Contents & Goals

Last Lecture:
- Extended Timed Automata Cont’d
- A Fragment of TCTL
- Testable DC Formulae

This Lecture:
- Educational Objectives: Capabilities for following tasks/questions.
  - Are all DC formulae testable?
  - What’s a TBA and what’s the difference to (extended) TA?
  - What’s undecidable for timed (Büchi) automata? Idea of the proof?
- Content:
  - An untestable DC formula.
  - Timed Büchi Automata and timed regular languages [Alur and Dill, 1994].
  - The Universality Problem is undecidable for TBA [Alur and Dill, 1994]
  - Why this is unfortunate.
  - Timed regular languages are not everything.
Recall: Testability

Definition 6.1. A DC formula $F$ is called testable if an observer (or test automaton (or monitor)) $A_F$ exists such that for all networks $N = C(A_1, \ldots, A_n)$ it holds that

\[ N \models F \iff C(A_1', \ldots, A_n', A_F) \models \Box \neg (A_F.q_{bad}) \]

Otherwise it’s called untestable.

Proposition 6.3. There exist untestable DC formulae.

Theorem 6.4. DC implementables are testable.
Whenever we observe a change from $A$ to $\neg A$ at time $t_A$, the system has to produce a change from $B$ to $\neg B$ at some time $t_B \in [t_A, t_A + 1]$ and a change from $C$ to $\neg C$ at time $t_B + 1$.

**Sketch of Proof**: Assume there is $A_F$ such that, for all networks $\mathcal{N}$, we have

\[ \mathcal{N} \models F \iff C(A'_1, \ldots, A'_n, A_F) \models \forall \Box \neg(A_F . q_{bad}) \]

Assume the number of clocks in $A_F$ is $n \in \mathbb{N}_0$.

Consider the following time points:

- $t_A := 1$
- $t_i^B := t_A + \frac{2i - 1}{2(n+1)}$ for $i = 1, \ldots, n + 1$
- $t_i^C \in [t_i^B + 1 - \frac{1}{4(n+1)}, t_i^B + 1 + \frac{1}{4(n+1)}]$ for $i = 1, \ldots, n + 1$

with $t_i^C - t_i^B \neq 1$ for $1 \leq i \leq n + 1$.

**Example**: $n = 3$
Example: \( n = 3 \)

- The shown interpretation \( \mathcal{I} \) satisfies assumption of property.
- It has \( n + 1 \) candidates to satisfy commitment.
- By choice of \( t_C^i \), the commitment is not satisfied; so \( F \) not satisfied.
- Because \( A_F \) is a test automaton for \( F \), is has a computation path to \( q_{bad} \).
- Because \( n = 3 \), \( A_F \) can not save all \( n + 1 \) time points \( t_B^i \).
- Thus there is \( 1 \leq i_0 \leq n \) such that all clocks of \( A_F \) have a valuation which is not in \( 2 - t_B^{i_0} + (-\frac{1}{4(n+1)}, \frac{1}{4(n+1)}) \)

- Because \( A_F \) is a test automaton for \( F \), is has a computation path to \( q_{bad} \).
- Thus there is \( 1 \leq i_0 \leq n \) such that all clocks of \( A_F \) have a valuation which is not in \( 2 - t_B^{i_0} + (-\frac{1}{4(n+1)}, \frac{1}{4(n+1)}) \)

- Modify the computation to \( \mathcal{I}' \) such that \( t_C^i := t_B^{i_0} + 1 \).
- Then \( \mathcal{I}' \models F \), but \( A_F \) reaches \( q_{bad} \) via the same path.
- That is: \( A_F \) claims \( \mathcal{I}' \not\models F \).
- Thus \( A_F \) is not a test automaton. \textbf{Contradiction.}
\[ L = \{ a, b \} \]

\[ L(A) = \{ a^n b a^m \mid n, m \in \mathbb{N} \} \]

\[ (ab)^* \]
Timed Büchi Automata

[Alur and Dill, 1994]
... vs. Timed Automata

\[ \xi = \langle \text{off}, 0 \rangle, 0 \xrightarrow{\text{press} \ ?} \langle \text{light}, 0 \rangle, 1 \xrightarrow{\text{press} \ ?} \langle \text{light}, 3 \rangle, 4 \xrightarrow{\text{press} \ ?} \langle \text{bright}, 3 \rangle, 4 \xrightarrow{\ldots} \]

\( \xi \) is a computation path and run of \( A \).

