Real-Time Systems

Lecture 09: PLC Automata

2013-05-29

Dr. Bernd Westphal

Albert-Ludwigs-Universität Freiburg, Germany
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 requirements to a correct implementation with DC?

- **Content:**
  - Programmable Logic Controllers (PLC)
    ("Speicherprogrammierbare Steuerungen" (SPS))
  - PLC Automata
  - An overapproximating DC semantics for PLCA
  - An reaction time theorem for PLCA
What is a PLC?
How do PLC look like?
What’s special about PLC?

- microprocessor, memory, **timers**
- digital (or analog) I/O ports
- possibly RS 232, fieldbuses, networking
- robust hardware
- reprogrammable
- **standardised programming model** (IEC 61131-3)
Where are PLC employed?

- mostly **process automatisation**
  - production lines
  - packaging lines
  - chemical plants
  - power plants
  - electric motors, pneumatic or hydraulic cylinders
  - ...

- not so much: **product automatisation**, there
  - tailored or OTS controller boards
  - embedded controllers
  - ...
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: reliable, stutter-free train 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 “no train” and “train passing” (and possibly X, for error).
After arrival of a train, ignore “no_tr” for 5 seconds.
**Example: Stutter Filter**

- **Idea:** After arrival of a train, ignore “no_tr” for 5 seconds.
How are PLC programmed, practically?

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: ENDIF
17: ELSIF state = 1 THEN
18:   tmr( IN := TRUE, PT := t#5.0s );
19:   IF (%input = no_tr AND NOT tmr.Q) THEN
20:      state := 0;
21:      tmr( IN := FALSE, PT := t#0.0s );
22:   ELSIF %input = Error THEN
23:      state := 2;
24:      %output := X;
25:      tmr( IN := FALSE, PT := t#0.0s );
26:   ENDIF
27: ENDIF
How are PLC programmed, practically?

1: PROGRAM PLC_PRG_FILTER
2: VAR
3:  state : INT := 0; (* 0:=N, 1:=T, 2:=X *)
4:  tmr : TP;
5: ENDVAR
6: IF state = 0 THEN
7:  %output := N;
8:  IF %input = tr THEN
9:    state := 1;
10:  %output := T;
11: ELSIF %input = Error THEN
12:    state := 2;
13:    %output := X;
14: ENDIF
15: ELSIF state = 1 THEN
16:  tmr( IN := TRUE, PT := t#0.0s );
17:  IF (%input = no_tr AND NOT tmr.Q) THEN
18:    state := 0;
19:    %output := N;
20:    tmr( IN := FALSE, PT := t#0.0s );
21:  ELSIF %input = Error THEN
22:    state := 2;
23:    %output := X;
24: ENDIF
25: ENDIF
26: ENDIF
Tied together by
- Sequential Function Charts (SFC)

Unfortunate: deviations in semantics... [?]
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 the set 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.

- PLC 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**,  
- \(\delta : Q \times \Sigma \to Q\) is the **transition function** (!),  
- \(S_t : Q \to \mathbb{R}_0^+\) assigns a **delay time** to each state,  
- \(S_e : Q \to 2^\Sigma\) assigns a set of **delayed inputs** to each state,  
- \(\Omega\) is a finite, non-empty set of **outputs**,  
- \(\omega : Q \to \Omega\) assigns an **output** to each state,  
- \(\varepsilon\) is an **upper time bound** for the execution cycle.
PLC Automata Example: Stuttering Filter

\[ A = (Q = \{q_0, q_1\}, \]
\[ \Sigma = \{\text{tr, no\_tr}\}, \]
\[ \delta = \{(q_0, \text{tr}) \mapsto q_1, (q_0, \text{no\_tr}) \mapsto q_0, (q_1, \text{tr}) \mapsto q_1, (q_1, \text{no\_tr}) \mapsto q_0\}, \]
\[ q_0 = q_0, \]
\[ \varepsilon = 0.2, \]
\[ S_t = \{q_0 \mapsto 0, q_1 \mapsto 5\}, \]
\[ S_e = \{q_0 \mapsto \emptyset, q_1 \mapsto \Sigma\}, \]
\[ \Omega = \{N, T\}, \]
\[ \omega = \{q_0 \mapsto N, q_1 \mapsto T\} \]

