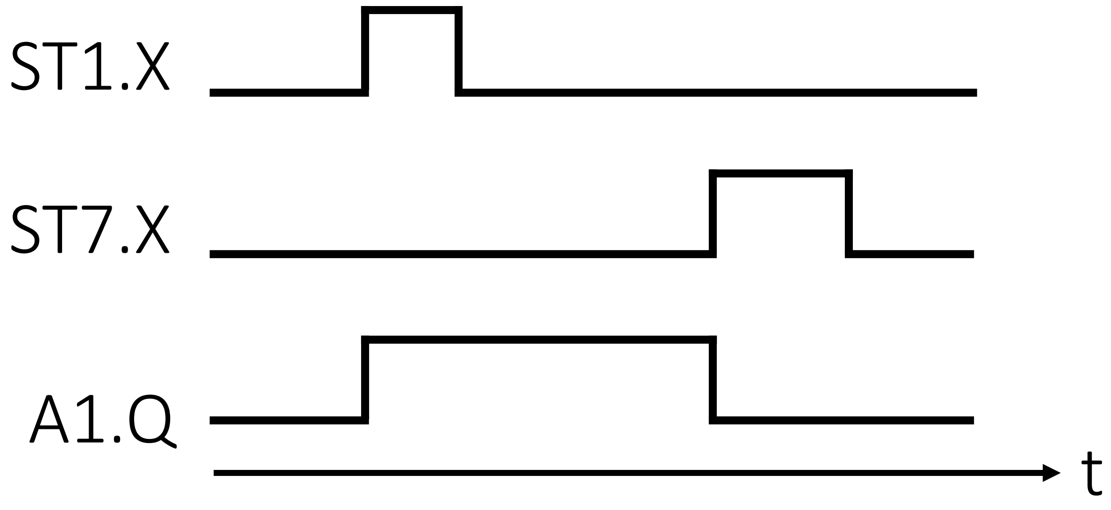




# Stored action with Reset

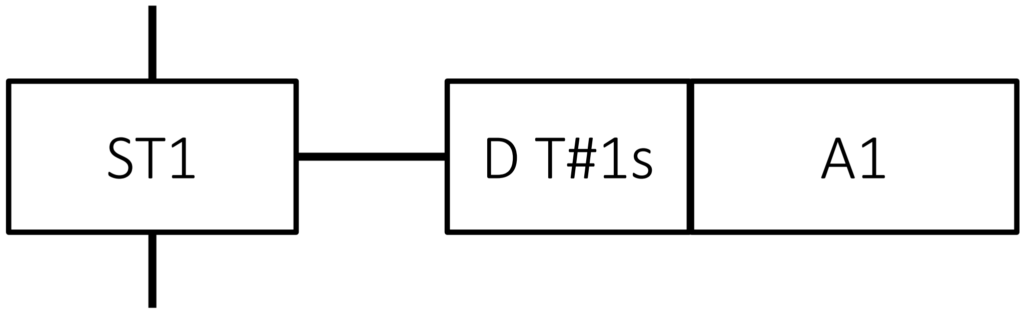


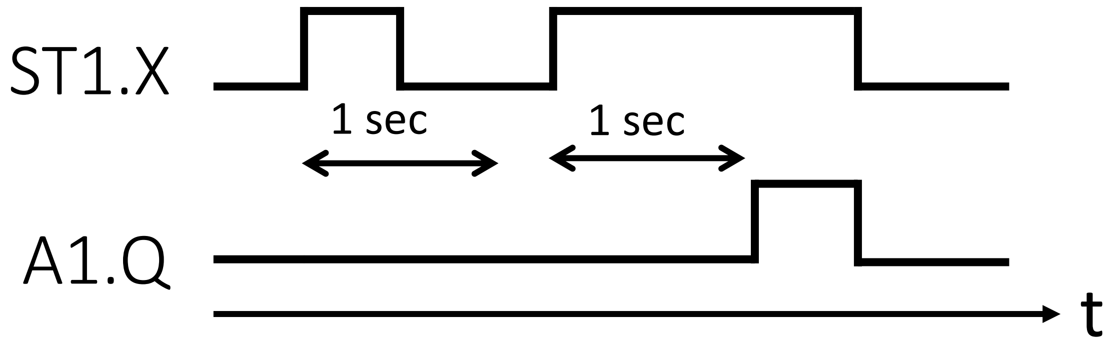
- Reset is overriding



# Time delayed action

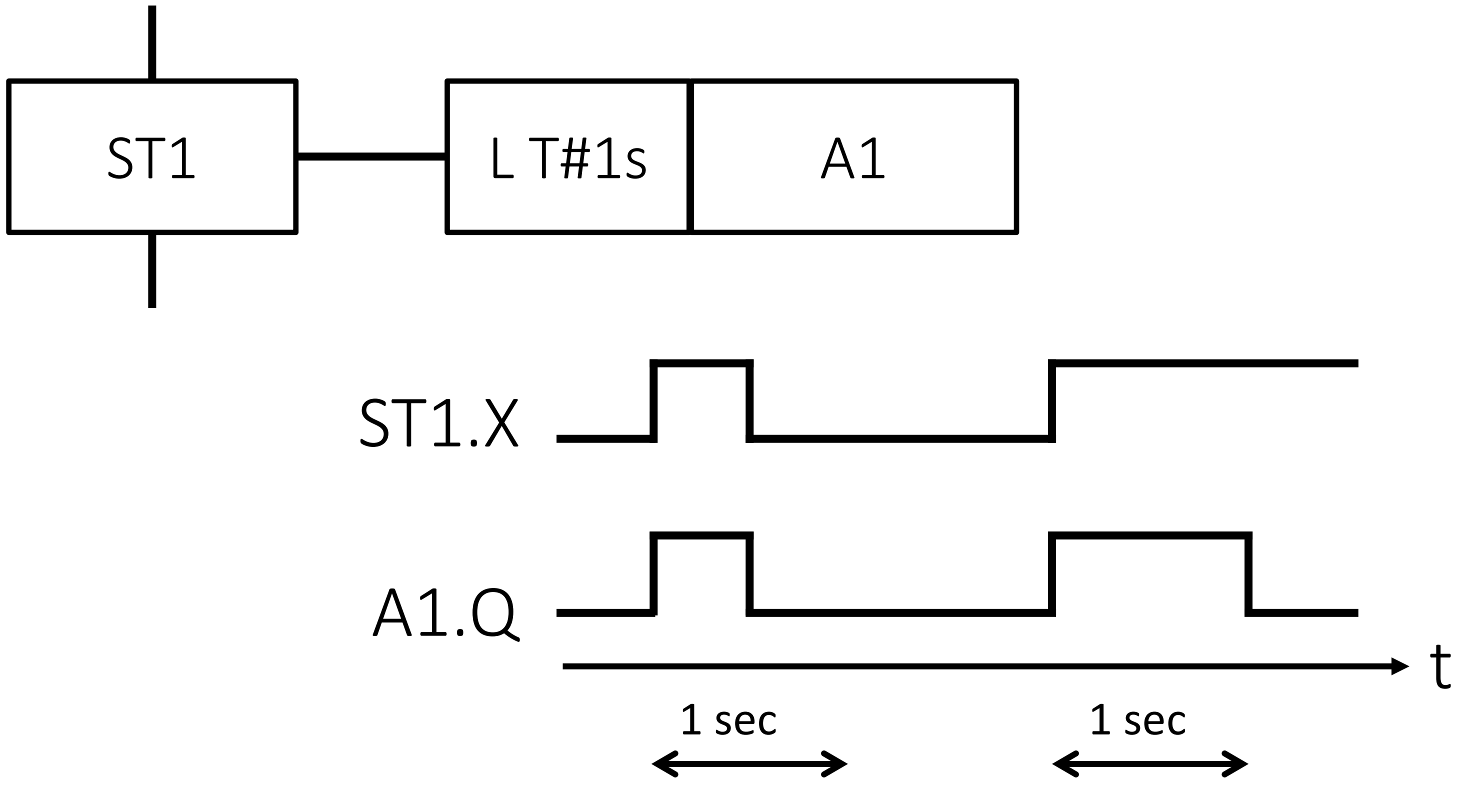
- Action flag `Q` is set the given time after the activation of the step if the step is still active
- Action flag is reset when the step is deactivated





