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

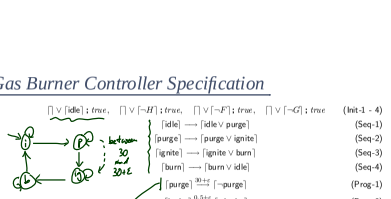
Example: Gas Burner

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 $\mathcal{D}(C) = \{\text{idle}, \text{purge}, \text{ignite}, \text{burn}\}$, representing the **controller**, (local)
- G : Boolean, representing **gas valve**. (output)

Gas Burner Controller Specification



$$\lceil \text{idle} \rceil ; true, \quad \lceil \vee \lceil \neg H \rceil ; true, \quad \lceil \vee \lceil \neg F \rceil ; true, \quad \lceil \vee \lceil \neg G \rceil ; true \qquad \text{(Init-1 - 4)}$$

$$\lceil \text{idle} \rceil \longrightarrow \lceil \text{idle} \vee \text{purge} \rceil \qquad \text{(Seq-1)}$$

$$\lceil \text{purge} \rceil \longrightarrow \lceil \text{purge} \vee \text{ignite} \rceil \qquad \text{(Seq-2)}$$

$$\lceil \text{ignite} \rceil \longrightarrow \lceil \text{ignite} \vee \text{burn} \rceil \qquad \text{(Seq-3)}$$

$$\lceil \text{burn} \rceil \longrightarrow \lceil \text{burn} \vee \text{idle} \rceil \qquad \text{(Seq-4)}$$

$$\lceil \text{purge} \rceil \xrightarrow{30 \cdot \ell \cdot \ell} \lceil \neg \text{purge} \rceil \qquad \text{(Prog-1)}$$

$$\lceil \text{ignite} \rceil \xrightarrow{0.5 \cdot \ell \cdot \ell} \lceil \neg \text{ignite} \rceil \qquad \text{(Prog-2)}$$

$$\lceil \text{idle} \wedge H \rceil \xrightarrow{\varepsilon} \lceil \neg \text{idle} \rceil \qquad \text{(Syn-1)}$$

$$\lceil \text{burn} \wedge (\neg H \vee \neg F) \rceil \xrightarrow{\varepsilon} \lceil \neg \text{burn} \rceil \qquad \text{(Syn-2)}$$

$$\lceil G \wedge (\text{idle} \vee \text{purge}) \rceil \xrightarrow{\varepsilon} \lceil \neg G \rceil \qquad \text{(Syn-3)}$$

$$\lceil \neg G \wedge (\text{ignite} \vee \text{burn}) \rceil \xrightarrow{\varepsilon} \lceil G \rceil \qquad \text{(Syn-4)}$$

$$\lceil \neg \text{idle} \rceil ; \lceil \text{idle} \wedge \neg H \rceil \longrightarrow \lceil \text{idle} \rceil \qquad \text{(Stab-1)}$$

$$\lceil \text{idle} \wedge \neg H \rceil \longrightarrow_0 \lceil \text{idle} \rceil \qquad \text{(Stab-1-init)}$$

$$\lceil \neg \text{purge} \rceil ; \lceil \text{purge} \rceil \xrightarrow{\leq 30} \lceil \text{purge} \rceil \qquad \text{(Stab-2)}$$

$$\lceil \neg \text{ignite} \rceil ; \lceil \text{ignite} \rceil \xrightarrow{\leq 0.5} \lceil \text{ignite} \rceil \qquad \text{(Stab-3)}$$

$$\lceil \neg \text{burn} \rceil ; \lceil \text{burn} \wedge H \wedge F \rceil \longrightarrow \lceil \text{burn} \rceil \qquad \text{(Stab-4)}$$

$$\lceil F \rceil ; \lceil \neg F \wedge \neg \text{ignite} \rceil \longrightarrow \lceil \neg F \rceil \qquad \text{(Stab-5)}$$