- \text{delay time}  
- \text{delayed inputs}

"in state \( q \in Q\), for time \( S_t(q) \) after entering \( q \) ignore \( S_e(q) \)"

(don't take any transition on \( S_e(q) \))
PLC Automata Example: Stuttering Filter with Exception
1: PROGRAM PLC_PRG_FILTER
2: VAR
3:   state : INT := 0; (* 0:=N, 1:=T, 2:=X *)
4:   tmr : TP;
5: ENDVAR
6: IF state = 0 THEN
7:   %output := N;
8:   IF %input = tr THEN
9:     state := 1;
10:    %output := T;
11:   ELSIF %input = Error THEN
12:     state := 2;
13:    %output := X;
14: ENDIF
15: ELSIF state = 1 THEN
16:   tmr( IN := TRUE, PT := t#5.0s );
17:   IF (%input = no_tr AND NOT tmr.Q) THEN
18:     state := 0;
19:     tmr( IN := FALSE, PT := t#0.0s );
20:   ELSIF %input = Error THEN
21:     state := 2;
22:    %output := X;
23:    tmr( IN := FALSE, PT := t#0.0s );
24: ENDIF
25: ENDIF
PLCA Semantics: Examples

```plaintext
1: PROGRAM PLC_PRG_FILTER
2: VAR
3:  state : INT := 0; (* 0:=N, 1:=T, 2:=X *)
4:  tmr : TP;
5: ENDVAR
6: IF state = 0 THEN
7:  %output := N;
8:  IF %input = tr THEN
9:     state := 1;
10:    %output := T;
11:  ELSIF %input = Error THEN
12:     state := 2;
13:     %output := X;
14: ENDIF
15: ENDIF
16: ELSIF state = 1 THEN
17:  tmr( IN := TRUE, PT := t#0.0s );
18:  IF (%input = no_tr AND NOT tmr.Q) THEN
19:    state := 0;
20:    %output := N;
21:  ELSE %input = Error THEN
22:     state := 2;
23:     %output := Error;
24:     tmr( IN := FALSE, PT := t#0.0s );
25: ELSIF %input = Error THEN
26:     state := 2;
27:     %output := X;
28:     tmr( IN := FALSE, PT := t#0.0s );
29: ENDIF
30: ENDIF
```
We assess correctness in terms of cycle time...

...but where does the cycle time come from?

- First of all, ST on 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
    (Major obstacle: caches, out-of-order execution.)

- Some PLC 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 with DC?
Wait, what is the Plan?

<table>
<thead>
<tr>
<th>Full DC</th>
<th>DC Implementables</th>
<th>PLC-Automata</th>
<th>IEC 61131-3</th>
<th>Binary</th>
</tr>
</thead>
</table>

- 'Req'
- 'Des'
- 'Impl'

\[
[A]_{DC} \quad \text{today}
\]

we have to check that \( t_{\text{max}} \leq \epsilon \)

(by example)

ST

(correct?) compiler
An Overapproximating DC Semantics for PLC Automata
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 $\llbracket \mathcal{A} \rrbracket_{DC}$ such that

  \[
  \forall I \in \llbracket \mathcal{A} \rrbracket \quad \Rightarrow \quad I \models \llbracket \mathcal{A} \rrbracket_{DC}
  \]

  but not necessarily the other way round.