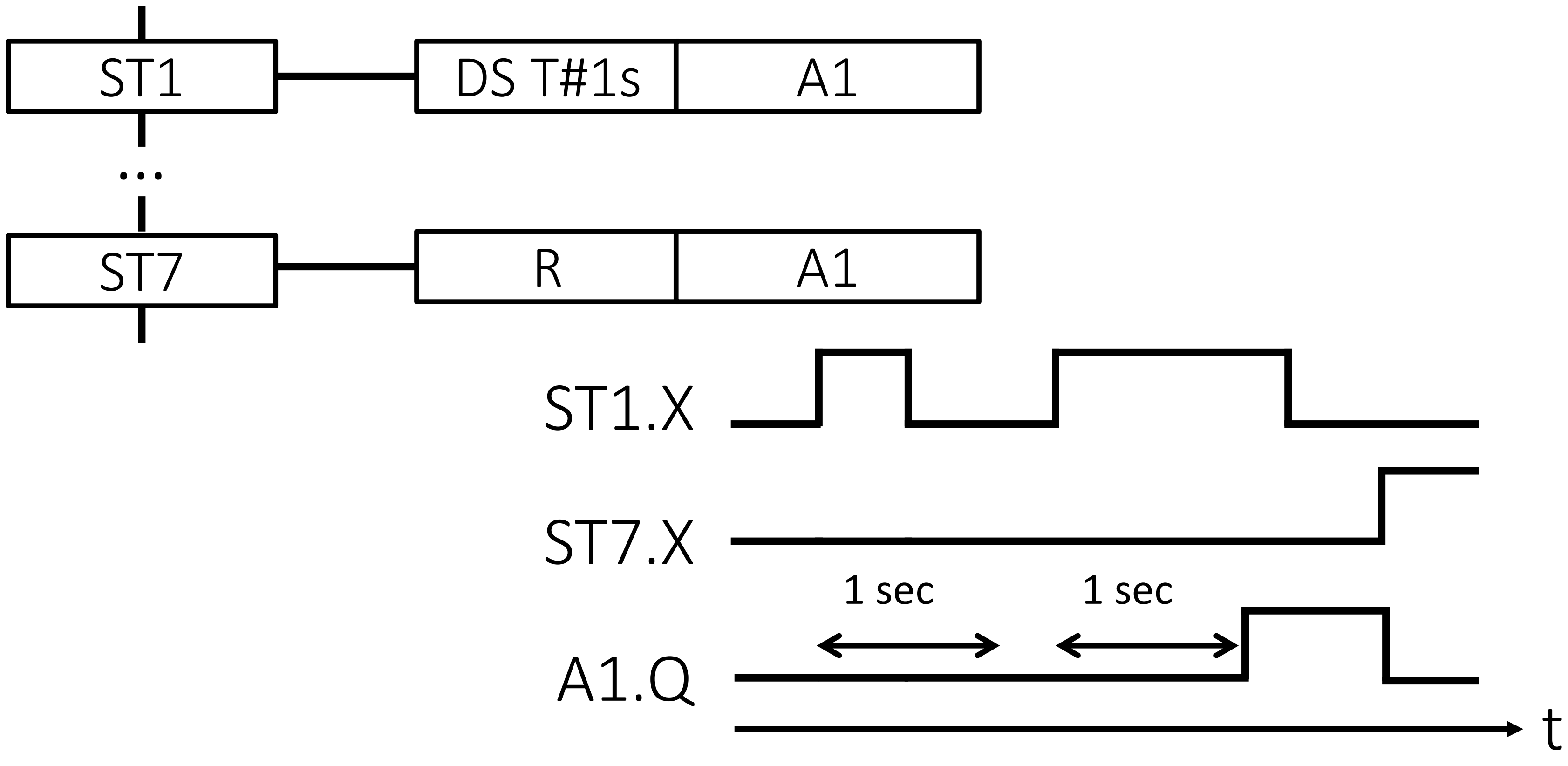
# Time limited action

- Action flag Q is set upon activating the step
- Action is active until the deactivation of the step, but at most for the specified period