$$\lceil \neg F \wedge \neg \text{ignite} \rceil \longrightarrow_0 \lceil \neg F \rceil \qquad \text{(Stab-5-init)}$$

$$\lceil G \rceil ; \lceil \neg G \wedge (\text{idle} \vee \text{purge}) \rceil \longrightarrow \lceil \neg G \rceil \qquad \text{(Stab-6)}$$

$$\lceil \neg G \wedge (\text{idle} \vee \text{purge}) \rceil \longrightarrow_0 \lceil \neg G \rceil \qquad \text{(Stab-6-init)}$$

$$\lceil \neg G \rceil ; \lceil G \wedge (\text{ignite} \vee \text{burn}) \rceil \longrightarrow \lceil G \rceil \qquad \text{(Stab-7)}$$

Gas Burner Controller Correctness Proof

$$\text{GB-Ctrl} := \text{Init-1} \wedge \cdots \wedge \text{Stab-7} \wedge \varepsilon > 0$$

Recall:

$$\text{Req} :\Longleftrightarrow \Box(\ell \geq 60 \implies 20 \cdot \textstyle\int L \leq \ell)$$

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

$$\models \text{Req-1} \implies \text{Req}$$

for the **simplified**

$$\text{Req-1} := \Box(\ell \leq 30 \implies \textstyle\int L \leq 1).$$

Here we show

$$\models \text{GB-Ctrl} \wedge A(\varepsilon) \implies \text{Req-1}.$$

Lemma 3.15

$$\models \text{GB-Ctrl} \implies \Box \begin{pmatrix} (\lceil \text{idle} \rceil \implies & \int G \le \varepsilon) \\ \wedge \ (\lceil \text{purge} \rceil \implies & \int G \le \varepsilon) \\ \wedge \ (\lceil \text{ignite} \rceil \implies & \ell \le 0.5 + \varepsilon) \\ \wedge \ (\lceil \text{burn} \rceil \implies & \int \neg F \le 2\varepsilon) \end{pmatrix}$$

- $\mathcal{I}, [b,e] \models \lceil \text{burn} \rceil$

  (Stab-2) $\quad \lceil \text{burn} \wedge (\neg H \wedge \neg F) \rceil \xrightarrow{\varepsilon} \lceil \neg \text{burn} \rceil$

  (Stab-5) $\quad \lceil F \rceil ; \lceil \neg F \wedge \neg \text{ignite} \rceil \longrightarrow \lceil \neg F \rceil$

  $\mathcal{I}, [b,e] \models \Box \left(\lceil \neg F \rceil \implies \ell \le \varepsilon \right) \quad \lceil F \rceil ; \lceil F \rceil$

  $\wedge \neg \Diamond \left(\lceil F \rceil ; \lceil \neg F \rceil ; \lceil F \rceil \right) \quad \lceil F \rceil ; \lceil \neg F \rceil ; \lceil F \rceil ; \lceil F \rceil$

Lemma 3.16

$$\models \exists \varepsilon \bullet \text{GB-Ctrl} \implies \underbrace{\Box (\ell \le 30 \implies \int L \le 1)}_{\text{Req-1}}$$

Proof Sketch:

Consider $\mathcal{I}, \mathcal{V}, [b,e]$ s.t. $\mathcal{I}, \mathcal{V}, [b,e] \models \text{GB-Ctrl} \wedge \ell \le 30$.