New: Given a timed word

\[(a, 1), (b, 2), (a, 3), (b, 4), (a, 5), (b, 6), (b, t) \ldots \]

does \( A \) accept it?

New: acceptance criterion is visiting accepting state infinitely often.

Timed Languages

Definition. A time sequence \( \tau = \tau_1, \tau_2, \ldots \) is an infinite sequence of time values \( \tau_i \in \mathbb{R}^+ \), satisfying the following constraints:

(i) Monotonicity: \( \tau \) increases strictly monotonically, i.e. \( \tau_i < \tau_{i+1} \) for all \( i \geq 1 \).

(ii) Progress: For every \( t \in \mathbb{R}^+ \), there is some \( i \geq 1 \) such that \( \tau_i > t \).

Definition. A timed word over an alphabet \( \Sigma \) is a pair \((\sigma, \tau)\) where

- \( \sigma = \sigma_1, \sigma_2, \ldots \in \Sigma^\omega \) is an infinite word over \( \Sigma \), and
- \( \tau \) is a time sequence.

Definition. A timed language over an alphabet \( \Sigma \) is a set of timed words over \( \Sigma \).
Example: Timed Language

**Timed word** over alphabet $\Sigma$: a pair $(\sigma, \tau)$ where
- $\sigma = \sigma_1, \sigma_2, \ldots$ is an infinite word over $\Sigma$, and
- $\tau$ is a time sequence (strictly (!) monotonic, non-Zeno).

$$L_{ct} = \{(ab)^\omega, \tau) | \exists i \forall j \geq i : (\tau_{2j} < \tau_{2j-1} + 2)\}$$

**Timed Büchi Automata**

**Definition.** The set $\Phi(X)$ of clock constraints over $X$ is defined inductively by
\[\delta ::= x \leq c | c \leq x | \neg \delta | \delta_1 \land \delta_2\]
where $x \in X$ and $c \in \mathbb{Q}$ is a rational constant.

**Definition.** A **timed Büchi automaton** (TBA) $A$ is a tuple $(\Sigma, S, S_0, X, E, F)$, where
- $\Sigma$ is an alphabet,
- $S$ is a finite set of states, $S_0 \subseteq S$ is a set of start states,
- $X$ is a finite set of clocks, and
- $E \subseteq S \times S \times \Sigma \times 2^X \times \Phi(X)$ gives the set of transitions.

An edge $(s, s', a, \lambda, \delta)$ represents a transition from state $s$ to state $s'$ on input symbol $a$. The set $\lambda \subseteq X$ gives the clocks to be reset with this transition, and $\delta$ is a clock constraint over $X$.
- $F \subseteq S$ is a set of accepting states.
Example: TBA

\[ A = (\Sigma, S, S_0, X, E, F) \]
\[(s, s', a, \lambda, \delta) \in E\]

(Accepting) TBA Runs

**Definition.** A run \( r \), denoted by \((\bar{s}, \bar{\nu})\), of a TBA \((\Sigma, S, S_0, X, E, F)\) over a timed word \((\sigma, \tau)\) is an **infinite** sequence of the form

\[ r : (s_0, \nu_0) \xrightarrow{\tau_1} (s_1, \nu_1) \xrightarrow{\tau_2} (s_2, \nu_2) \xrightarrow{\tau_3} \ldots \]

with \( s_i \in S \) and \( \nu_i : X \to \mathbb{R}_+^+ \), satisfying the following requirements:

- **Initiation:** \( s_0 \in S_0 \) and \( \nu(x) = 0 \) for all \( x \in X \).
- **Consecution:** for all \( i \geq 1 \), there is an edge in \( E \) of the form \((s_{i-1}, s_i, \sigma_i, \lambda_i, \delta_i)\) such that
  - \( (\nu_{i-1} + (\tau_i - \tau_{i-1})) \) satisfies \( \delta_i \) and
  - \( \nu_i = (\nu_{i-1} + (\tau_i - \tau_{i-1}))[\lambda_i := 0] \).

