Introduction

- Observables and Evolutions
- Duration Calculus (DC)
- Semantical Correctness Proofs
- DC Decidability
- DC Implementables
- PLC-Automata
- Timed Automata (TA), Uppaal
- Networks of Timed Automata
- Region/Zone-Abstraction
- TA model-checking
- Extended Timed Automata
- Undecidability Results

Requirements vs. Implementations

- Problem: in general, a DC requirement doesn't tell how to achieve it, how to build a controller/write a program which ensures it.

\[
\begin{align*}
\lceil \neg B \rceil \land \ell = 5 & \Rightarrow \lceil L = \text{yellow} \rceil \land \text{true} \\
\lceil B \land L = \text{green} \rceil \land \ell = 5 & \Rightarrow \text{true} \land \lceil L = \text{red} \rceil
\end{align*}
\]

"whenever a pedestrian presses the button 5 time units from now, then now the traffic lights should already be yellow"

Plus: road traffic should not see 'yellow' all the time.

\[
\begin{align*}
\lceil B \land L = \text{green} \rceil \land \ell = 5 & \Rightarrow \text{true} \land \lceil L = \text{red} \rceil
\end{align*}
\]

"whenever a pedestrian presses the button while road traffic sees 'green', then 5 time units later (the latest) road traffic should see 'red'"

Motivation: Why DC Implementables?

- What can we assume of controller platforms?
- DC Standard Forms
- Followed-by, Followed-by-initially
- (Timed) Leads-to
- (Timed) Up-to, (Timed) Up-to-initially
- Control Automata
- phases, basic phases
- DC Implementables
- Initialisation, Sequencing, Progress
- Synchronisation, (Un)Bounded Stability
- (Un)Bounded Initial Stability

Example: A correct controller for the Gas Burner specified by DC Implementables.
Plan:

• Introduce DC Standard Forms (a sub-language of DC)
  - DC Standard Forms: Followed-by

• Introduce Control Automata

• Introduce DC Implementables as a subset of DC Standard Forms

Example: a correct controller design for the notorious Gas Burner

DC Standard Forms: Followed-by

In the following:

- $F$ is a DC formula,
- $P$ a state assertion,
- $\theta$ a rigid term.

Followed-by:

$F \rightarrow \lceil P \rceil$:

$\iff \neg \diamondsuit (F; \lceil \neg P \rceil)$

$\iff \Box \neg (F; \lceil \neg P \rceil)$

in other symbols

\[
\forall x \cdot \Box ((F \land \ell = x); \ell > 0 \Rightarrow (F \land \ell = x); \lceil P \rceil; \text{true})
\]
∀ x • □ (F ∧ ℓ = x) ; ℓ > 0 =⇒ (F ∧ ℓ = x) ; ⌈P⌉ ; true

b e (⌈Q⌉ ∧ ℓ = 1) − → ⌈P⌉

(Timed) leads-to:
F θ − → ⌈P⌉ : ⇐ ⇒ (F ∧ ℓ = θ) − → ⌈P⌉

b e ⌈Q⌉ ; ⌈Q⌉ ≤ 1 − → ⌈P⌉

(Timed) up-to:
F ≤ θ − → ⌈P⌉ :

b e ⌈¬Q⌉ ; ⌈Q⌉ ≤ 1 − → ⌈P⌉

Control Automata

Let $G$ be a control automaton with DC formula $\varphi$. Then $\varphi$ holds if and only if $G$ is a run $s_0, s_1, \ldots, s_n \in S^G$ such that $s_0$ is a start state and $s_n$ is a target state.

Example

Let $X = \{\text{red}, \text{yellow}, \text{green}\}$. Consider a system of traffic lights $T = \{\text{red}, \text{yellow}, \text{green}\}$ with initial state $s_0 = \text{red}$. The control automaton $G$ has the following properties:

- Initial state: $s_0 = \text{red}$
- Target state: $s_n = \text{green}$
- Transitions:
  - $s_0 \xrightarrow{\text{red} \to \text{yellow}} s_1$
  - $s_1 \xrightarrow{\text{yellow} \to \text{green}} s_2$

The formula $\varphi = X_0 \land F X_1 \land X_2$ correctly describes this system.

DC Standard Forms: Initialisation

Let $G$ be a control automaton with DC formula $\varphi$. Then $\varphi$ holds if and only if $G$ is a run $s_0, s_1, \ldots, s_n \in S^G$ such that $s_0$ is a start state and $s_n$ is a target state.