Distinguish 5 cases:

$\mathcal{I}, \mathcal{V}, [b,e] \models \lceil \rceil$

$\vee \ (\lceil \text{idle} \rceil ; \text{true} \wedge \ell \le 30)$ (1)

$\vee \ (\lceil \text{purge} \rceil ; \text{true} \wedge \ell \le 30)$ (4)

$\vee \ (\lceil \text{ignite} \rceil ; \text{true} \wedge \ell \le 30)$ (3)

$\vee \ (\lceil \text{burn} \rceil ; \text{true} \wedge \ell \le 30)$ (2)

Lemma 3.16 Cont'd

- Case 0: $\mathcal{I}, \mathcal{V}, [b,e] \models \lceil \rceil$ ✓
- Case 1: $\mathcal{I}, \mathcal{V}, [b,e] \models \lceil \text{idle} \rceil ; \text{true} \wedge \ell \le 30$

$\lceil \text{idle} \rceil \longrightarrow \lceil \text{idle} \vee \text{purge} \rceil$ (Seq-1)

$\lceil \neg \text{purge} \rceil ; \lceil \text{purge} \rceil \xrightarrow{\le 30} \lceil \text{purge} \rceil$ (Stab-2)

↳ $\mathcal{I}, \mathcal{V}, [b,e] \models \lceil \text{idle} \rceil ; \lceil \text{idle} \rceil ; \lceil \text{purge} \rceil$

3.15 ↳ $\mathcal{I}, \mathcal{V}, [b,e] \models \int L \le \varepsilon \vee \int L \le \varepsilon ; \int L \le \varepsilon$

↳ $\mathcal{I}, \mathcal{V}, [b,e] \models \int L \le 2 \cdot \varepsilon$

Thus $\boxed{\varepsilon \le 0.5}$ is sufficient for Req-1 in this case.

Lemma 3.16 Cont'd

- Case 2: $\mathcal{I}, \mathcal{V}, [b,e] \models \lceil \text{burn} \rceil ; \text{true} \wedge \ell \le 30$

$\lceil \text{burn} \rceil \longrightarrow \lceil \text{burn} \vee \text{idle} \rceil$ (Seq-4)

↳ $\mathcal{I}, \mathcal{V}, [b,e] \models (\lceil \text{burn} \rceil \vee \lceil \text{burn} \rceil ; \lceil \text{idle} \rceil ; \text{true}) \wedge \ell \le 30$

3.15, Case 1 ↳ $\mathcal{I}, \mathcal{V}, [b,e] \models (\int L \le 2\varepsilon \vee \int L \le 2\varepsilon ; \int L \le 2\varepsilon) \wedge \ell \le 30$

↳ $\mathcal{I}, \mathcal{V}, [b,e] \models \int L \le 4 \cdot \varepsilon$

Thus $\boxed{\varepsilon \le 0.25}$ sufficient for Req-1.

Lemma 3.16 Cont'd

- Case 3: $\mathcal{I}, \mathcal{V}, [b,e] \models \lceil \text{ignite} \rceil ; \text{true} \wedge \ell \le 30$

$\lceil \text{ignite} \rceil \longrightarrow \lceil \text{ignite} \vee \text{burn} \rceil$ (Seq-3)

↳ $\mathcal{I}, \mathcal{V}, [b,e] \models \lceil \text{ignite} \rceil ; \lceil \text{ignite} \rceil ; \lceil \text{burn} \rceil ; \text{true}$

3.15, Case 2 ↳ $\mathcal{I}, \mathcal{V}, [b,e] \models \int L \le 0.5 + \varepsilon \vee (\int L \le 0.5 + \varepsilon ; \int L \le 0.5 + \varepsilon)$

↳ $\mathcal{I}, \mathcal{V}, [b,e] \models \int L \le 0.5 + 5\varepsilon$

So $\boxed{\varepsilon \le 0.1}$ sufficient for Req-1.