- **In even other words:** $\llbracket \mathcal{A} \rrbracket \subseteq \{ I \mid I \models \llbracket \mathcal{A} \rrbracket_{DC} \}$. 
Observables

- Consider

\[ \mathcal{A} = (Q, \Sigma, \delta, q_0, \varepsilon, S_t, S_e, \Omega, \omega). \]

- The DC formula \( \mathbb{A}_{DC} \) we construct ranges over the observables
  - \( \operatorname{In}_A, \mathcal{D}(\operatorname{In}_A) = \Sigma \) — values of the inputs
  - \( \operatorname{St}_A, \mathcal{D}(\operatorname{St}_A) = Q \) — current local state
  - \( \operatorname{Out}_A, \mathcal{D}(\operatorname{Out}_A) = \Omega \) — values of the outputs
Overview

\[ A = (Q, \Sigma, \delta, q_0, \varepsilon, S_t, S_e, \Omega, \omega) \]

- **Initial State:**
  \[ [] \lor [q_0] ; true \] (DC-1)

- **Effect of Transitions, untimed:**
  \[ [\neg q] ; [q \land A] \longrightarrow [q \lor \delta(q, A)] \] (DC-2)

- **Cycle time:**
  \[ [q \land A] \xrightarrow{\varepsilon} [q \lor \delta(q, A)] \] (DC-3)

- **Delays:**
  \[ S_t(q) > 0 \implies [\neg q] ; [q \land A] \xrightarrow{\leq S_t(q)} [q \lor \delta(q, A \setminus S_e(q))] \] (DC-4)
  \[ S_t(q) > 0 \implies [\neg q] ; [q] ; [q \land A]^{\varepsilon} \xrightarrow{\leq S_t(q)} [q \lor \delta(q, A \setminus S_e(q))] \] (DC-5)

- **Additional Notes:**
  - \( A \) arbitrary with \( \emptyset \neq A \subseteq \Sigma \),
  - \([q \land A] \) abbreviates \([St_A = q \land \text{In}_A \in A] \),
  - \( \delta(q, A) \) abbreviates \( St_A \in \{\delta(q, a) \mid a \in A\} \).
Overview

\[ A = (Q, \Sigma, \delta, q_0, \varepsilon, S_t, S_e, \Omega, \omega) \]

- Progress from non-delayed inputs:

\[
S_t(q) = 0 \land q \notin \delta(q, A) \implies \Box([q \land A] \implies \ell < 2\varepsilon) \quad \text{(DC-6)}
\]

\[
S_t(q) = 0 \land q \notin \delta(q, A) \implies \neg q ; [q \land A]^{\varepsilon} \rightarrow \neg q \quad \text{(DC-7)}
\]

- Progress from delayed inputs:

\[
S_t(q) > 0 \land q \notin \delta(q, A)
\]

\[
\implies \Box([q]^{S_t(q)} ; [q \land A] \implies \ell < S_t(q) + 2\varepsilon) \quad \text{(DC-8)}
\]

\[
S_t(q) > 0 \land A \cap S_e(q) = \emptyset \land q \notin \delta(q, A)
\]

\[
\implies \Box([q \land A] \implies \ell < 2\varepsilon) \quad \text{(DC-9)}
\]

\[
S_t(q) > 0 \land A \cap S_e(q) = \emptyset \land q \notin \delta(q, A)
\]

\[
\implies \neg q ; [q \land A]^{\varepsilon} \rightarrow \neg q \quad \text{(DC-10)}
\]
Effect of Transitions, untimed

\[ [q] ; [q \land A] \rightarrow [q \lor \delta(q, A)] \] (DC-2)