Example

Let $X = \{\text{red}, \text{yellow}, \text{green}\}$. Consider a system of traffic lights $T = \{\text{red}, \text{yellow}, \text{green}\}$ with initial state $s_0 = \text{red}$. The control automaton $G$ has the following properties:

- Initial state: $s_0 = \text{red}$
- Target state: $s_n = \text{green}$
- Transitions:
  - $s_0 \xrightarrow{\text{red} \to \text{yellow}} s_1$
  - $s_1 \xrightarrow{\text{yellow} \to \text{green}} s_2$

The formula $\varphi = X_0 \land F X_1 \land X_2$ correctly describes this system.
Example

\[ X = X \] of Phases

\[ \text{of Basic phases} \]

\[ \text{of Phases} \]

Abbreviations:

\[ \text{DC Implementables} \]

\[ \text{Phases} \]

\[ \text{Basic phases} \]

\[ \text{Control Automata} \]

\[ \text{Traffic lights} \]

\[ \text{Abbreviations:} \]

\[ \text{DC Implementables} \]

\[ \text{Phases} \]

\[ \text{Basic phases} \]

\[ \text{Control Automata} \]

\[ \text{Traffic lights} \]
Dear programmers,

I am writing to clarify the connection between the design document and the implementation. The design document describes the requirements (Req) and the implementation (Impl) that constrains phases of the controller. The goal is to ensure that the implementation meets the design specifications.

The connection is made through a local state that denotes the state of the controller, which can be interpreted as a transition to one of the phases, denoted as \( \Pi \). The time since the control automaton switched from one phase to another does not exceed a certain threshold, denoted as \( \Theta \), and \( \Theta \leq \Pi \).

In the initialisation phase, the local state is set to true, which triggers the transition to the next phase. The phases are connected by transitions, denoted as \( \Pi \), and the time since the transition does not exceed \( \Theta \).

In the bounded stability phase, the local state is set to true, which triggers the transition to the next phase. The time since the transition does not exceed \( \Theta \).

In the bounded initial stability phase, the local state is set to true, which triggers the transition to the next phase. The time since the transition does not exceed \( \Theta \).

In the synchronisation phase, the local state is set to true, which triggers the transition to the next phase. The time since the transition does not exceed \( \Theta \).

In the unbounded stability phase, the local state is set to true, which triggers the transition to the next phase. The time since the transition does not exceed \( \Theta \).

In the unbounded initial stability phase, the local state is set to true, which triggers the transition to the next phase. The time since the transition does not exceed \( \Theta \).

In the progress phase, the local state is set to true, which triggers the transition to the next phase. The time since the transition does not exceed \( \Theta \).

In the heating request phase, the local state is set to true, which triggers the transition to the next phase. The time since the transition does not exceed \( \Theta \).

In the gas burner controller phase, the local state is set to true, which triggers the transition to the next phase. The time since the transition does not exceed \( \Theta \).

The process is repeated for each phase, ensuring that the implementation meets the design specifications.
C: {idle, purge, ignite, burn}

⌈⌉ ∨ ⌈idle⌉;
true

(Init-1)

⌈idle⌉ −→ ⌈idle ∨ purge⌉

(Seq-1)

⌈purge⌉ −→ ⌈purge ∨ ignite⌉

(Seq-2)

⌈ignite⌉ −→ ⌈ignite ∨ burn⌉

(Seq-3)

⌈burn⌉ −→ ⌈burn ∨ idle⌉

(Seq-4)

e = idle

p = purge

i = ignite

b = burn
Gas Burner Controller: Environment Assumptions

$G:\{0,1\} \cup \{\neg G\}; true$

$(Init-4)$

$\neg G \leq \epsilon \lor purge \leq \epsilon \lor ignite \lor burn \lor idle \lor purge \lor ignite \lor burn$
Controller hardware platforms can
• read inputs,
• change local state,
• wait,
• write outputs.

If we limit controller behaviour descriptions
to these "operations", there's (at least) no
principle obstacle to implement the design.

One such limited specification language:
DC Implementables, a set of patterns of
DC Standard Forms.

DC Implementables basically contrain:
• local state changes, synchronisation with inputs
• and outputs, timed stability and progress.

This is sufficient to formalise a
correct (safe) Gas Burner controller design specification.

References
Cambridge University Press.