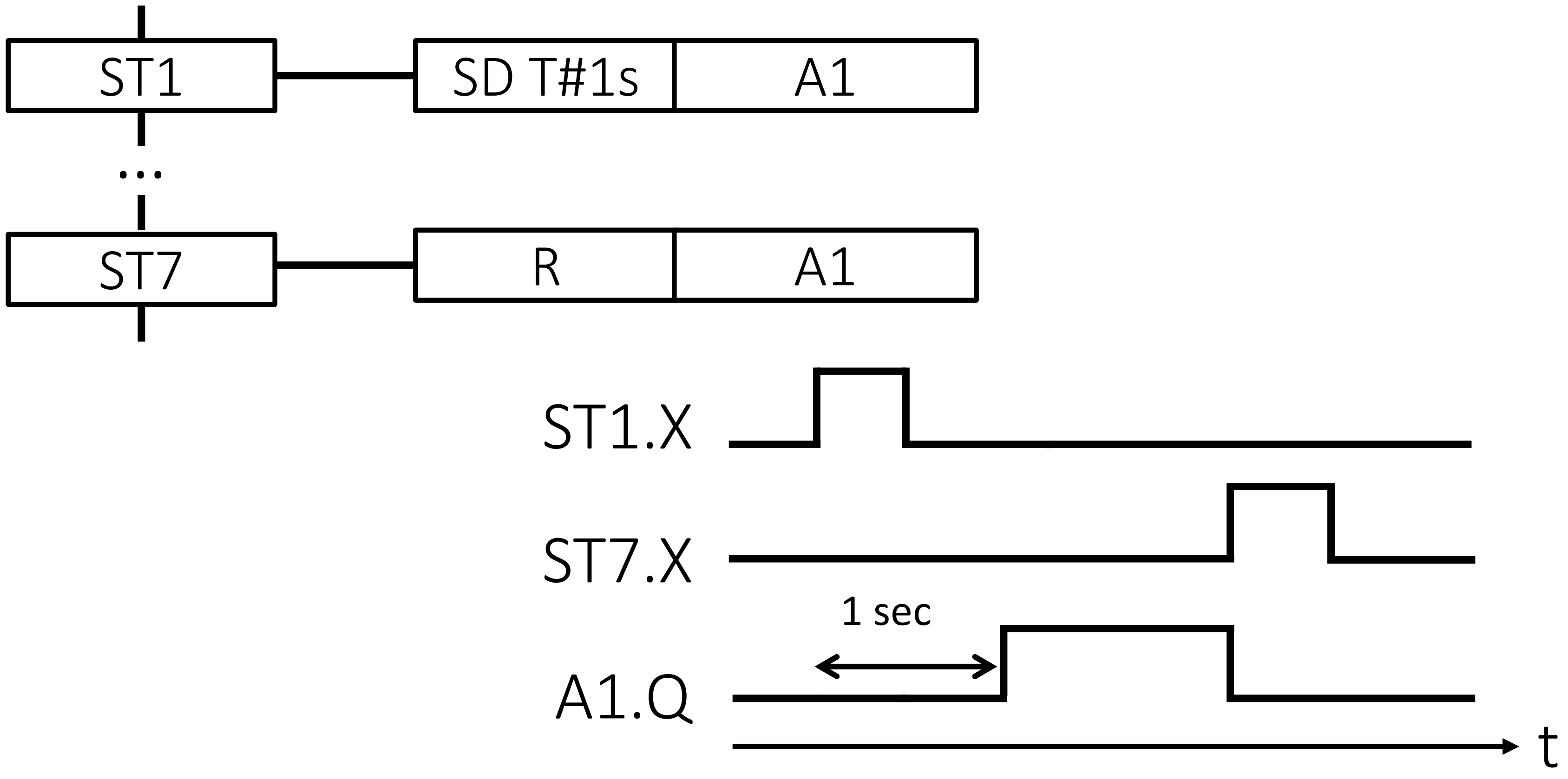
# Delayed and stored action

- Action flag Q is set the given time after the activation of the step, if the step is still active
- Cleared by executing the action with R qualifier only, not by deactivating the step



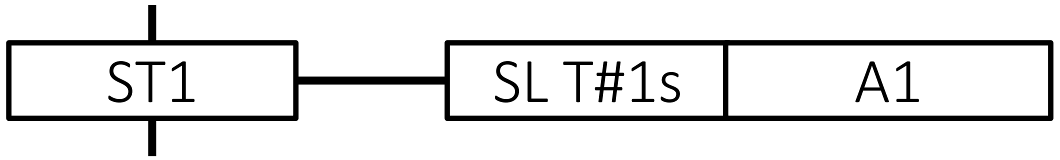
# Stored and delayed action

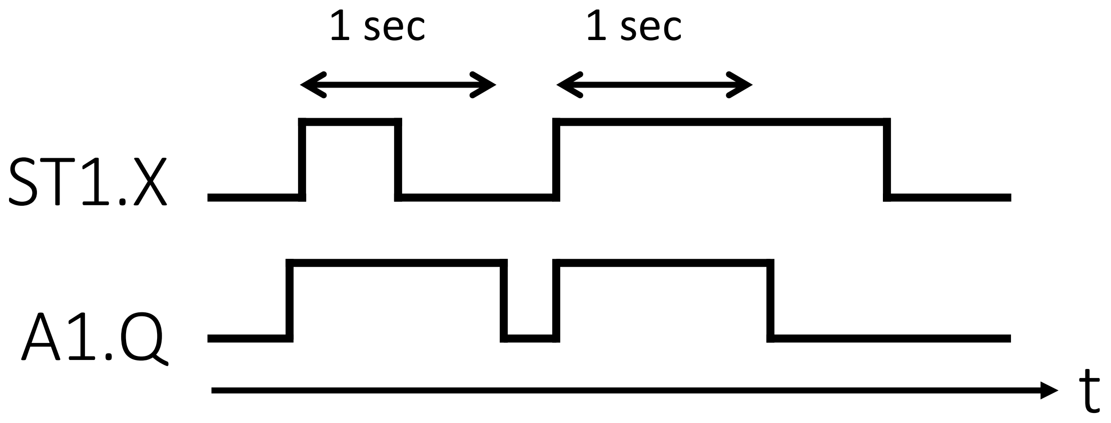
- Action flag Q is set the given time after the activation of the step even if the step is not active anymore
- Cleared by executing the action with R qualifier only



# Stored and limited action

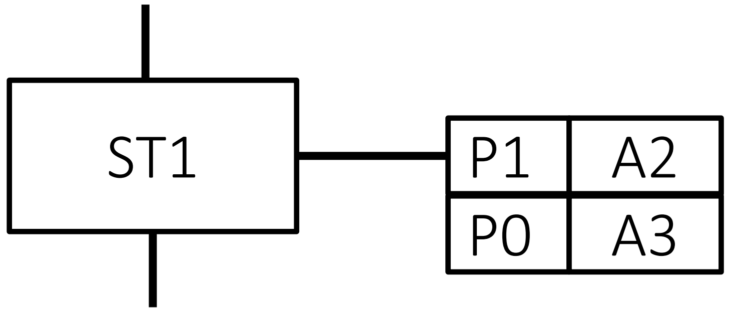
- Action flag Q is set when the step is activated
- Action flag is reset after the given time even if the step is still active
- Needs to be cleared by calling with a Reset qualifier before next execution