Lemma 3.16 Cont'd

- Case 4: $\mathcal{I}, \mathcal{V}, [b,e] \models \lceil \text{purge} \rceil ; \text{true} \wedge \ell \le 30$

$\lceil \text{purge} \rceil \longrightarrow \lceil \text{purge} \vee \text{ignite} \rceil$ (Seq-2)

↳ $\mathcal{I}, \mathcal{V}, [b,e] \models \lceil \text{purge} \rceil ; \lceil \text{purge} \rceil ; \lceil \text{ignite} \rceil ; \text{true}$

3.15, (var) ↳ $\mathcal{I}, \mathcal{V}, [b,e] \models \int L \le \varepsilon \vee (\int L \le \varepsilon ; \int L \le 0.5 + 5\varepsilon)$

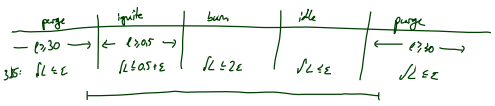
↳ $\mathcal{I}, \mathcal{V}, [b,e] \models \int L \le 0.5 + 6\varepsilon$

Thus $\boxed{\varepsilon \le \frac{1}{12}}$ is sufficient for Req-1 in this case.

Correctness Result

Theorem 3.17.

$$\models \left(\text{GB-Ctrl} \wedge \varepsilon \leq \frac{1}{12}\right) \implies \text{Req}$$



↳ $\leq 0.5+6\varepsilon$ in the worst case

↳ $A(\varepsilon) :\Leftrightarrow \varepsilon \leq \frac{1}{12}$

Discussion

- We used only

  'Seq-1', 'Seq-2', 'Seq-3', 'Seq-4',
  'Prog-2', 'Syn-2', 'Syn-3',
  'Stab-2', 'Stab-5', 'Stab-6'.

What about

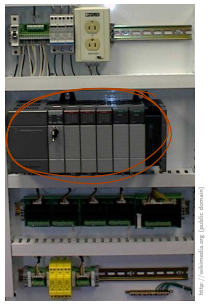
$$\text{Prog-1} = \lceil \text{purge} \rceil \xrightarrow{30+\varepsilon} \lceil \neg\text{purge} \rceil$$
$$\lceil \text{idle} \wedge H \rceil \xrightarrow{\varepsilon} \lceil \text{idle} \rceil$$

for instance?

Nope, there is the requirement (not explicitly noted down) that the system does something finally, e.g. get the heating going on segment.

What is a PLC?

How do PLC look like?



https://commons.wikimedia.org (public domain)

https://commons.wikimedia.org (CC-by-sa 2.0, 1001100)

What's special about PLC?

PLC - programmable logic controller
SPS - Speicherprogrammierbare Steuerung

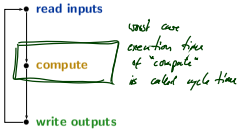
- microprocessor, memory, **timers**
- digital (or analog) I/O ports
- possibly RS 232, fieldbuses, networking
- robust hardware
- reprogrammable
- **standardised programming model** (IEC 61131-3)

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:

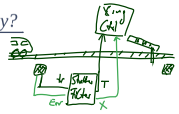


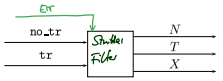
- Cyclic operation is repeated until external interruption (such as shutdown or reset).
- Cycle time: typically a few milliseconds. [Lukoschus, 2004]

- Programming for PLC means providing the "**compute**" part.
- Input/output values are available via designated local variables.

How are PLC programmed, practically?

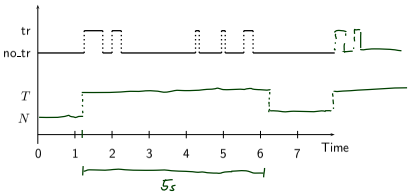
- **Example:** reliable, stutter-free train sensor.
  - Assume a track-side sensor with outputs:
    - `no_tr` — "no passing train"
    - `tr` — "a train is passing"
  - Assume that a change from "`no_tr`" to "`tr`" signals arrival of a train. (No spurious sensor values.)
  - **Problem:** the sensor 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.



