Contents & Goals

Last Lecture:
• DC Implementables.
• A controller for the gas burner.

This Lecture:
• Educational Objectives: Capabilities for following tasks/questions.
• What is the "philosophy" of PLC? What did we generalise/abstract them to?
• What's an example for giving a DC semantics for a constructive formalism?
• How does the proposed approach work, from requirement to a correct implementation with DC?

Content:
• Programmable Logic Controllers (PLC) ("Speicherprogrammierbare Steuerungen" (SPS))
• PLC Automata
• An over-approximating DC semantics for PLC
• An reaction time theorem for PLC

What is a PLC?

How do PLC look like?

What’s special about PLC?
• microprocessor, memory, timers
• digital (or analog) I/O ports
• possibly RS232, field buses, networking
• robust hardware
• reprogrammable
• standardised programming model (IEC61131-3)

Where are PLC employed?
• mostly process automation
• production lines
• packaging lines
• chemical plants
• powerplants
• electric motors, pneumatic or hydraulic cylinders
• ...
• not so much:
  • product automation, there
  • tailored or OTS controller boards
  • embedded controllers
  • ...

What is a PLC?
How are PLC programmed?

- PLC have in common that they operate in a cyclic manner:
  - read inputs
  - compute
  - write outputs
  - Cyclic operation is repeated until external interruption (such as shutdown or reset).
  - Cycle time: typically a few milliseconds.

Programming for PLC means providing the "compute" part.

- Input/output values are available via designated local variables.

Example: Practically

- Example: Reliable, Stutter-free trains sensor.
  - Assume a track-side sensor with outputs:
    - no tr — "no passing train"
    - tr — "a train is passing"
  - Assume that a change from "no tr" to "tr" signals arrival of a train. (No spurious sensor values.)
  - Problem: The sensor may stutter, i.e. oscillate between "no tr" and "tr" multiple times.
  - Idea: A stutter filter with outputs N and T, for "notrain" and "train passing" (and possibly X, for error).
  - After arrival of a train, ignore "no tr" for 5 seconds.

```
1: PROGRAM PLC_PRG_FILTER
2: VAR
3: state : INT := 0; (* 0:=N, 1:=T, 2:=X *)
4: tmr : TP;
5: ENDVAR
6:7: IF state = 0 THEN
8: %output := N;
9: IF %input = tr THEN
10: state := 1;
11: %output := T;
12: ELSIF %input = Error THEN
13: state := 2;
14: %output := X;
15: ENDIF
16: ELSIF state = 1 ... );
17: ELSIF %input = Error THEN
18: state := 2;
19: %output := X;
20: tmr( IN := FALSE, PT := t#0.0s );
21: ENDIF
22: ENDIF
```

Alternative Programming Languages by IEC 61131-3

- Instruction List
- ST (Structured Text)
- LD (Ladder Diagram)
- x := y OR z

Figure 2.2: Implementations of the operation "x" becomes y \lor z" [?]

Tied together by:
- Sequential Function Charts (SFC)

Unfortunate: deviations in semantics...

```
s0: step (initial)
s1: transition
s2: transition condition (guard)
s3: action block
PPPPPPPPPPP i
PPPPPPPPPPP i
```

Figure 2.3: Elements of sequential function charts [?]
Why study PLC?

Note: The discussion here is not limited to PLC and IEC 61131-3 languages. Any programming language on an operating system with at least one real-time clock will do. (Where a real-time clock is a piece of hardware such that,

• we can program it to wait for \( t \) time units,
• we can query whether that time has elapsed,
• if we program it to wait for \( t \) time units, it does so with negligible deviation.)

And strictly speaking, we don’t even need “full-blown” operating systems. PLCs are just a formalisation on a good level of abstraction:

• there are inputs somehow available as local variables,
• there are outputs somehow available as local variables,
• somehow, inputs are polled and outputs updated atomically,
• there is some interface to a real-time clock.

PLC Automata

Definition 5.2. A PLC-Automaton is a structure \( A = (Q, \Sigma, \delta, q_0, \varepsilon, S_t, S_e, \Omega, \omega) \) where

• \( (q \in \) \( Q \)) is a finite set of states, \( q_0 \in Q \) is the initial state,
• \( (\sigma \in \) \( \Sigma \)) is a finite set of inputs,
• \( \delta : Q \times \Sigma \rightarrow Q \) is the transition function (!),
• \( S_t : Q \rightarrow \mathbb{R}^+ \) assigns a delay time to each state,
• \( S_e : Q \rightarrow 2^\Sigma \) assigns a set of delayed inputs to each state,
• \( \Omega \) is a finite, non-empty set of outputs,
• \( \omega : Q \rightarrow \Omega \) assigns an output to each state,
• \( \varepsilon \) is an upper time bound for the execution cycle.