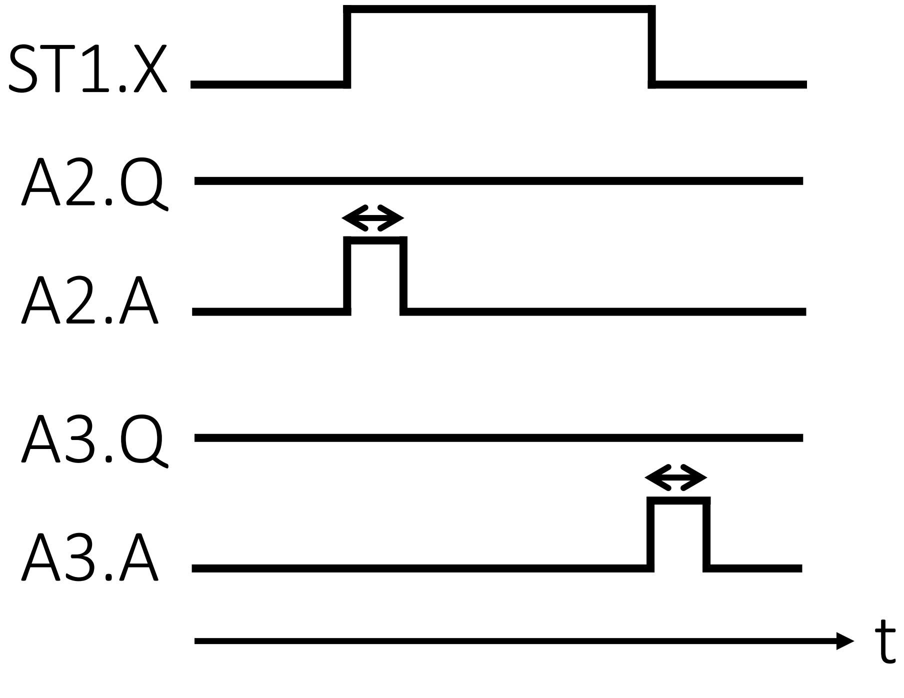




# P1 and P0 pulse actions

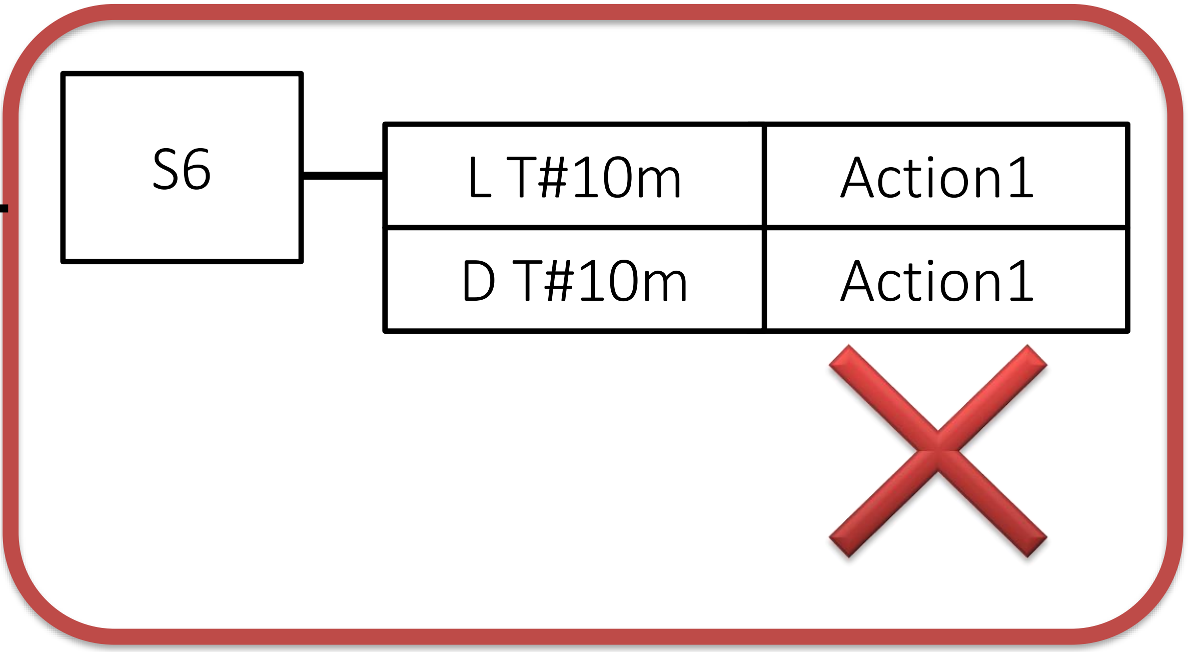
- P1 and P0 action qualifiers are defined by the standard but are not supported by development environments
- P1 és P0 actions are applicable for non-Boolean actions if final scan action control is used
- They set the activity flag but not the action flag

