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

\[\begin{align*}
\text{Init-1} :&= \text{true}; \text{GB-Ctrl} = \text{true}; \\
\text{Init-2} :&= \text{true}; \text{GB-Ctrl} = \text{false}; \\
\text{Init-3} :&= \text{false}; \text{GB-Ctrl} = \text{true}; \\
\text{Init-4} :&= \text{false}; \text{GB-Ctrl} = \text{false}.
\end{align*}\]

Recall: Control Automata

Gas Burner Controller Specification

Capabilities for following tasks/questions.

Educational Objectives:

- This Lecture: Lecture 09: PLCAutomata
- Contents & Goals
- Real-Time Systems
- Example: Gas Burner

An overapproximating DC semantics for PLCA

Implementation with DC?

What is the "philosophy" of PLC?

What did we generalise/abstract them to?

How does the proposed approach work, from requirements to a correct?

What's an example for giving a DC semantics for a constructive formalism?
Lemma 3.15
\[
\begin{align*}
\mathbb{G}_\text{Ctrl} = & \implies \Box \\
\left( \lceil \text{idle} \rceil = \implies \int G \leq \varepsilon \right) \land \\
\left( \lceil \text{purge} \rceil = \implies \int G \leq \varepsilon \right) \land \\
\left( \lceil \text{ignite} \rceil = \implies \ell \leq 0.5 + \varepsilon \right) \land \\
\left( \lceil \text{burn} \rceil = \implies \int \neg F \leq 2 \varepsilon \right)
\end{align*}
\]

Lemma 3.16
\[
\begin{align*}
\exists \varepsilon \cdot \mathbb{G}_\text{Ctrl} = & \implies \Box \\
\left( \ell \leq 30 = \implies \int L \leq 1 \right)
\end{align*}
\]

Lemma 3.16 Cont'd
- Case 0: 
  \[ \text{I, V, } [b,e] = \lceil \text{idle} \rceil ; \text{true} \land \ell \leq 30 \implies \lceil \text{idle} \lor \text{purge} \rceil \] (Seq-1)
  \[ \lceil \neg \text{purge} \rceil ; \lceil \text{purge} \rceil \leq 30 \implies \lceil \text{purge} \rceil \] (Stab-2)

Lemma 3.16 Cont'd
- Case 1: 
  \[ \text{I, V, } [b,e] = \lceil \text{idle} \rceil ; \text{true} \land \ell \leq 30 \implies \lceil \text{idle} \lor \text{purge} \rceil \] (Seq-1)
  \[ \lceil \neg \text{purge} \rceil ; \lceil \text{purge} \rceil \leq 30 \implies \lceil \text{purge} \rceil \] (Stab-2)

Lemma 3.16 Cont'd
- Case 2: 
  \[ \text{I, V, } [b,e] = \lceil \text{burn} \rceil ; \text{true} \land \ell \leq 30 \implies \lceil \text{burn} \lor \text{idle} \rceil \] (Seq-4)

Lemma 3.16 Cont'd
- Case 3: 
  \[ \text{I, V, } [b,e] = \lceil \text{ignite} \rceil ; \text{true} \land \ell \leq 30 \implies \lceil \text{ignite} \lor \text{burn} \rceil \] (Seq-3)

Lemma 3.16 Cont'd
- Case 4: 
  \[ \text{I, V, } [b,e] = \lceil \text{purge} \rceil ; \text{true} \land \ell \leq 30 \implies \lceil \text{purge} \lor \text{ignite} \rceil \] (Seq-2)
Theorem 3.17: \[ \frac{1}{12} \leq \varepsilon \leq \frac{1}{12} \Rightarrow \text{Req} \]

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 'Prog-1' = \( \lceil \text{purge} \rceil + 30 \epsilon \rightarrow \lceil \neg \text{purge} \rceil \) for instance?

What is a PLC?

- Microprocessor, memory, timers
- Digital (or analog) I/O ports
- Possibly RS232, fieldbuses, networking
- Robust hardware
- Reprogrammable
- Standardized programming model (IEC 61131-3)

Where are PLC employed?

- Mostly process automation
- Production lines
- Packaging lines
- Chemical plants
- Power plants
- Electric motors, pneumatic or hydraulic cylinders

Not so much:

- Product automation, tailored or OTS controller boards
- Embedded controllers

What do PLC look like?

- [Image of PLCs with labels]
How are PLCs programmed?

- PLCs 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: How are PLCs 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.

Program: PLC_PRG_FILTER

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: 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
- Structured Text
- (Relay) Ladder Diagram
- Function Block Diagram

Figure 2.2: Implementations of the operation "x becomes y or z". [Lukoschus, 2004]

Tied together by:
- Sequential Function Charts (SFC)

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

Figure 2.3: Elements of sequential function charts. [Lukoschus, 2004]
WhystudyPLC?

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 time units, we can query whether the set time has elapsed, if we program it to wait for 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.

