Real-Time Systems

Lecture 09: PLC Automata

2012-06-14

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:**
  - Continue Implementables Example
  - Programmable Logic Controllers (PLC) (“Speicherprogrammierbare Steuerungen” (SPS))
  - PLC Automata
  - An overapproximating DC semantics for PLCA
  - An reaction time theorem for PLCA
Example: Gas Burner
Recall: Control Automata

Model of Gas Burner controller as a system of four control automata:

- \( H \) : Boolean, representing **heat request**, (input)
- \( F \) : Boolean, representing **flame**, (input)
- \( C \) with \( \mathcal{D}(C) = \{ \text{idle, purge, ignite, burn} \} \), representing the **controller**, (local)
- \( G \) : Boolean, representing **gas valve**, (output)
Gas Burner Controller Specification

\[
\begin{align*}
\mathbf{Init-1} - 4: & \quad [\neg H] \lor [\neg F] \lor [\neg G]; true \\
\mathbf{Seq-1}: & \quad [\neg G]; true \\
\mathbf{Seq-2}: & \quad [\neg H]; true \\
\mathbf{Seq-3}: & \quad [\neg F]; true \\
\mathbf{Seq-4}: & \quad [\neg H] \lor [\neg F]; true
\end{align*}
\]

\[
\begin{align*}
\mathbf{Init-1} - 4: & \quad [idle] \rightarrow [idle \lor purge] \\
\mathbf{Seq-1}: & \quad [purge] \rightarrow [purge \lor ignite] \\
\mathbf{Seq-1}: & \quad [ignite] \rightarrow [ignite \lor burn] \\
\mathbf{Seq-1}: & \quad [burn] \rightarrow [burn \lor idle] \\
\mathbf{Prog-1}: & \quad [purge] \rightarrow [\neg purge] \\
\mathbf{Prog-2}: & \quad [ignite] \rightarrow [\neg ignite] \\
\mathbf{Syn-1}: & \quad [idle] \rightarrow [\neg idle] \\
\mathbf{Syn-2}: & \quad [burn \land (\neg H \lor \neg F)] \rightarrow [\neg burn] \\
\mathbf{Syn-3}: & \quad [G \land (idle \lor purge)] \rightarrow [\neg G] \\
\mathbf{Stab-1}: & \quad [\neg G \land (\neg \neg H \lor \neg F); true \lor \neg G] \rightarrow [G] \\
\mathbf{Stab-2}: & \quad [idle] \lor [idle \land \neg H] \rightarrow [idle] \\
\mathbf{Stab-3}: & \quad [\neg idle] \lor [idle \land \neg H] \rightarrow [idle] \\
\mathbf{Stab-4}: & \quad [\neg purge] \lor [purge] \rightarrow [\neg purge] \\
\mathbf{Stab-5}: & \quad [\neg ignite] \lor [ignite] \rightarrow [\neg ignite] \\
\mathbf{Stab-6}: & \quad [\neg burn] \lor [burn \land H \land F] \rightarrow [burn] \\
\mathbf{Stab-5}: & \quad [\neg F] \lor [\neg F \land \neg ignite] \rightarrow [\neg F] \\
\mathbf{Stab-7}: & \quad [\neg F \land \neg ignite] \rightarrow [\neg F] \\
\mathbf{Stab-6}: & \quad [\neg G] \lor [\neg G \land (idle \lor purge)] \rightarrow [\neg G] \\
\mathbf{Stab-6}: & \quad [\neg G] \lor [\neg G \land (idle \lor purge)] \rightarrow [\neg G] \\
\mathbf{Stab-7}: & \quad [\neg G] \lor [G \land (\neg idle \lor \neg purge)] \rightarrow [G]
\end{align*}
\]
Gas Burner Controller Correctness Proof

\[ \text{GB-Ctrl} \equiv \text{Init-1} \land \cdots \land \text{Stab-7} \land \varepsilon > 0 \]

Recall:

\[ \text{Req} \iff \Box (\ell \geq 60 \implies 20 \cdot \int L \leq \ell) \]

and (cf. [Olderog and Dierks, 2008])

\[ \models \text{Req-1} \implies \text{Req} \]

for the simplified

\[ \text{Req-1} :\equiv \Box (\ell \leq 30 \implies \int L \leq 1) \]