\begin{array}{|c|c|c|c|c|}
\hline
[q_1 \land A] & \text{holds in} & \text{with input} & \text{After} & \text{state} & \text{output} \\
\hline
[t_0, t_1] & A = \{\text{no\_tr}\} & t_1 & \{q_1\} & \{N\} \\
[t_0, t_2] & A = \{\text{no\_tr}, \text{tr}\} & t_2 & \{q_1\} & \{N, T\} \\
[t_0, t_3] & A = \{\text{no\_tr}, \text{tr}\} & t_3 & \{q_1, q_2\} & \{N, T\} \\
[t_0, t_4] & A = \{\text{no\_tr}, \text{tr}\} & t_4 & \{q_1, q_2\} & \{N, T\} \\
[t_0, t_5] & A = \{\text{no\_tr}, \text{tr}, \text{Error}\} & t_5 & \{q_1, q_2, q_3\} & \{N, T, X\} \\
[t_0, t_6] & A = \{\text{no\_tr}, \text{tr}, \text{Error}\} & t_6 & \{q_1, q_2, q_3\} & \{N, T, X\} \\
\hline
\end{array}
Cycle Time

\([q \land A] \xrightarrow{\varepsilon} [q \lor \delta(q, A)]\)  \((DC-3)\)

<table>
<thead>
<tr>
<th>[q_1 \land A] holds in</th>
<th>with input</th>
<th>After</th>
<th>state</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>([t_1, t_2]) A = {\text{no_tr, tr}}</td>
<td>t_2</td>
<td>{q_1, q_2}</td>
<td>{N, T}</td>
<td></td>
</tr>
<tr>
<td>([t_2, t_3]) A = {\text{no_tr, tr}}</td>
<td>t_3</td>
<td>{q_1, q_2}</td>
<td>{N, T}</td>
<td></td>
</tr>
<tr>
<td>([t_3, t_4]) A = {\text{no_tr}}</td>
<td>t_4</td>
<td>{q_1}</td>
<td>{N}</td>
<td></td>
</tr>
<tr>
<td>([t_4, t_5]) A = {\text{no_tr, Error}}</td>
<td>t_5</td>
<td>{q_1, q_3}</td>
<td>{N, X}</td>
<td></td>
</tr>
<tr>
<td>([t_5, t_6]) A = {\text{Error}}</td>
<td>t_6</td>
<td>{q_1, q_3}</td>
<td>{N, X}</td>
<td></td>
</tr>
</tbody>
</table>
### Delays

![Diagram showing states, inputs, and outputs with time intervals]

\[
S_t(q) > 0 \implies [\neg q] ; [q \land A] \xrightarrow{\leq S_t(q)} [q \lor \delta(q, A \setminus S_e(q))] \quad \text{(DC-4)}
\]

<table>
<thead>
<tr>
<th>[q_1 \land A] holds in</th>
<th>with input</th>
<th>After</th>
<th>state</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>[t_0, t_1]</td>
<td>(A = {\text{no_tr}})</td>
<td>(t_1)</td>
<td>({q_2})</td>
<td>({T})</td>
</tr>
<tr>
<td>[t_0, t_2]</td>
<td>(A = {\text{no_tr}, \text{tr}})</td>
<td>(t_2)</td>
<td>({q_2})</td>
<td>({T})</td>
</tr>
<tr>
<td>[t_0, t_3]</td>
<td>(A = {\text{no_tr}, \text{tr}, \text{Error}})</td>
<td>(t_3)</td>
<td>({q_2, q_3})</td>
<td>({T, X})</td>
</tr>
<tr>
<td>[t_0, t_4]</td>
<td>(A = {\text{no_tr}, \text{tr}, \text{Error}})</td>
<td>(t_4)</td>
<td>({q_2, q_3})</td>
<td>({T, X})</td>
</tr>
<tr>
<td>[t_0, t_5]</td>
<td>(A = {\text{no_tr}, \text{tr}, \text{Error}})</td>
<td>(t_5)</td>
<td>({q_2, q_3})</td>
<td>({T, X})</td>
</tr>
<tr>
<td>[t_0, t_6]</td>
<td>(A = {\text{no_tr}, \text{tr}, \text{Error}})</td>
<td>(t_6)</td>
<td>({q_2, q_3})</td>
<td>({T, X})</td>
</tr>
</tbody>
</table>
$S_t(q) > 0 \implies [\neg q] ; [q] ; [q \land A]^\epsilon \xrightarrow{\leq S_t(q)} [q \lor \delta(q, A \setminus S_e(q))]$ (DC-5)