PLC Automata Example: Stuttering Filter

\[ A = (Q = \{ q_0, q_1 \}, \Sigma = \{ tr, no \cdot tr \}, \delta = \{ \langle q_0, tr \rangle \mapsto q_1, \langle q_0, no \cdot tr \rangle \mapsto q_0, \langle q_1, tr \rangle \mapsto q_1, \langle q_1, no \cdot tr \rangle \mapsto q_0 \}, q_0, \varepsilon = 0, 2, S_t = \{ q_0 \mapsto 0, q_1 \mapsto 5 \}, S_e = \{ q_0 \mapsto \emptyset, q_1 \mapsto \Sigma \}, \Omega = \{ N, T \}, \omega = \{ q_0 \mapsto N, q_1 \mapsto T \}) \]

PLC Automata Example: Stuttering Filter with Exception

\[ \begin{align*}
(\{ q_0 \} \mapsto \emptyset, \{ no \cdot tr, tr \}) & = \emptyset \\
(\{ q_0 \} \mapsto \emptyset, \{ no \cdot tr, tr \}) & = \emptyset \\
(\{ q_1 \} \mapsto \emptyset, \{ no \cdot tr, tr \}) & = \emptyset \\
(\{ q_0 \} \mapsto \emptyset, \{ no \cdot tr, tr \}) & = \emptyset \\
\end{align*} \]

PLC Automaton Semantics

Recall:

1: PROGRAM PLC_PRG_FILTER
2: VAR
3: state : INT := 0; (* 0:=N, 1:=T, 2:=X *)
4: tmr : TP;
5: ENDVAR
6: 
7: IF state = 0 THEN
8: %output := N;
9: IF %input = tr THEN
10: state := 1;
11: %output := T;
12: ELSIF %input = Error THEN
13: state := 2;
14: %output := X;
15: ENDIF
16: ELSIF state = 1 ... );
17: ELSIF %input = Error THEN
18: state := 2;
19: %output := X;
20: tmr( IN := FALSE, PT := t#0.0s );
21: ENDIF
22: ENDIF
We assess correctness in terms of cycle time...

But where does the cycle time come from?

First of all, the hardware has a cycle time... so we can measure it—if it is larger than \( \varepsilon \), don't use this program on this controller.

We can estimate (approximate) the WCET (worst-case execution time)—if it's larger than \( \varepsilon \), don't use it; if it's smaller, we're safe.

Some PLCs have a watchdog:

• set it to \( \varepsilon \),
• if the current "computing" cycle takes longer,
• then the watchdog forces the PLC into an error state and signals the error condition.

And what does this have to do with DC?

Interesting overall approach:

• Define PLC-automaton syntax (abstract and concrete).
• Define PLC-automaton semantics by translation to ST (structured text).
• Give DC over-approximation of PLC-automaton semantics.
• Assess correctness of over-approximation against DC requirements.

In other words: we'll define \( \text{DC} \) such that

\[ I \in \text{DC} \Rightarrow I | \text{DC} \]

but not necessarily the other way round.

In even other words:

\[ \text{DC} \subseteq \{ I | I | \text{DC} \} \].
One Application: Reaction Times

- Given a PLC-Automaton, one often wants to know whether it guarantees properties of the form $\left\lceil \mathcal{S}t_{A} \in Q \land \text{In}_{A} = \text{emergency signal} \right\rceil_{0.1} \rightarrow \left\lceil \mathcal{S}t_{A} = \text{motor off} \right\rceil$ ("whenever the emergency signal is observed, the PLC-Automaton switches the motor off within at most 0.1 seconds").

Which is (why?) for from obvious from the PLC-Automaton in general.

We will give a theorem, that allows us to compute an upper bound on such reaction times. Then in the above example, we could simply compare this upper bound with the required 0.1 seconds.

The Reaction Time Problem in General

- Let $\Pi \subseteq Q$ be a set of start states,
- $A \subseteq \Sigma$ be a set of inputs,
- $c \in \text{Time}$ be a time bound, and
- $\Pi_{\text{target}} \subseteq Q$ be a set of target states.

Then we seek to establish properties of the form $\left\lceil \mathcal{S}t_{A} \in \Pi \land \text{In}_{A} \in A \right\rceil_{c} \rightarrow \left\lceil \mathcal{S}t_{A} \in \Pi_{\text{target}} \right\rceil$, abbreviated as $\left\lceil \Pi \land A \right\rceil_{c} \rightarrow \left\lceil \Pi_{\text{target}} \right\rceil$.