Here we show

\[ \models \text{GB-Ctrl} \land A(\varepsilon) \implies \text{Req-1} . \]
Lemma 3.15

\[ \models \text{GB-Ctrl} \implies \Box \left( \begin{array}{l}
(\lceil \text{idle} \rceil \implies \int G \leq \varepsilon) \\
\land (\lceil \text{purge} \rceil \implies \int G \leq \varepsilon) \\
\land (\lceil \text{ignite} \rceil \implies \ell \leq 0.5 + \varepsilon) \\
\land (\lceil \text{burn} \rceil \implies \int \neg F \leq 2\varepsilon)
\end{array} \right) \]

\[ \triangleright I, (b, e) \vdash \Gamma_{\text{burn}} \]

\[ (\text{Synt-2}) \quad \Gamma_{\text{burn}} \land (\Gamma_{\neg H \land \neg F}) \Gamma_{\text{burn}} \]

\[ (\text{Synt-5}) \quad \Gamma_{F - l}; \Gamma_{F \land \neg \text{ignite}} \rightarrow \Gamma_{F - l} \]

\[ I, (b, e) \vdash \Box (\Gamma_{F - l} \Rightarrow \ell \leq \varepsilon) \]

\[ \land \neg \Box (\Gamma_{F - l}; \Gamma_{F - l}; \Gamma_{F - l}) \]
Lemma 3.16

\[ \models \exists \varepsilon \cdot GB-Ctrl \implies \square (\ell \leq 30 \implies \int L \leq 1) \]

**Proof Sketch:**

Choose \( I, V, [6, \varepsilon] \) s.t. \( I, V, [6, \varepsilon] \models GB-Ctrl \wedge \ell \leq 30 \).

Distinguish 5 cases:

1. \( I, V, [5, \varepsilon] \models \top \)
2. \( \forall (\text{fill}) ; \text{true} \wedge \ell \leq 30 \)
3. \( \forall (\text{ign} 2) ; \text{true} \wedge \ell \leq 30 \)
4. \( \forall (\text{ign} 3) ; \text{true} \wedge \ell \leq 30 \)
5. \( \forall (\text{ign} 4) ; \text{true} \wedge \ell \leq 30 \)
Lemma 3.16 Cont’d

- Case 0: $\mathcal{I}, \mathcal{V}, [b, e] \models \top$

- Case 1: $\mathcal{I}, \mathcal{V}, [b, e] \models [idle] ; true \wedge \ell \leq 30$

\[
[\text{idle}] \longrightarrow [\text{idle} \lor \text{purge}]
\]  \hspace{1cm} (Seq-1)

\[
[\neg \text{purge}] ; [\text{purge}] \xrightarrow{\leq 30} [\text{purge}]
\]  \hspace{1cm} (Stab-2)

Thus $[\exists \leq 0.5]$ is sufficient for Reg-1 in this case.
Lemma 3.16 Cont’d

- Case 2: \( I, \mathcal{V}, [b, e] \models [\text{burn}] ; \text{true} \land \ell \leq 30 \)

\[
[burn] \rightarrow [burn \lor \text{idle}] \quad \text{(Seq-4)}
\]
Case 3: \( \mathcal{I}, \mathcal{V}, [b, e] \models [\text{ignite}] \land true \land \ell \leq 30 \)

\[
[\text{ignite}] \rightarrow [\text{ignite} \lor \text{burn}] \quad \text{(Seq-3)}
\]

So \( \exists \leq 0.1 \) sufficient for Req. 1.
Lemma 3.16 Cont’d

• Case 4: $\mathcal{I}, \mathcal{V}, [b, e] \models [\text{purge}] ; \text{true} \land \ell \leq 30$

\[
\begin{align*}
&\rightarrow [\text{purge} \lor \text{ignite}] \\
\mathcal{I}, \mathcal{V}, [b, e] &\models [\mathcal{L} \leq \varepsilon \lor [\text{purge}] ; \text{ignite} ; \text{true}]
\end{align*}
\]

\[
\begin{align*}
(&35, \alpha) &\rightarrow [\mathcal{L} \leq 3 \lor \varepsilon ; \mathcal{L} \leq 0.5 + 5\varepsilon] \\
&\mathcal{I}, \mathcal{V}, [b, e] &\models \mathcal{L} \leq 0.5 + 6\varepsilon
\end{align*}
\]

Thus $\left[\varepsilon \leq \frac{1}{12}\right]$ is sufficient for Eq. 1 in this case.
**Correctness Result**

Theorem 3.17.

\[ \models \left( \text{GB-Ctrl} \land \varepsilon \leq \frac{1}{12} \right) \implies \text{Req} \]

Diagram:

<table>
<thead>
<tr>
<th>State</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>purge</td>
<td>( \leq 3 )</td>
</tr>
<tr>
<td>ignite</td>
<td>( \geq 3 )</td>
</tr>
<tr>
<td>burn</td>
<td>( \leq 2\varepsilon )</td>
</tr>
<tr>
<td>idle</td>
<td>( \leq \varepsilon )</td>
</tr>
<tr>
<td>purge</td>
<td>( \geq 3 )</td>
</tr>
</tbody>
</table>