The set \( \inf(r) \subseteq S \) consists of those states \( s \in S \) such that \( s = s_i \) for infinitely many \( i \geq 0 \).

**Definition.** A run \( r = (\bar{s}, \bar{\nu}) \) of a TBA over timed word \((\sigma, \tau)\) is called **accepting** (run) if and only if \( \inf(r) \cap F \neq \emptyset \).
Example: (Accepting) Runs

\[ r : (s_0, \nu_0) \xrightarrow{\tau_1} (s_1, \nu_1) \xrightarrow{\sigma_2} (s_2, \nu_2) \xrightarrow{\tau_3} \ldots \text{ initial and } (s_{i-1}, s_i, \sigma_i, \lambda_i, \delta_i) \in E, \text{ s.t. } (\nu_{i-1} + (\tau_i - \tau_{i-1})) = \delta_i, \nu_i = (\nu_{i-1} + (\tau_i - \tau_{i-1})) \lfloor \lambda_i = 0 \lfloor. \text{ Accepting iff } \text{inf}(r) \cap F \neq \emptyset. \]

**Timed word:** \( (a, 1), (b, 2), (a, 3), (b, 4), (a, 5), (b, 6), \ldots \)

- **Can we construct any run?** Is it accepting?

\[ \langle s_0, 0 \rangle \xrightarrow{a} \langle s_1, 0 \rangle \xrightarrow{b} \langle s_2, 1 \rangle \xrightarrow{a} \langle s_3, 1 \rangle \ldots \checkmark \]

- **Can we construct a non-run?**

- **Can we construct a (non-)accepting run?**

**The Language of a TBA**

**Definition.** For a TBA \( \mathcal{A} \), the language \( L(\mathcal{A}) \) of timed words it accepts is defined to be the set

\[ \{ (\sigma, \tau) \mid \mathcal{A} \text{ has an accepting run over } (\sigma, \tau) \}. \]

For short: \( L(\mathcal{A}) \) is the language of \( \mathcal{A} \).

**Definition.** A timed language \( L \) is a timed regular language if and only if \( L = L(\mathcal{A}) \) for some TBA \( \mathcal{A} \).
Example: Language of a TBA

\[ L(A) = \{ (\sigma, \tau) \mid A \text{ has an accepting run over } (\sigma, \tau) \} \]

Claim:

\[ L(A) = L_{\text{crt}} = \{ ((ab)^\omega, \tau) \mid \exists i \forall j \geq i : (\tau_{2j} < \tau_{2j-1} + 2) \} \]

Question: Is \( L_{\text{crt}} \) timed regular or not?

The Universality Problem is Undecidable for TBA

[Alur and Dill, 1994]
The Universality Problem

• **Given:** A TBA $A$ over alphabet $\Sigma$.
• **Question:** Does $A$ accept all timed words over $\Sigma$?
  In other words: Is $L(A) = \{(\sigma, \tau) \mid \sigma \in \Sigma^\omega, \tau \text{ time sequence}\}$.

\[ \Sigma = \{a, b, c\} \quad A: \]

**Theorem 5.2.** The problem of deciding whether a timed automaton over alphabet $\Sigma$ accepts all timed words over $\Sigma$ is $\Pi^1_1$-hard.

(“The class $\Pi^1_1$ consists of highly undecidable problems, including some nonarithmetical sets (for an exposition of the analytical hierarchy consult, see for instance [Rogers, 1967].)

**Recall:** With classical Büchi Automata (untimed), this is different:

• Let $B$ be a Büchi Automaton over $\Sigma$.
• $B$ is universal if and only if $L(B) = \emptyset$.
• $B'$ such that $L(B') = \overline{L(B)}$ is effectively computable.
• Language emptyness is decidable for Büchi Automata.
Proof Idea

Theorem 5.2. The problem of deciding whether a timed automaton over alphabet $\Sigma$ accepts all timed words over $\Sigma$ is $\Pi^1_1$-hard.

Proof Idea:
• Consider a language $L_{\text{undec}}$ which consists of the recurring computations of a 2-counter machine $M$.
• Construct a TBA $A$ from $M$ which accepts the complement of $L_{\text{undec}}$, i.e. with $L(A) = \overline{L_{\text{undec}}}$.
• Then $A$ is universal if and only if $L_{\text{undec}}$ is empty... which is the case if and only if $M$ doesn’t have a recurring computation.