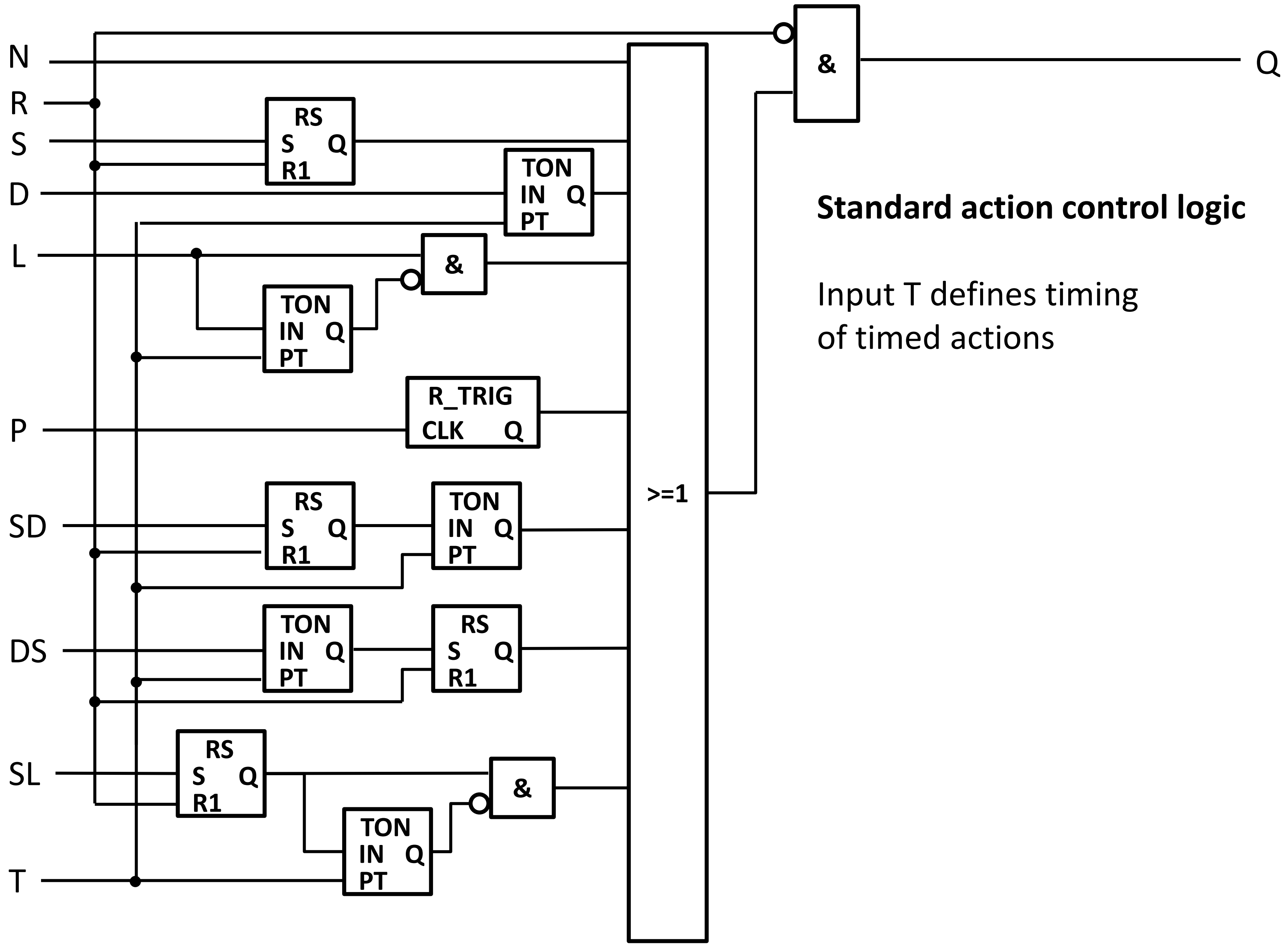




## Timed qualifier

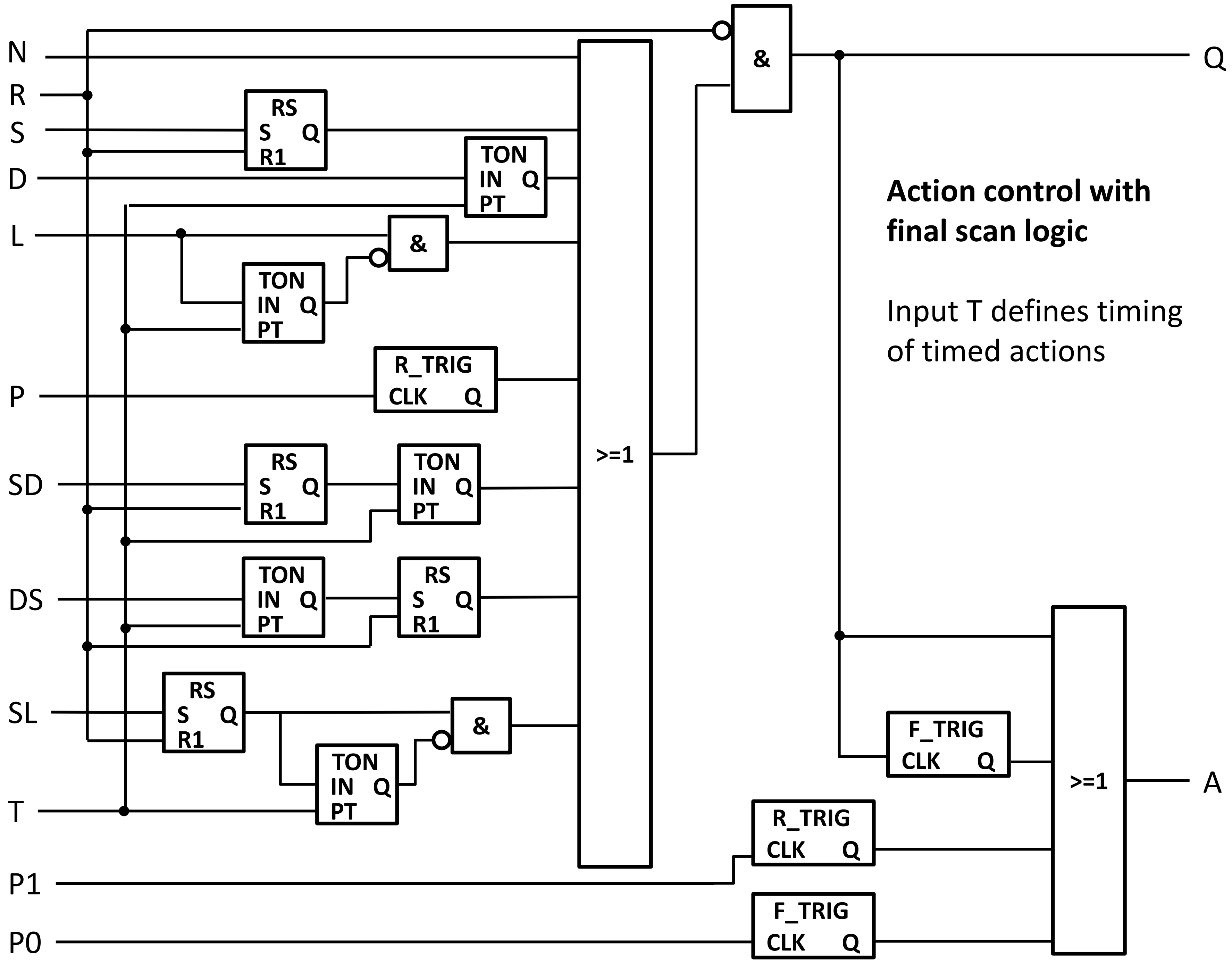
- A given action can be associated to a step with one single timed qualifier only
  - L
  - D
  - SD
  - DS
  - SL