PLCAutomata

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 (!),
- $S_t : Q \rightarrow \mathbb{R}^+ \cup \{0\}$ 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.

PLCAutomata Example: Stuttering Filter

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

PLCAutomata Example: Stuttering Filter with Exception

$A = (Q = \{N, T, X\}, \Sigma = \{tr, no\ tr\}, \delta = \{ (N, tr) \mapsto T, (N, no\ tr) \mapsto X, (T, tr) \mapsto T, (T, no\ tr) \mapsto T, (X, tr) \mapsto X, (X, no\ tr) \mapsto X\}, tmr( IN \rightarrow FALSE, PT \rightarrow t#0.0s ))

PLCAutomaton Semantics

$\text{Recall:}$

PROGRAM PLC_PRG_FILTER
VAR
state : INT := 0; (* 0:=N, 1:=T, 2:=X *)
tmr : TP;
ENDVAR

IF state = 0 THEN
%output := N;
IF %input = tr THEN
state := 1;
%output := T;
ELSIF %input = Error THEN
state := 2;
%output := X;
ENDIF
ELSIF state = 1 THEN
%output := T;
ELSIF state = 2 THEN
%output := X;
tmr( IN := FALSE, PT := t#0.0s );
ENDIF
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 ε, don't use this program on this controller.
• We can estimate (approximate) the WCET (worst-case execution time)—if it is larger than ε, don't use it, if it is 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 all 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 \( I \in \mathbb{C}_3 \mathbb{A} \) such that

\( I \models \mathbb{C}_2 \mathbb{A} \mathbb{C}_3 \mathbb{D} \mathbb{C} \)

but not necessarily the other way round.

In even other words:

\( \mathbb{C}_2 \mathbb{A} \mathbb{C}_3 \mathbb{D} \subseteq \{ I | I \models \mathbb{C}_2 \mathbb{A} \mathbb{C}_3 \mathbb{D} \} \).
The Duration Calculus formula defined in Section 5.1 is applied to the state transition system of the PLC to define the Duration Calculus semantics of the PLC automaton. The Duration Calculus semantics are used to define the behavior of the PLC automaton, including the behavior of the output and the system start.

The Duration Calculus semantics of the PLC automaton are defined as follows:

- The initial state of the PLC automaton is defined as
  \[ \pi = (Q, A, \delta, \epsilon, S, \emptyset) \]

- The behavior of the output and the system start is defined as
  \[ \text{Behaviourof theOutputandSystemStart} \]

The Duration Calculus semantics are used to define the behavior of the PLC automaton, including the behavior of the output and the system start.
Given a PLC-Automaton, one often wants to know whether it guarantees properties of the form
\[
\begin{align*}
&\lceil St_A \in Q \land In_A = \text{emergency signal} \rceil \\
&\xrightarrow{0.1} \lceil St_A = \text{motor off} \rceil
\end{align*}
\] ("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
\[
\begin{align*}
&\lceil St_A \in \Pi \land In_A \in A \rceil \xrightarrow{c} \lceil St_A \in \Pi\text{-target} \rceil
\end{align*}
\]
abbreviated as
\[
\lceil \Pi \land A \rceil \xrightarrow{c} \lceil \Pi\text{-target} \rceil
\]

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
\[
\lceil \Pi \land A \rceil \xrightarrow{c} \lceil \delta_n(\Pi, A) \rceil = \Pi\text{-target}
\]
where
\[ c = \varepsilon + \max\left( \{0\} \cup \left\{ s(\pi, A) \mid \pi \in \Pi \setminus \delta_n(\Pi, A) \} \right\} \]
and
\[ s(\pi, A) = \begin{cases} 
S_t(\pi) + 2 \varepsilon, & \text{if } S_t(\pi) > 0 \text{ and } A \cap S_e(\pi) \neq \emptyset \\
\varepsilon, & \text{otherwise}
\end{cases} \]

The Reaction Time Theorem (General Case)

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 \( n \in \mathbb{N}_0 \),
\[
\lceil \Pi \land A \rceil \xrightarrow{c_n} \lceil \delta_n(\Pi, A) \rceil = \Pi\text{-target}
\]
where
\[ c_n = \varepsilon + \max\left( \{0\} \cup \left\{ \sum_{i=1}^{k} s(\pi_i, A) \mid 1 \leq k \leq n \land \exists \pi_1, \ldots, \pi_k \in \Pi \setminus \delta_n(\Pi, A) \forall j \in \{1, \ldots, k-1\} : \pi_j + 1 \in \delta(\pi_j, A) \} \right\} \]
Methodology

FullDCDCImplementablesPLC-AutomataIEC61131-3Binary

Req

Des

What does 

E.g.: What assumptions did we use?

Impl

/C2

0s

T

5s

0.2s

tr

no

no

tr

q

0

q

1

ST

=⇒

⇐

=⇒

⇐

=⇒

synthesis

byexample

(correct?)

compiler

References