\( \Delta \varepsilon \leq 0.5 + 3\varepsilon \) in the worst case

\( \Delta A(\varepsilon) \ni \varepsilon \leq \frac{1}{12} \)
Discussion

- We used only
  

What about

\[ \text{Prog-1} = [\text{purge}]^{30+\varepsilon} \rightarrow [\neg\text{purge}] \]

for instance?

\[ \text{Idle} \rightarrow \text{Idle} \]

Np, there is the requirement (not explicitly asked down) that the system does something finally, e.g. get the heating going on again.
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:

  - Cyclic operation is repeated until external interruption (such as shutdown or reset).
  - Cycle time: typically a few milliseconds. [Lukoschus, 2004]
  - Programming for PLC means providing the "compute" part.
  - Input/output values are available via designated local variables.

![Cyclic Operation Diagram](image-url)
How are PLC programmed, practically?

- **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 : PT;
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: ELSE IF state = 1 THEN
17:     tmr( IN := TRUE, PT := t#5.0s );
18:     IF (%input = no_tr AND NOT tmr.Q) THEN
19:        state := 0;
20:        %output := N;
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?

```plaintext
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 THEN
17:   tmr( IN := TRUE, PT := t#5.0s );
18:   IF (%input = no_tr AND NOT tmr.Q) THEN
19:      state := 0;
20:      %output := N;
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
```
Tied together by

- Sequential Function Charts (SFC)

Unfortunate: deviations in semantics... [Bauer, 2003]
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

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

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

\[ A = (Q = \{q_0, q_1\}, \]

\[ \Sigma = \{\text{tr, no\text{-}tr}\}, \]

\[ \delta = \{(q_0, \text{tr}) \mapsto q_1, (q_0, \text{no\text{-}tr}) \mapsto q_0, (q_1, \text{tr}) \mapsto q_1, (q_1, \text{no\text{-}tr}) \mapsto q_0\}, \]

\[ q_0 = q_0, \]

\[ \varepsilon = 0.2, \]

\[ S_t = \{q_0 \mapsto 0, q_1 \mapsto 5\}, \]

\[ S_e = \{q_0 \mapsto \emptyset, q_1 \mapsto \Sigma\}, \]

\[ \Omega = \{N, T\}, \]

\[ \omega = \{q_0 \mapsto N, q_1 \mapsto T\} \]
PLC Automata Example: Stuttering Filter with Exception

---

Diagram of a PLC automaton with states labeled as `N`, `T`, and `X`, with transitions labeled as `no_tr` and `tr`, including timing delays of 0.2 seconds and 5 seconds. The automaton transitions between states based on the truth value `true` or `false`.
PLC Automaton Semantics

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 THEN
17:     tmr( IN := TRUE, PT := t#5.0s );
18:     IF (%input = no_tr AND NOT tmr.Q) THEN
19:       state := 0;
20:       %output := N;
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
28: ENDIF

Recall:
- read inputs
- compute
- write outputs
PLCA Semantics: Examples

```
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 THEN
17:   tmr( IN := TRUE, PT := t#5.0s );
18:   IF (%input = no_tr AND NOT tmr.Q) THEN
19:      state := 0;
20:      %output := N;
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
```
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>
<tbody>
<tr>
<td>‘Req’</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>‘Des’</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>‘Impl’</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

`\[ A \]_{DC}`

synthesis

today

(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

  \[ \mathcal{I} \in \llbracket \mathcal{A} \rrbracket \implies \mathcal{I} \models \llbracket \mathcal{A} \rrbracket_{DC} \]

  but not necessarily the other way round.

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

- Consider

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

- The DC formula \([A]_{DC}\) we construct ranges over the observables

  - \(\text{In}_A, \mathcal{D}(\text{In}_A) = \Sigma\) — values of the inputs
  
  - \(\text{St}_A, \mathcal{D}(\text{St}_A) = Q\) — current local state

  - \(\text{Out}_A, \mathcal{D}(\text{Out}_A) = \Omega\) — values of the outputs
Overview

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

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

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

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

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

A = (Q, Σ, δ, q₀, ε, Sₜ, Sₑ, Ω, ω)

- A arbitrary with ∅ ≠ A ⊆ Σ,
- [q ∧ A] abbreviates [Stₐ = q ∧ lnₐ ∈ A],
- δ(q, A) abbreviates Stₐ ∈ {δ(q, a) | a ∈ A}.