Once Again: 2-Counter Mach. (Different Flavour)

A two-counter machine $M$
• has two counters $C$, $D$ and
• a finite program consisting of $n$ instructions.
• An instruction increments or decrements one of the counters, or jumps, here even non-deterministically.
• A configuration of $M$ is a triple $\langle i, c, d \rangle$:
  program counter $i \in \{1, \ldots, n\}$, values $c, d \in \mathbb{N}_0$ of $C$ and $D$.
• A computation of $M$ is an infinite consecutive sequence
  \[ \langle 1, 0, 0 \rangle = \langle i_0, c_0, d_0 \rangle, \langle i_1, c_1, d_1 \rangle, \langle i_2, c_2, d_2 \rangle, \ldots \]
that is, $\langle i_j+1, c_{j+1}, d_{j+1} \rangle$ is a result executing instruction $i_j$ at $\langle i_j, c_j, d_j \rangle$.
A computation of $M$ is called recurring iff $i_j = 1$ for infinitely many $j \in \mathbb{N}_0$. 
Step 1: The Language of Recurring Computations

- Let $M$ be a 2CM with $n$ instructions.

**Wanted:** A timed language $L_{\text{undec}}$ (over some alphabet) representing exactly the recurring computations of $M$.
(In particular s.t. $L_{\text{undec}} = \emptyset$ if and only if $M$ has no recurring computation.)

- Choose $\Sigma = \{b_1, \ldots, b_n, a_1, a_2\}$ as alphabet.
- We represent a configuration $\langle i, c, d \rangle$ of $M$ by the sequence

$$b_1 \ a_1 \ldots \ a_1 \ a_2 \ldots \ a_2 = b_1a_1^ca_2^d$$

![Diagram](image)

Step 1: The Language of Recurring Computations

Let $L_{\text{undec}}$ be the set of the timed words $(\sigma, \tau)$ with

- $\sigma$ is of the form $b_i, a_1^c a_2^d b_i, a_1^c a_2^d b_i, \ldots$
- $\langle i_1, c_1, d_1 \rangle, \langle i_2, c_2, d_2 \rangle, \ldots$ is a recurring computation of $M$.

For all $j \in \mathbb{N}_0$.

- the time of $b_{ij}$ is $j$.
- if $c_{j+1} = c_j$:
  for every $a_1$ at time $t$ in the interval $[j, j+1]$ there is an $a_1$ at time $t+1$,
- if $c_{j+1} = c_j + 1$:
  for every $a_1$ at time $t$ in the interval $[j+1, j+2]$, except for the last one, there is an $a_1$ at time $t - 1$,
- if $c_{j+1} = c_j - 1$:
  for every $a_1$ at time $t$ in the interval $[j, j+1]$, except for the last one, there is an $a_1$ at time $t + 1$,

And analogously for the $a_2$’s.
Step 2: Construct “Observer” for $L_{\text{undec}}$

**Wanted:** A TBA $\mathcal{A}$ such that $L(\mathcal{A}) = \overline{L_{\text{undec}}}$, i.e., $\mathcal{A}$ accepts a timed word $(\sigma, \tau)$ if and only if $(\sigma, \tau) \notin L_{\text{undec}}$.

**Approach:** What are the reasons for a timed word not to be in $L_{\text{undec}}$?

**Recall:** $(\sigma, \tau)$ is in $L_{\text{undec}}$ if and only if:

- $\sigma = b_1a_1^0a_2^1b_1a_1^2a_2^0$
- $\langle i_1, c_1, d_1 \rangle, \langle i_2, c_2, d_2 \rangle, \ldots$ is a recurring computation of $M$.
- the time of $b_{i_j}$ is $j$.
- if $c_{j+1} = c_j (= c_j + 1, = c_j - 1)$: 
  
  (i) The $b_i$ at time $j \in \mathbb{N}$ is missing, or there is a spurious $b_i$ at time $t \in \mathbb{N}$.
  (ii) The prefix of the timed word with times $0 \leq t < 1$ doesn’t encode $\langle 1, 0, 0 \rangle$.
  (iii) The timed word is not recurring, i.e., it has only finitely many $b_i$.
  (iv) The configuration encoded in $[j + 1, j + 2]$ doesn’t faithfully represent the effect of instruction $b_i$ on the configuration encoded in $[j, j + 1]$.