**Standard action control logic**

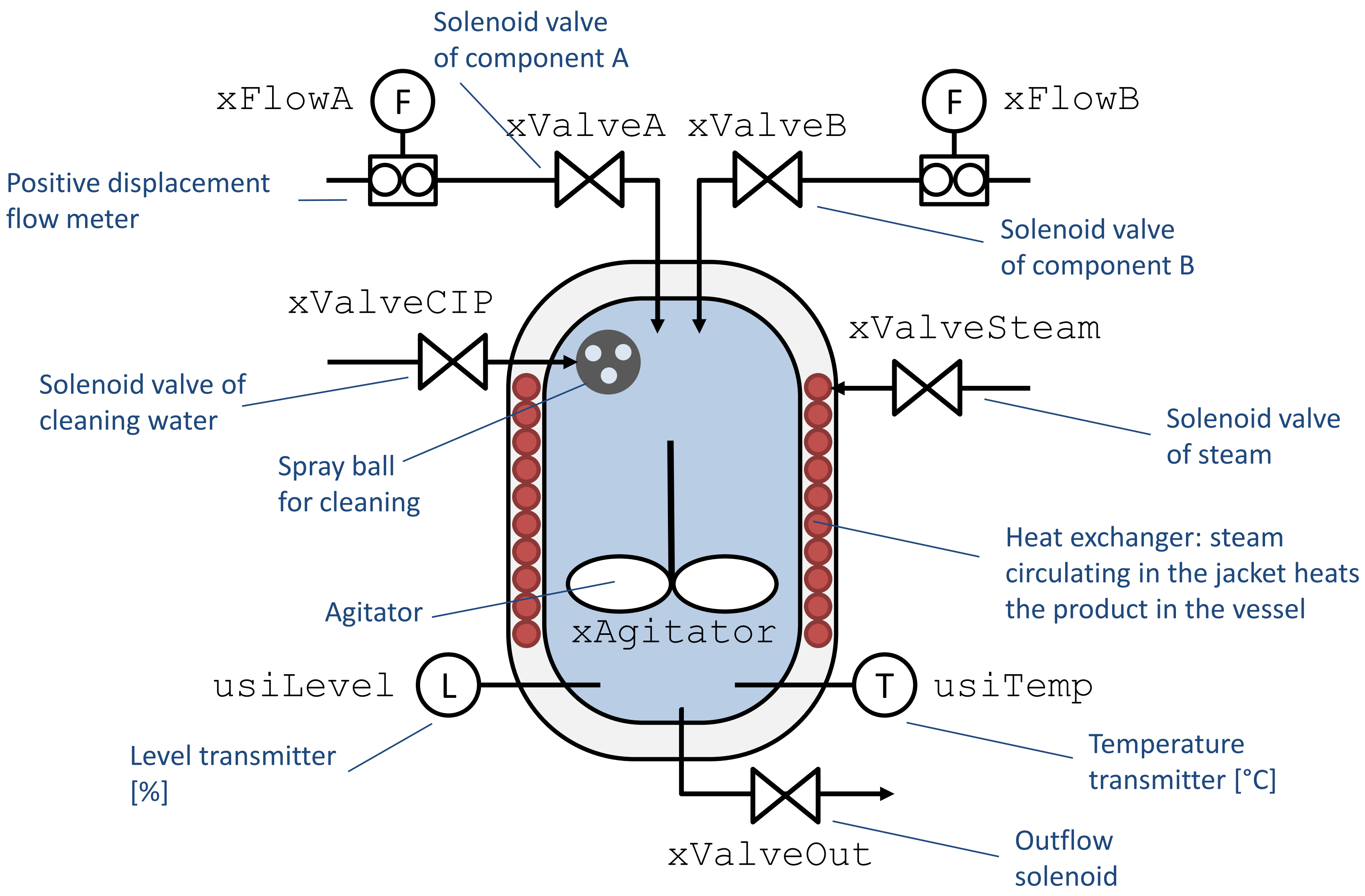
Input T defines timing
of timed actions



**Action control with final scan logic**

Input T defines timing of timed actions

# Problem – The process


Solenoid valve of component A
xFlowA
F
xValveA
xValveB
F
xFlowB
Positive displacement flow meter
Solenoid valve of component B
xValveCIP
xValveSteam
Solenoid valve of cleaning water
Solenoid valve of steam
Spray ball for cleaning
Heat exchanger: steam circulating in the jacket heats the product in the vessel
Agitator
xAgitator
usiLevel
L
T
usiTemp
Level transmitter [%]
Temperature transmitter [°C]
xValveOut
Outflow solenoid

## Problem - Specification

- Pressing the Start button shall start a production cycle according to the followings:
  1. 10 units of component A and 20 units of component B shall be stored in the vessel, and if the level rises to 10%, the agitator shall be turned on
  2. If required units have been stored of both components, the agitator shall be kept running, and after 5 seconds, heating of the vessel shall be turned on
  3. If the temperature of the product reaches 60°C, the agitator shall be turned off, but the temperature shall be kept at 60°C for an additional 10 seconds
  4. After the 10 second delay, the agitator shall be turned on and kept running until the opening of the drain valve, but at most for 8 seconds
  5. If the temperature of the product drops below 40°C, the outflow valve shall be opened and kept open until the vessel becomes empty.

## Problem - Specification

- Pressing the Clean pushbutton, cleaning of the vessel shall be started according to the followings:
  1. The CIP spray ball shall be supplied with cleaning water by opening the CIP valve for 5 seconds
  2. The outflow valve shall be kept while the cleaning spay ball is supplied with cleaning water and for 3 additional seconds
  3. A new process (production or cleaning) can be started after closing the drain valve

*Timings in the problem are not realistic, real processes need much longer durations for the operations.*

## Non-Boolean actions

- Counting units of components stored in the vessel
  - positive displacement flow meters provide a pulse upon each unit stored (incremental encoder on the shaft)
  - the amount stored in the tank can be determined by counting the pulses
  - pulses can be counted by standard CTU function blocks
  - counters shall be reset upon finishing a cycle

- Temperature control
  - solenoid valve allows the use of hysteresis control
  - temperature of the product shall be kept in range $60 \pm 2°\mathrm{C}$

# Counting the amount stored in the vessel

```
VAR
   CounterA, CounterB : CTU;
END_VAR
```

Instantiating counter FB instances by declaring them as local variables of the program

```
ACTION CountA
   CounterA(CU:=xFlowA, PV:=10, RESET:=FALSE);
END_ACTION
```

Counting amount stored of component *A* with setting the preset value

```
ACTION CountB
   CounterB(CU:=xFlowB, RESET:=FALSE);
END_ACTION
```

Counting amount stored of component *B* without setting the preset value

```
ACTION ResetCounters
   CounterA(RESET:=TRUE);
   CounterB(RESET:=TRUE);
END_ACTION
```

Resetting the counters

## Temperature control

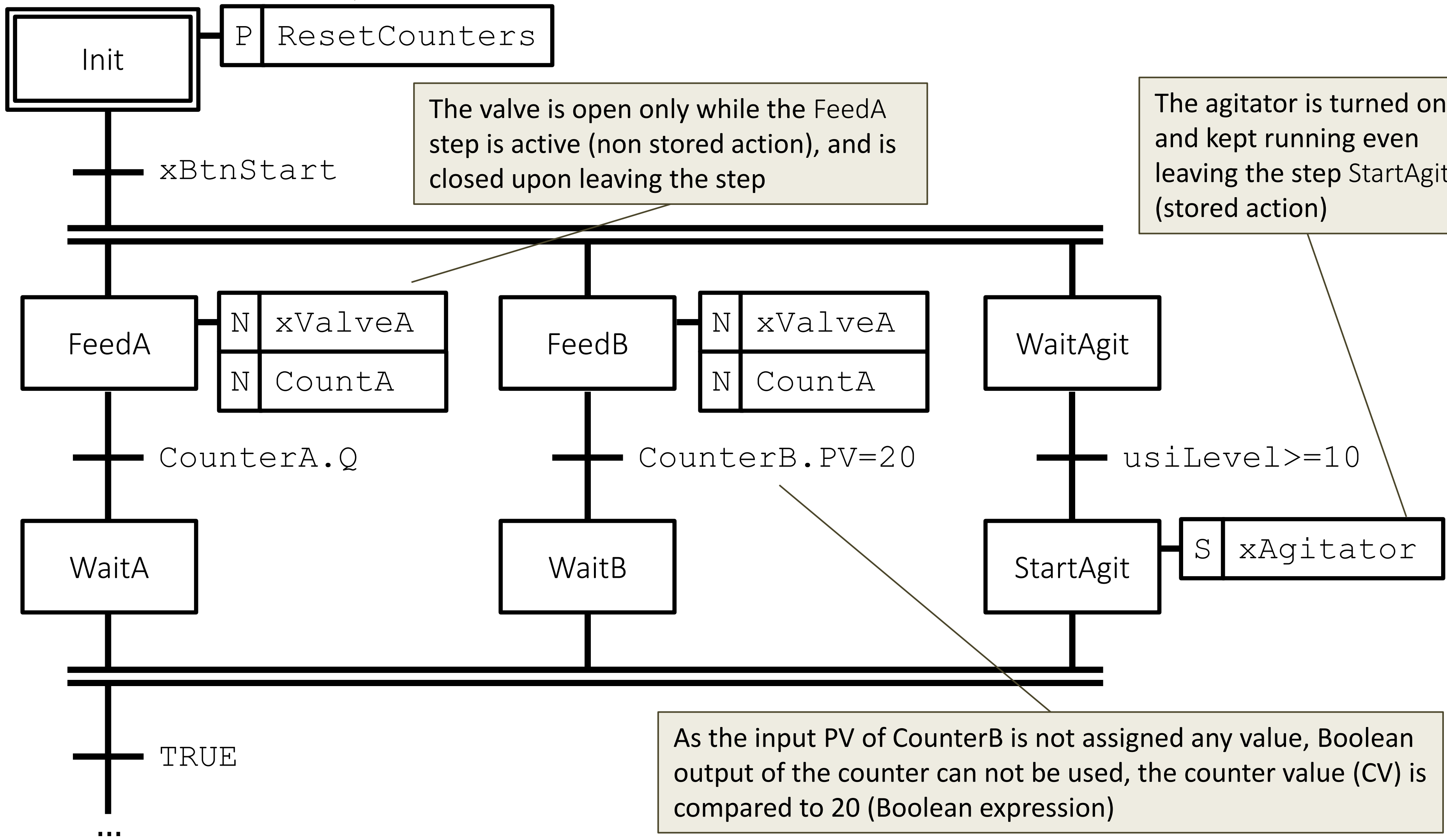
```
ACTION HeatingControl

   IF usiTemp<58
      THEN xValveSteam:=TRUE;
   ELSIF usiTemp>62
      THEN xValveSteam:=FALSE;
   END_IF

   IF NOT(_HeatingControl._x) THEN
      xValveSteam:=FALSE;
   END_IF

END_ACTION
```