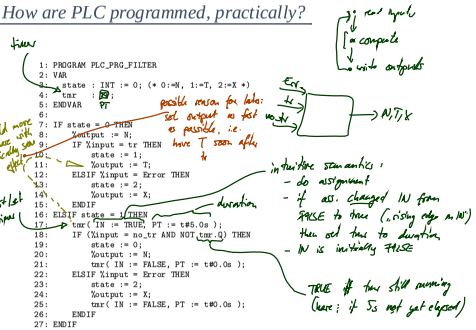


Example: Stutter Filter

- **Idea:** After arrival of a train, ignore "`no_tr`" for 5 seconds.



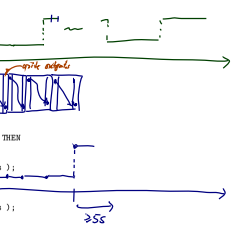
How are PLC programmed, practically?



```
1: PROGRAM PLC_PRG_FILTER
2: VAR
3:   state : INT := 0; (* 0:=N, 1:=T, 2:=X *)
4:   tmr   : 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
```


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: 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
```



Alternative Programming Languages by IEC 61131-3

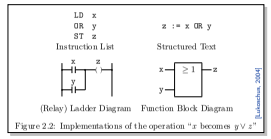


Figure 2.2: Implementations of the operation "x becomes $y \vee z$"

Tied together by

- Sequential Function Charts (SFC)

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

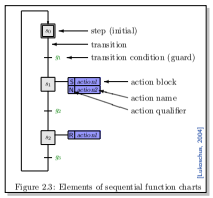


Figure 2.3: Elements of sequential function charts

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 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

PLC Automata

Definition 5.2. A **PLC-Automaton** is a structure

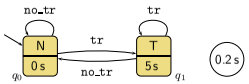
$$\mathcal{A} = (Q, \Sigma, \delta, q_0, \varepsilon, S_t, S_e, \Omega, \omega)$$

where

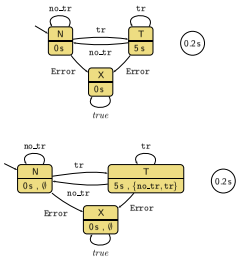
- $(q \in)$ Q is a finite set of **states**, $q_0 \in Q$ is the **initial state**,
- $(\sigma \in)$ Σ is a finite set of **inputs**,
- $\delta : Q \times \Sigma \to Q$ is the **transition function** (!),
- $S_t : Q \to \mathbb{R}_0^+$ assigns a **delay time** to each state,
- $S_e : Q \to 2^\Sigma$ assigns a set of **delayed inputs** to each state,
- Ω is a finite, non-empty set of **outputs**,
- $\omega : Q \to \Omega$ assigns an **output** to each state,
- ε is an **upper time bound** for the execution cycle.

PLC Automata Example: Stuttering Filter

$$\begin{aligned}\mathcal{A} = (&Q = \{q_0, q_1\},\\ &\Sigma = \{\texttt{tr}, \texttt{no\_tr}\},\\ &\delta = \{(q_0, \texttt{tr}) \mapsto q_1, (q_0, \texttt{no\_tr}) \mapsto q_0, (q_1, \texttt{tr}) \mapsto q_1, (q_1, \texttt{no\_tr}) \mapsto q_0\},\\ &q_0 = q_0,\\ &\varepsilon = 0.2,\\ &S_t = \{q_0 \mapsto 0, q_1 \mapsto 5\},\\ &S_e = \{q_0 \mapsto \emptyset, q_1 \mapsto \Sigma\},\\ &\Omega = \{N, T\},\\ &\omega = \{q_0 \mapsto N, q_1 \mapsto T\})\end{aligned}$$

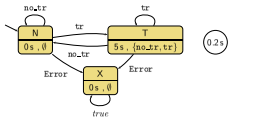


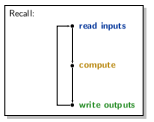
PLC Automata Example: Stuttering Filter with Exception



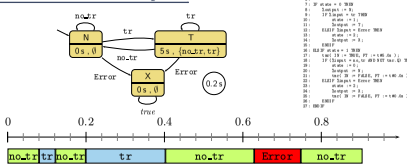
PLC Automaton Semantics