The Reaction Time Theorem (Special Case $n = 1$)

Theorem 5.6. Let $A = (Q, \Sigma, \delta, q_{0}, \varepsilon, S_{t}, S_{e}, \Omega, \omega)$, $\Pi \subseteq Q$, and $A \subseteq \Sigma$ with $\delta(\Pi, A) \subseteq \Pi$. Then $\left\lceil \Pi \land A \right\rceil_{c} \rightarrow \left\lceil \delta(\Pi, A) \right\rceil = \Pi_{\text{target}}$ where $c := \varepsilon + \max\left\{ 0 \cup \left\{ s(\pi, A) \mid \pi \in \Pi \setminus \delta(\Pi, A) \right\} \right\}$ and $s(\pi, A) := \begin{cases} S_{t}(\pi) + 2\varepsilon, & \text{if } S_{t}(\pi) > 0 \text{ and } A \cap S_{e}(\pi) \neq \emptyset \\ \varepsilon, & \text{otherwise} \end{cases}$.

Reaction Time Theorem: Example 1

(1) $\left\lceil \{N, T\} \land \{\text{no tr}\} \right\rceil_{5+3\varepsilon} \rightarrow \left\lceil N \right\rceil$.
We can prove this by showing the following:\n\[ (x_1, x_2, x_3, x_4, x_5) \rightarrow (y_1, y_2, y_3, y_4, y_5) \]
\[ (x_1, x_2, x_3, x_4, x_5) \rightarrow (z_1, z_2, z_3, z_4, z_5) \]
\[ (x_1, x_2, x_3, x_4, x_5) \rightarrow (w_1, w_2, w_3, w_4, w_5) \]
\[ (x_1, x_2, x_3, x_4, x_5) \rightarrow (v_1, v_2, v_3, v_4, v_5) \]
\[ (x_1, x_2, x_3, x_4, x_5) \rightarrow (u_1, u_2, u_3, u_4, u_5) \]
\[ (x_1, x_2, x_3, x_4, x_5) \rightarrow (t_1, t_2, t_3, t_4, t_5) \]
\[ (x_1, x_2, x_3, x_4, x_5) \rightarrow (s_1, s_2, s_3, s_4, s_5) \]
\[ (x_1, x_2, x_3, x_4, x_5) \rightarrow (r_1, r_2, r_3, r_4, r_5) \]
\[ (x_1, x_2, x_3, x_4, x_5) \rightarrow (q_1, q_2, q_3, q_4, q_5) \]
\[ (x_1, x_2, x_3, x_4, x_5) \rightarrow (p_1, p_2, p_3, p_4, p_5) \]
\[ (x_1, x_2, x_3, x_4, x_5) \rightarrow (o_1, o_2, o_3, o_4, o_5) \]
\[ (x_1, x_2, x_3, x_4, x_5) \rightarrow (n_1, n_2, n_3, n_4, n_5) \]
\[ (x_1, x_2, x_3, x_4, x_5) \rightarrow (m_1, m_2, m_3, m_4, m_5) \]
\[ (x_1, x_2, x_3, x_4, x_5) \rightarrow (l_1, l_2, l_3, l_4, l_5) \]
\[ (x_1, x_2, x_3, x_4, x_5) \rightarrow (k_1, k_2, k_3, k_4, k_5) \]
\[ (x_1, x_2, x_3, x_4, x_5) \rightarrow (j_1, j_2, j_3, j_4, j_5) \]
\[ (x_1, x_2, x_3, x_4, x_5) \rightarrow (i_1, i_2, i_3, i_4, i_5) \]
\[ (x_1, x_2, x_3, x_4, x_5) \rightarrow (h_1, h_2, h_3, h_4, h_5) \]
\[ (x_1, x_2, x_3, x_4, x_5) \rightarrow (g_1, g_2, g_3, g_4, g_5) \]
\[ (x_1, x_2, x_3, x_4, x_5) \rightarrow (f_1, f_2, f_3, f_4, f_5) \]
\[ (x_1, x_2, x_3, x_4, x_5) \rightarrow (e_1, e_2, e_3, e_4, e_5) \]
\[ (x_1, x_2, x_3, x_4, x_5) \rightarrow (d_1, d_2, d_3, d_4, d_5) \]
\[ (x_1, x_2, x_3, x_4, x_5) \rightarrow (c_1, c_2, c_3, c_4, c_5) \]
\[ (x_1, x_2, x_3, x_4, x_5) \rightarrow (b_1, b_2, b_3, b_4, b_5) \]
\[ (x_1, x_2, x_3, x_4, x_5) \rightarrow (a_1, a_2, a_3, a_4, a_5) \]

Since the transition function is monotonic, we have shown that the desired result holds.\n
Methodology

Full DCDC
Implementable
PLC-Automata
IEC 61131-3
Binary

' Req
'Des

What does " " help us?
E.g.: What assumptions did we use?

Impl
/C2
A
/C3
DC
N
0s
T
5s
0.2s
tr
no
tr
q
0
q
1
ST

=⇒
⇐
=⇒
⇐
=⇒
synthesis
lecture
by example
(c correct?) compiler

References