<table>
<thead>
<tr>
<th>$[q_1 \land A]$ holds in</th>
<th>with input</th>
<th>After</th>
<th>state</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[t_1, t_2]$</td>
<td>$A = {\text{no_tr, tr}}$</td>
<td>$t_2$</td>
<td>${q_2}$</td>
<td>${T}$</td>
</tr>
<tr>
<td>$[t_2, t_3]$</td>
<td>$A = {\text{tr, Error}}$</td>
<td>$t_3$</td>
<td>${q_2, q_3}$</td>
<td>${T, X}$</td>
</tr>
<tr>
<td>$[t_3, t_4]$</td>
<td>$A = {\text{no_tr, Error}}$</td>
<td>$t_4$</td>
<td>${q_2, q_3}$</td>
<td>${T, X}$</td>
</tr>
<tr>
<td>$[t_4, t_5]$</td>
<td>$A = {\text{no_tr}}$</td>
<td>$t_5$</td>
<td>${q_2}$</td>
<td>${T}$</td>
</tr>
<tr>
<td>$[t_5, t_6]$</td>
<td>$A = {\text{no_tr, Error}}$</td>
<td>$t_6$</td>
<td>${q_2, q_3}$</td>
<td>${T, X}$</td>
</tr>
</tbody>
</table>
Progress from non-delayed inputs

$$S_t(q) = 0 \land q \notin \delta(q, A) \implies \square([q \land A] \implies \ell < 2\varepsilon) \quad (DC-6)$$

$$S_t(q) = 0 \land q \notin \delta(q, A) \implies \neg q ; [q \land A]^\varepsilon \implies \neg q \quad (DC-7)$$

- Due to (DC-6):
  - $t_5 - t_4 < 2\varepsilon$
  - $t_3 - t_2 < 2\varepsilon$

- Due to (DC-7):
  - $t_1 - t_0 < \varepsilon$
Progress from delayed inputs

\[ S_t(q) > 0 \land q \notin \delta(q, A) \]
\[ \implies \square([q]^{S_t(q)} ; [q \land A] \implies \ell < S_t(q) + 2\varepsilon) \]  
(DC-8)

\[ S_t(q) > 0 \land A \cap S_e(q) = \emptyset \land q \notin \delta(q, A) \]
\[ \implies \square([q \land A] \implies \ell < 2\varepsilon) \]  
(DC-9)

\[ S_t(q) > 0 \land A \cap S_e(q) = \emptyset \land q \notin \delta(q, A) \]
\[ \implies \lceil \neg q \rceil ; [q \land A]^\varepsilon \rightarrow \lceil \neg q \rceil \]  
(DC-10)

- Due to (DC-8):  
  - \( t_5 - t_4 < 2\varepsilon \)
- Due to (DC-9):  
  - \( t_3 - t_2 < 2\varepsilon \)
- Due to (DC-10):  
  - \( t_1 - t_0 < \varepsilon \)
Behaviour of the Output and System Start

\[ \square([q] \implies [\omega(q)]) \]  \hspace{1cm} (DC-11) 

\[ [q_0 \land A] \xrightarrow{0} [q_0 \lor \delta(q_0, A)] \]  \hspace{1cm} (DC-2') 

\[ S_t(q_0) > 0 \implies [q_0 \land A] \xrightarrow{\leq S_t(q_0)} 0 [q_0 \lor \delta(q_0, A \setminus S_e(q_0))] \]  \hspace{1cm} (DC-4') 

\[ S_t(q_0) > 0 \implies [q_0] ; [q_0 \land A]^\varepsilon \xrightarrow{\leq S_t(q_0)} 0 [q_0 \lor \delta(q_0, A \setminus S_e(q_0))] \]  \hspace{1cm} (DC-5') 