Steam valve is opened if the temperature is below 58 °C, closed if the temperature is above 62 °C, kept unchanged in between (hysteresis control)

After the last execution of the action, the steam valve shall be closed (otherwise the product shall be kept heated on). An execution of the action is the last if the the action flag (`HeatingControl.Q`) is false. The action flag can be accessed with syntax `_<action>._x` in CODESYS.

# Storing the components

The action resetting the counters is executed once upon activating the Init step (pulse action)

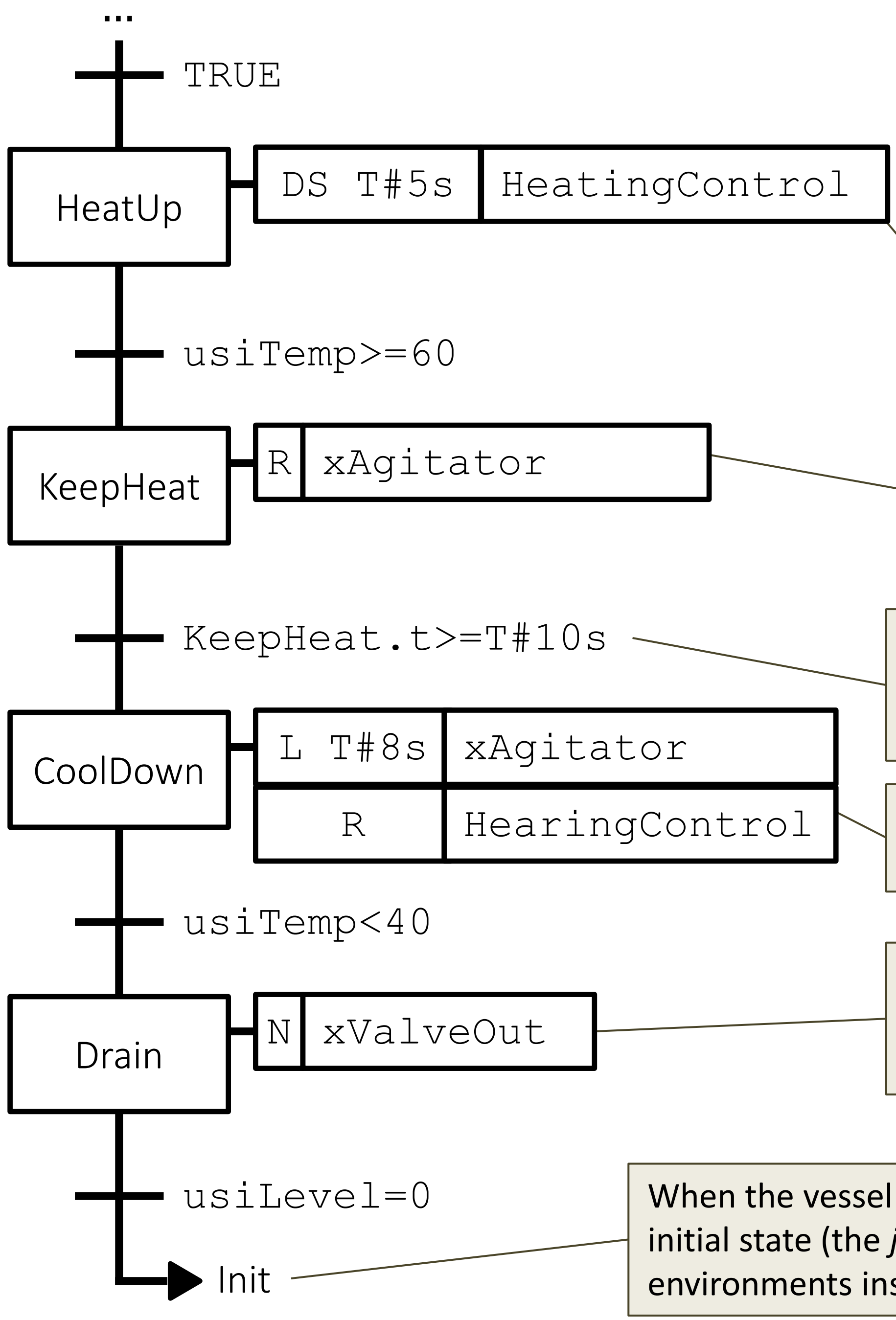


The valve is open only while the FeedA step is active (non stored action), and is closed upon leaving the step

The agitator is turned on and kept running even leaving the step StartAgit (stored action)

As the input PV of CounterB is not assigned any value, Boolean output of the counter can not be used, the counter value (CV) is compared to 20 (Boolean expression)

# Heating and draining



The action implementing temperature control is started 5 seconds after activating the step HeatUp (delayed action)

When the temperature reaches 60 °C, heating control is kept running but the agitator is turned off (reset action)