```
 1: PROGRAM PLC_PRG_FILTER
 2: VAR
 3:   state : INT := 0; (* 0:=N, 1:=T, 2:=X *)
 4:   tmr   : TP;
 5: END_VAR
 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:     END_IF
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:     END_IF
27: END_IF
```





PLCA Semantics: Examples



We assess correctness in terms of cycle time...

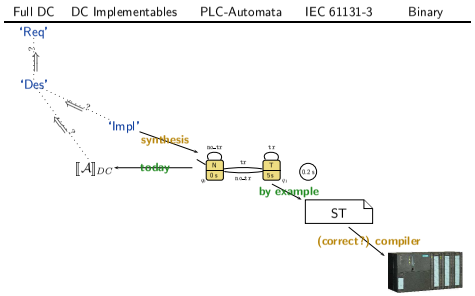
...but where does the cycle time come from?

- First of all, ST on the hardware **has** a cycle time
  - so we can **measure** it — if it is larger than ε, don't use this program on this controller
  - we can **estimate** (approximate) the **WCET** (worst case execution time) — if it's larger than ε, 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 ε,
  - 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?



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 $[\![\mathcal{A}]\!]_{DC}$ such that

$$\text{“}\mathcal{I} \in [\![\mathcal{A}]\!]\text{”} \implies \mathcal{I} \models [\![\mathcal{A}]\!]_{DC}$$

  but not necessarily the other way round.
- **In even other words:** $\text{“}[\![\mathcal{A}]\!]\text{”} \subseteq \{\mathcal{I} \mid \mathcal{I} \models [\![\mathcal{A}]\!]_{DC}\}$.

Observables

- Consider
$$\mathcal{A} = (Q, \Sigma, \delta, q_0, \varepsilon, S_t, S_e, \Omega, \omega).$$
- The DC formula $[\![\mathcal{A}]\!]_{DC}$ we construct ranges over the observables
  - $\mathsf{In}_{\mathcal{A}}$, $\mathcal{D}(\mathsf{In}_{\mathcal{A}}) = \Sigma$ — values of the **inputs**
  - $\mathsf{St}_{\mathcal{A}}$, $\mathcal{D}(\mathsf{St}_{\mathcal{A}}) = Q$ — current **local state**
  - $\mathsf{Out}_{\mathcal{A}}$, $\mathcal{D}(\mathsf{Out}_{\mathcal{A}}) = \Omega$ — values of the **outputs**

Overview

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

- A arbitrary with $\emptyset \neq A \subseteq \Sigma$,
- $\lceil q \wedge A \rceil$ abbreviates $\lceil \mathsf{St}_{\mathcal{A}} = q \wedge \mathsf{In}_{\mathcal{A}} \in A \rceil$,
- $\delta(q, A)$ abbreviates $\mathsf{St}_{\mathcal{A}} \in \{\delta(q, a) \mid a \in A\}$.

- **Initial State**:
$$\lceil \rceil \vee \lceil q_0 \rceil \,;\, true \qquad \text{(DC-1)}$$
- **Effect of Transitions, untimed**:
$$\lceil \neg q \rceil \,;\, \lceil q \wedge A \rceil \longrightarrow \lceil q \vee \delta(q, A) \rceil \qquad \text{(DC-2)}$$
- **Cycle time**:
$$\lceil q \wedge A \rceil \xrightarrow{\varepsilon} \lceil q \vee \delta(q, A) \rceil \qquad \text{(DC-3)}$$
- **Delays**:
$$S_t(q) > 0 \implies \lceil \neg q \rceil \,;\, \lceil q \wedge A \rceil \xrightarrow{\leq S_t(q)} \lceil q \vee \delta(q, A \setminus S_e(q)) \rceil \qquad \text{(DC-4)}$$
$$S_t(q) > 0 \implies \lceil \neg q \rceil \,;\, \lceil q \rceil \,;\, \lceil q \wedge A \rceil^{\varepsilon} \xrightarrow{\leq S_t(q)} \lceil q \vee \delta(q, A \setminus S_e(q)) \rceil \qquad \text{(DC-5)}$$