\[ S_t(q_0) = 0 \land q_0 \notin \delta(q_0, A) \implies [q_0 \land A]^\varepsilon \xrightarrow{0} [\neg q_0] \]  \hspace{1cm} (DC-7') 

\[ S_t(q_0) > 0 \land A \cap S_e(q_0) = \emptyset \land q_0 \notin \delta(q_0, A) \implies [q_0 \land A]^\varepsilon \xrightarrow{0} [\neg q_0] \]  \hspace{1cm} (DC-10')
Definition 5.3.
The **Duration Calculus semantics** of a PLC Automaton \(\mathcal{A}\) is

\[
\llbracket \mathcal{A} \rrbracket_{DC} := \bigwedge_{q \in Q, \emptyset \neq A \subseteq \Sigma} \text{DC-1} \land \cdots \land \text{DC-11} \land \text{DC-2}' \land \text{DC-4}'
\]

\[
\land \text{DC-5}' \land \text{DC-7}' \land \text{DC-10}'.
\]

Claim:

- Let \(P_{\mathcal{A}}\) be the ST program semantics of \(\mathcal{A}\).
- Let \(\pi\) be a recording over time of then inputs, local states, and outputs of a PLC device running \(P_{\mathcal{A}}\).
- Let \(I_{\pi}\) be an encoding of \(\pi\) as an interpretation of \(\text{In}_{\mathcal{A}}, \text{St}_{\mathcal{A}}, \text{Out}_{\mathcal{A}}\).
- Then \(I_{\pi} \models \llbracket \mathcal{A} \rrbracket_{DC}\).
- But not necessarily the other way round.
One Application: Reaction Times
One Application: Reaction Times

- Given a PLC-Automaton, one often wants to know whether it guarantees properties of the form

\[ \left[ \text{St}_A \in Q \land \text{ln}_A = \text{emergency\_signal} \right] \xrightarrow{0.1} \left[ \text{St}_A = \text{motor\_off} \right] \]

("whenever the emergency signal is observed, the PLC Automaton switches the motor off within at most 0.1 seconds")

- Which is \textit{(why?)} 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 one against the required 0.1 seconds.
The Reaction Time Problem in General

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

- Then we seek to establish properties of the form
  \[
  \left[ \text{St}_A \in \Pi \land \text{In}_A \in A \right] \xrightarrow{c} \left[ \text{St}_A \in \Pi_{target} \right],
  \]
  abbreviated as
  \[
  \left[ \Pi \land A \right] \xrightarrow{c} \left[ \Pi_{target} \right].
  \]
Reaction Time Theorem Premises

- Actually, the reaction time theorem addresses **only** the special case

\[
\left[ \Pi \land A \right] \xrightarrow{c_n} \left[ \delta^n(\Pi, A) \right] = \Pi_{\text{target}}
\]

for PLC Automata with

\[\delta(\Pi, A) \subseteq \Pi.\]

- Where the transition function is canonically **extended** to **sets** of start states and inputs:

\[\delta(\Pi, A) := \{ \delta(q, a) \mid q \in \Pi \land a \in A \}.\]
Examples:

- $\Pi = \{N, T\}$, $A = \{\text{no_tr}\}$
  - $\delta(\Pi, A) = \{N\} \subseteq \Pi$

- $\Pi = \{N, T, X\}$, $A = \{\text{Error}\}$
  - $\delta(\Pi, A) = \{X\} \subseteq \Pi$

- $\Pi = \{T\}$, $A = \{\text{no_tr}\}$
  - $\delta(\Pi, A) = \{N\} \not\subseteq \Pi$
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

\[
\left[ \Pi \land A \right] \xrightarrow{c} \left[ \delta(\Pi, A) \right] = \Pi_{target}
\]

where

\[
c := \varepsilon + \max(\{0\} \cup \{s(\pi, A) \mid \pi \in \Pi \setminus \delta(\Pi, A)\})
\]