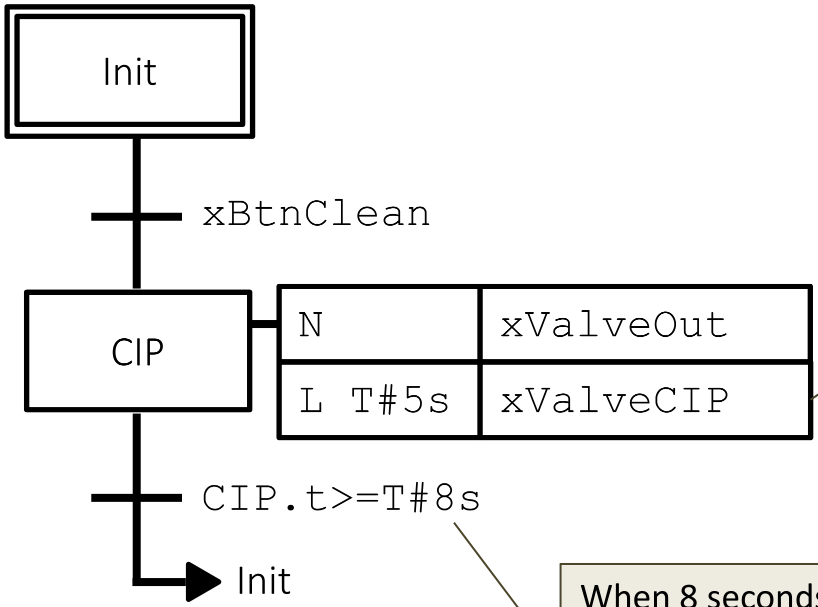
The 10 second delay (duration passed since the activation of the step KeepHeat) is evaluated by the comparison of the step time to a duration literal

The agitator runs for at most 8 seconds, but is turned off immediately if the step is deactivated (limited action)

If the temperature of the product drops under 40 °C, the outflow valve is opened. Upon the vessel becomes empty, the valve is closed (non stored action).

When the vessel is empty, the chart returns to its initial state (the *jump* notation is used by various environments instead of a bottom-up directed edge)

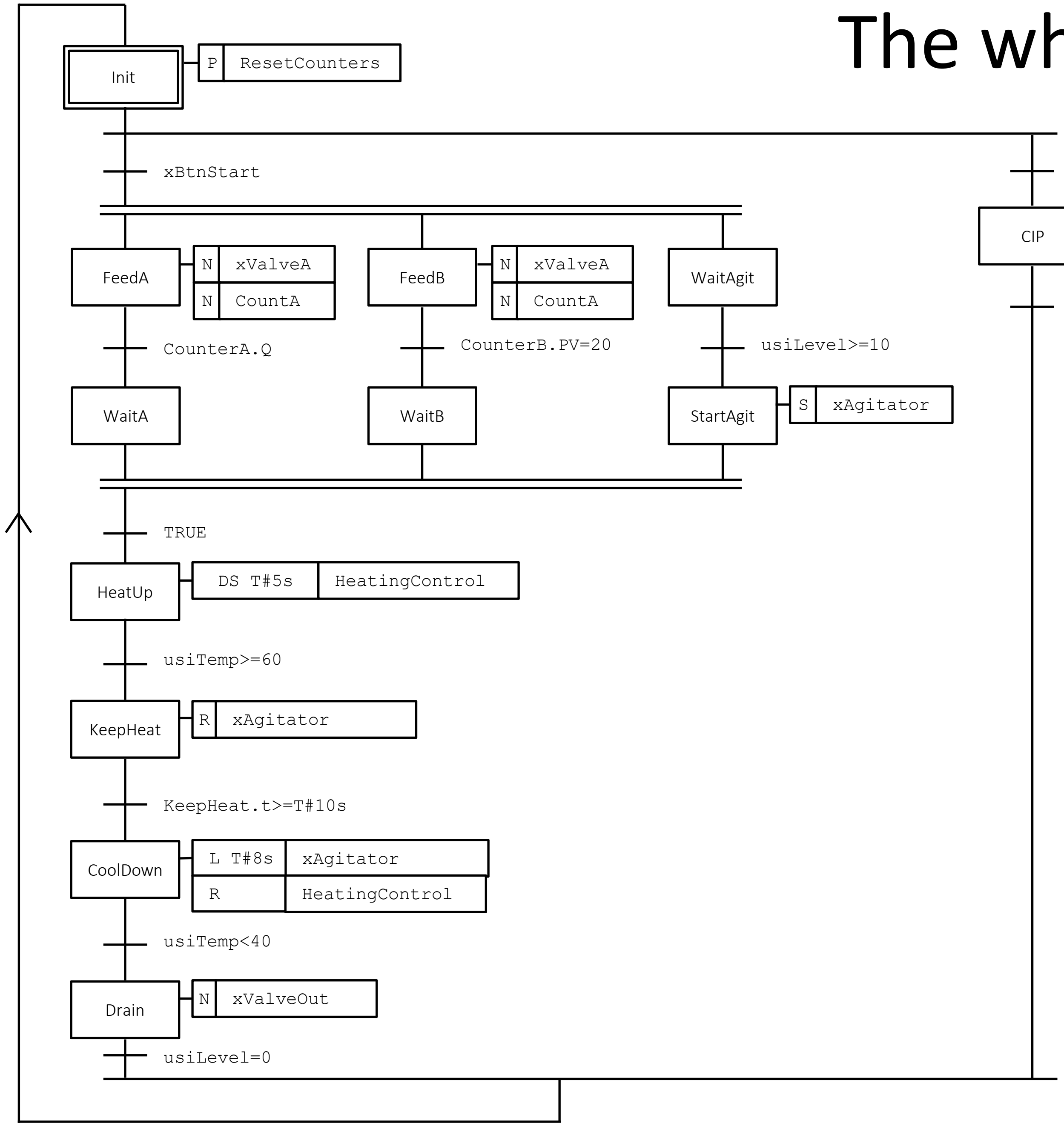
# Cleaning



After pressing the Clean button, the CIP valve is opened for 5 seconds (limited action)

When 8 seconds have passed since the activation of the step CIP, the initial step is re-entered. Then the output connected to the drain valve is deactivated, hence the drain valve is closed 3 seconds after closing the CIP valve.

# The whole program



```
VAR
   CounterA, CounterB : CTU;
END_VAR

ACTION CountA
  CounterA(CU:=xFlowA,PV:=10,
                 RESET:=FALSE);
END_ACTION

ACTION CountB
   CounterB(CU:=xFlowB,
                 RESET:=FALSE);
END_ACTION

ACTION ResetCounters
   CounterA(RESET:=TRUE);
   CounterB(RESET:=TRUE);
END_ACTION

ACTION HeatingControl

   IF usiTemp<58
      THEN xValveSteam:=TRUE;
   ELSIF usiTemp>62
      THEN xValveSteam:=FALSE;
   END_IF
   IF NOT(_HeatingControl._x) THEN
      xValveSteam:=FALSE;
   END_IF

END_ACTION
```

# Implementation in the template project

- The project already contains a program (`PLC_PRG`) with the declarations of input and output variables, and the freewheeling task executing the program
- Non-Boolean actions can be added to the program by right-clicking `PLC_PRG` in the project tree and selecting the Add Object > Action command