Overview

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

- A arbitrary with $\emptyset \neq A \subseteq \Sigma$,
- $\lceil q \wedge A \rceil$ abbreviates $\lceil \mathsf{St}_{\mathcal{A}} = q \wedge \mathsf{In}_{\mathcal{A}} \in A \rceil$,
- $\delta(q, A)$ abbreviates $\mathsf{St}_{\mathcal{A}} \in \{\delta(q, a) \mid a \in A\}$.

- **Progress from non-delayed inputs**:
$$S_t(q) = 0 \wedge q \notin \delta(q, A) \implies \Box(\lceil q \wedge A \rceil \implies \ell < 2\varepsilon) \qquad \text{(DC-6)}$$
$$S_t(q) = 0 \wedge q \notin \delta(q, A) \implies \lceil \neg q \rceil \,;\, \lceil q \wedge A \rceil^{\varepsilon} \longrightarrow \lceil \neg q \rceil \qquad \text{(DC-7)}$$

- **Progress from delayed inputs**:
$$S_t(q) > 0 \wedge q \notin \delta(q, A) \implies \Box(\lceil q \rceil^{S_t(q)} \,;\, \lceil q \wedge A \rceil \implies \ell < S_t(q) + 2\varepsilon) \qquad \text{(DC-8)}$$
$$S_t(q) > 0 \wedge A \cap S_e(q) = \emptyset \wedge q \notin \delta(q, A) \implies \Box(\lceil q \wedge A \rceil \implies \ell < 2\varepsilon) \qquad \text{(DC-9)}$$
$$S_t(q) > 0 \wedge A \cap S_e(q) = \emptyset \wedge q \notin \delta(q, A) \implies \lceil \neg q \rceil \,;\, \lceil q \wedge A \rceil^{\varepsilon} \longrightarrow \lceil \neg q \rceil \qquad \text{(DC-10)}$$

Behaviour of the Output and System Start

$$\Box(\lceil q \rceil \implies \lceil \omega(q) \rceil) \qquad \text{(DC-11)}$$

$$\lceil q_0 \wedge A \rceil \longrightarrow_0 \lceil q_0 \vee \delta(q_0, A) \rceil \qquad \text{(DC-2')}$$

$$S_t(q_0) > 0 \implies \lceil q_0 \wedge A \rceil \xrightarrow{\leq S_t(q_0)}_0 \lceil q_0 \vee \delta(q_0, A \setminus S_e(q_0)) \rceil \qquad \text{(DC-4')}$$

$$S_t(q_0) > 0 \implies \lceil q_0 \rceil \,;\, \lceil q_0 \wedge A \rceil^{\varepsilon} \xrightarrow{\leq S_t(q_0)}_0 \lceil q_0 \vee \delta(q_0, A \setminus S_e(q_0)) \rceil \qquad \text{(DC-5')}$$

$$S_t(q_0) = 0 \wedge q_0 \notin \delta(q_0, A) \implies \lceil q_0 \wedge A \rceil^{\varepsilon} \longrightarrow_0 \lceil \neg q_0 \rceil \qquad \text{(DC-7')}$$

$$S_t(q_0) > 0 \wedge A \cap S_e(q_0) = \emptyset \wedge q_0 \notin \delta(q_0, A) \implies \lceil q_0 \wedge A \rceil^{\varepsilon} \longrightarrow_0 \lceil \neg q_0 \rceil \qquad \text{(DC-10')}$$

DC Semantics of PLC Automata

Definition 5.3.
The **Duration Calculus semantics** of a PLC Automaton $\mathcal{A}$ is

$$[\![\mathcal{A}]\!]_{DC} := \bigwedge_{\substack{q \in Q, \\ \emptyset \neq A \subseteq \Sigma}} \text{DC-1} \wedge \cdots \wedge \text{DC-11} \wedge \text{DC-2}' \wedge \text{DC-4}' \wedge \text{DC-5}' \wedge \text{DC-7}' \wedge \text{DC-10}'.$$

Claim:

- Let $P_{\mathcal{A}}$ be the ST program semantics of $\mathcal{A}$.