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) \([\{N,T\} \land \{\text{no_tr}\}] \xrightarrow{5+3\varepsilon} [N]::\)
Reaction Time Theorem: Example 2

(2) $[\{N, T, X\} \land \{\text{Error}\}] \xrightarrow{2\varepsilon} [X]$: 
Monotonicity of Generalised Transition Function

- Define
  \[ \delta^0(\Pi, A) := \Pi, \quad \delta^{n+1}(\Pi, A) := \delta(\delta^n(\Pi, A), A). \]

- If we have \( \delta(\Pi, A) \subseteq \Pi \), then we have
  \[ \delta^{n+1}(\Pi, A) \subseteq \delta^n(\Pi, A) \subseteq \cdots \subseteq \delta(\delta(\Pi, A), A) \subseteq \delta(\Pi, A) \subseteq \Pi \]
  i.e. the sequence is a contraction.

- (Because the extended transition function has the following (not so surprising) monotonicity property):

  \[ \Pi \subseteq \Pi' \subseteq Q \text{ and } A \subseteq A' \subseteq \Sigma \text{ implies } \delta(\Pi, A) \subseteq \delta(\Pi', A'). \]

  **Proposition 5.4.**
Contraction Examples

Examples:

1. \( \Pi = \{N, T\}, A = \{\text{no\_tr}\} \)
   - \( \delta^0(\Pi, A) = \{N, T\} \)
   - \( \delta(\delta^0(\Pi, A), A) = \{N\} \subseteq \Pi \)
   - \( \delta^n(\delta^0(\Pi, A), A) = \{N\} \)

2. \( \Pi = \{N, T, X\}, A = \{\text{Error}\} \)
   - \( \delta^0(\Pi, A) = \{N, T, X\} \)
   - \( \delta(\delta^0(\Pi, A), A) = \{X\} \subseteq \Pi \)
   - \( \delta^n(\delta^0(\Pi, A), A) = \{X\} \)

3. \( \Pi = \{T\}, A = \{\text{no\_tr}\} \)
   - \( \delta(\Pi, A) = \{N\} \not\subseteq \Pi \)
Theorem 5.8. 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 for all $n \in \mathbb{N}_0$,

$$[\Pi \land A] \xrightarrow{c_n} \left[\delta^n(\Pi, A)\right] = \Pi_{\text{target}}$$

where

$$c_n := \varepsilon + \max\left(\{0\} \cup \sum_{i=1}^k s(\pi_i, A) \bigg| \begin{array}{c} 1 \leq k \leq n \land \exists \pi_1, \ldots, \pi_k \in \Pi \setminus \delta^n(\Pi, A) \land \forall j \in \{1, \ldots, k - 1\} : \\
\pi_{j+1} \in \delta(\pi_j, A) \end{array} \right)$$

and $s(\pi, A)$ as before.
Proof Idea of Reaction Time Theorem

(by contradiction)

• Assume, we would not have

\[ [\Pi \land A] \xrightarrow{c_n} [\delta^n(\Pi, A)]. \]

• This is equivalent to not having

\[ \neg (true ; [\Pi \land A]^{c_n} ; [\neg \delta^n(\Pi, A)] ; true) \]

• Which is equivalent to having

\[ true ; [\Pi \land A]^{c_n} ; [\neg \delta^n(\Pi, A)] ; true. \]

• Using finite variability, (DC-2), (DC-3), (DC-6), (DC-7), (DC-8), (DC-9), and (DC-10) we can show that the duration of \([\Pi \land A]\) is strictly smaller than \(c_n\).
Methodology: Overview
Methodology

<table>
<thead>
<tr>
<th>Full DC</th>
<th>DC Implementables</th>
<th>PLC-Automata</th>
<th>IEC 61131-3</th>
<th>Binary</th>
</tr>
</thead>
</table>

What does “◦” help us?
E.g.: What assumptions did we use?

by example

ST

(correct?) compiler
References