Progress from non-delayed inputs:

\[ Sₜ(q) = 0 \land q \notin δ(q, A) \implies □([q ∧ A] \implies ℓ < 2ε) \]  \hspace{1cm} (DC-6)

\[ Sₜ(q) = 0 \land q \notin δ(q, A) \implies [¬q] ; [q ∧ A] \hspace{0.5cm} ε \implies [¬q] \]  \hspace{1cm} (DC-7)

Progress from delayed inputs:

\[ Sₜ(q) > 0 \land q \notin δ(q, A) \]
\[ \implies □([q]^{Sₜ(q)} ; [q ∧ A] \implies ℓ < Sₜ(q) + 2ε) \]  \hspace{1cm} (DC-8)

\[ Sₜ(q) > 0 \land A ∩ Sₑ(q) = ∅ \land q \notin δ(q, A) \]
\[ \implies □([q ∧ A] \implies ℓ < 2ε) \]  \hspace{1cm} (DC-9)

\[ Sₜ(q) > 0 \land A ∩ Sₑ(q) = ∅ \land q \notin δ(q, A) \]
\[ \implies [¬q] ; [q ∧ A] \hspace{0.5cm} ε \implies [¬q] \]  \hspace{1cm} (DC-10)
 Behaviour of the Output and System Start

\[ \Box([q] \implies [\omega(q)]) \]  

(\text{DC-11})

\[ [q_0 \land A] \rightarrow_0 [q_0 \lor \delta(q_0, A)] \]  

(\text{DC-2'})

\[ S_t(q_0) > 0 \implies [q_0 \land A] \xrightarrow{\leq S_t(q_0)}_0 [q_0 \lor \delta(q_0, A \setminus S_e(q_0))] \]  

(\text{DC-4'})

\[ S_t(q_0) > 0 \implies [q_0] ; [q_0 \land A]^\varepsilon \xrightarrow{\leq S_t(q_0)}_0 [q_0 \lor \delta(q_0, A \setminus S_e(q_0))] \]  

(\text{DC-5'})

\[ S_t(q_0) = 0 \land q_0 \notin \delta(q_0, A) \implies [q_0 \land A]^\varepsilon \rightarrow_0 [\neg q_0] \]  

(\text{DC-7'})

\[ S_t(q_0) > 0 \land A \cap S_e(q_0) = \emptyset \land q_0 \notin \delta(q_0, A) \implies [q_0 \land A]^\varepsilon \rightarrow_0 [\neg q_0] \]  

(\text{DC-10'})
Definition 5.3.
The **Duration Calculus semantics** of a PLC Automaton $\mathcal{A}$ is

$$\llbracket \mathcal{A} \rrbracket_{DC} := \bigwedge_{q \in Q, \emptyset \neq A \subseteq \Sigma} DC-1 \land \cdots \land DC-11 \land DC-2' \land DC-4' \land DC-5' \land DC-7' \land DC-10'.$$

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 $\mathcal{I}_\pi$ be an encoding of $\pi$ as an interpretation of $\text{In}_\mathcal{A}$, $\text{St}_\mathcal{A}$, and $\text{Out}_\mathcal{A}$.
- Then $\mathcal{I}_\pi \models \llbracket \mathcal{A} \rrbracket_{DC}$.
- 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

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

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

- Which is (why?) far 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 **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
  
  $\left[ \text{St}_A \in \Pi \land \text{In}_A \in A \right] \xrightarrow{c} \left[ \text{St}_A \in \Pi_{\text{target}} \right],$

  abbreviated as
  
  $\left[ \Pi \land A \right] \xrightarrow{c} \left[ \Pi_{\text{target}} \right].$
Reaction Time Theorem Premises

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

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

for PLC Automata with

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

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

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

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

where

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

and

$$s(\pi, A) := \begin{cases} S_t(\pi) + 2\varepsilon & \text{, if } S_t(\pi) > 0 \text{ and } A \cap S_e(\pi) \neq \emptyset \\ \varepsilon & \text{, otherwise.} \end{cases}$$
**Theorem 5.8.** Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, \varepsilon, S_t, S_e, \Omega, \omega)$, $\Pi \subseteq Q$, and $A \subseteq \Sigma$ with

$$\delta(\Pi, A) \subseteq \Pi.$$ 

Then for all $n \in \mathbb{N}_0$,

$$[\Pi \wedge A] \xrightarrow{c_n} \underbrace{[\delta^n(\Pi, A)]}_{=\Pi_{\text{target}}}$$

where

$$c_n := \varepsilon + \max\left(\left\{0\right\} \cup \sum_{i=1}^{k} s(\pi_i, A) \right)$$

and $s(\pi, A)$ as before.
Methodology: Overview
What does “◦” help us?
E.g.: What assumptions did we use?

(by example)

ST

(correct?) compiler
References
References