- Let π 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 π as an interpretation of $\mathsf{In}_{\mathcal{A}}$, $\mathsf{St}_{\mathcal{A}}$, and $\mathsf{Out}_{\mathcal{A}}$.
- Then $\mathcal{I}_{\pi} \models [\![\mathcal{A}]\!]_{DC}$.
- 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

$$\lceil \mathsf{St}_{\mathcal{A}} \in Q \wedge \mathsf{In}_{\mathcal{A}} = emergency\_signal \rceil \xrightarrow{0.1} \lceil \mathsf{St}_{\mathcal{A}} = motor\_off \rceil$$

("**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**,
  - $c \in \mathsf{Time}$ be a **time bound**, and
  - $\Pi_{target} \subseteq Q$ be a set of **target states**.
- Then we seek to establish properties of the form

$$\lceil \mathsf{St}_{\mathcal{A}} \in \Pi \wedge \mathsf{In}_{\mathcal{A}} \in A \rceil \xrightarrow{c} \lceil \mathsf{St}_{\mathcal{A}} \in \Pi_{target} \rceil,$$

abbreviated as

$$\lceil \Pi \wedge A \rceil \xrightarrow{c} \lceil \Pi_{target} \rceil.$$

Reaction Time Theorem Premises

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

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

for PLC Automata with

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

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

$$\delta(\Pi, A) := \{\delta(q, a) \mid q \in \Pi \wedge a \in A\}.$$

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

Theorem 5.6. 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

$$\lceil \Pi \wedge A \rceil \xrightarrow{c} \lceil \underbrace{\delta(\Pi, A)}_{=\Pi_{\text{target}}} \rceil$$

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 (General Case)

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 $n \in \mathbb{N}_0$,

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

where

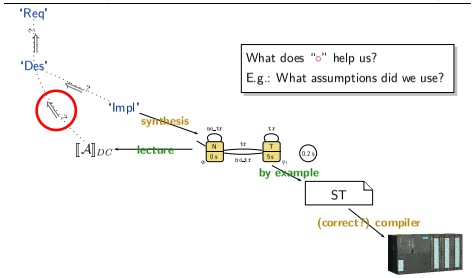
$$c_n := \varepsilon + \max\left(\{0\} \cup \left\{ \sum_{i=1}^{k} s(\pi_i, A) \;\middle|\; \begin{array}{l} 1 \leq k \leq n \wedge \\ \exists\, \pi_1, \ldots, \pi_k \in \Pi \setminus \delta^n(\Pi, A) \\ \quad \forall\, j \in \{1, \ldots, k-1\} : \\ \quad\quad \pi_{j+1} \in \delta(\pi_j, A) \end{array}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

and $s(\pi, A)$ as before.

Methodology: Overview

Methodology

Full DC | DC Implementables | PLC-Automata | IEC 61131-3 | Binary



References

References

[Bauer, 2003] Bauer, N. (2003). *Formale Analyse von Sequential Function Charts*. PhD thesis, Universitt Dortmund.

[Lukoschus, 2004] Lukoschus, B. (2004). *Compositional Verification of Industrial Control Systems*. PhD thesis, Christian-Albrechts-Universität zu Kiel.

[Olderog and Dierks, 2008] Olderog, E.-R. and Dierks, H. (2008). *Real-Time Systems - Formal Specification and Automatic Verification*. Cambridge University Press.