**Plan:** Construct a TBA $\mathcal{A}_0$ for case (i), a TBA $\mathcal{A}_{\text{init}}$ for case (ii), a TBA $\mathcal{A}_{\text{recur}}$ for case (iii), and one TBA $\mathcal{A}_i$ for each instruction for case (iv).

Then set

$$\mathcal{A} = \mathcal{A}_0 \cup \mathcal{A}_{\text{init}} \cup \mathcal{A}_{\text{recur}} \cup \bigcup_{1 \leq i \leq n} \mathcal{A}_i$$
Step 2.(i): Construct $A_0$

(i) The $b_i$ at time $j \in \mathbb{N}$ is missing, or there is a spurious $b_i$ at time $t \in [j, j+1[$.

[Alur and Dill, 1994]: “It is easy to construct such a timed automaton.”

Step 2.(ii): Construct $A_{init}$

(ii) The prefix of the timed word with times $0 \leq t < 1$ doesn’t encode $(1, 0, 0)$.

- It accepts

$$\{(\sigma_j, \tau_j)_{j \in \mathbb{N}_0} \mid (\sigma_0 \neq b_1) \lor (\tau_0 \neq 0) \lor (\tau_1 \neq 1)\}.$$
Step 2.(iii): Construct $A_{\text{recur}}$

(iii) The timed word is not recurring, i.e. it has only finitely many $b_i$.

- $A_{\text{recur}}$ accepts words with only finitely many $b_i$.

Step 2.(iv): Construct $A_i$

(iv) The configuration encoded in $[j+1, j+2]$ doesn’t faithfully represent the effect of instruction $b_j$ on the configuration encoded in $[j, j+1]$.

**Example**: assume instruction 7 is:

Increment counter $D$ and jump non-deterministically to instruction 3 or 5.

**Once again**: stepwise. $A_7$ is $A_7^1 \cup \cdots \cup A_7^6$.

- $A_7^1$ accepts words with $b_7$ at time $j$ but neither $b_3$ nor $b_5$ at time $j + 1$.
  - “Easy to construct.”
- $A_7^2$ is

\[
\begin{array}{c}
\text{f}_0 \\
 x \geq 0 \\
 b_7
\end{array} \quad \begin{array}{c}
\text{f}_1 \\
 x < 1 \\
 a_1
\end{array} \quad \begin{array}{c}
\text{f}_2 \\
 x \neq 1 \\
 \neg a_1, x = 1
\end{array}
\]

- $A_7^3$ accepts words which encode unexpected increment of counter $C$.
- $A_7^4, \ldots, A_7^6$ accept words with missing decrement of $D$. 

Consequences: Language Inclusion

- **Given:** Two TBAs $\mathcal{A}_1$ and $\mathcal{A}_2$ over alphabet $B$.
- **Question:** Is $\mathcal{L}(\mathcal{A}_1) \subseteq \mathcal{L}(\mathcal{A}_2)$?

**Possible applications of a decision procedure:**
- Characterise the allowed behaviour as $\mathcal{A}_2$ and model the design as $\mathcal{A}_1$.
- Automatically check whether the behaviour of the design is a subset of the allowed behaviour.

- If **language inclusion** was decidable, then we could use it to decide universality of $\mathcal{A}$ by checking

$$\mathcal{L}(\mathcal{A}_{\text{univ}}) \subseteq \mathcal{L}(\mathcal{A})$$

where $\mathcal{A}_{\text{univ}}$ is any universal TBA (which is easy to construct).
Consequences: Complementation

- **Given**: A timed regular language $W$ over $B$ (that is, there is a TBA $A$ such that $L(A) = W$).
- **Question**: Is $\overline{W}$ timed regular?

Possible applications of a decision procedure:
- Characterise the allowed behaviour as $A_2$ and model the design as $A_1$.
- Automatically construct $A_3$ with $L(A_3) = L(A_2)$ and check
  
  $$L(A_1) \cap L(A_3) = \emptyset,$$
  
  that is, whether the design has any non-allowed behaviour.
- Taking for granted that:
  - The intersection automaton is effectively computable.
  - The emptiness problem for Büchi automata is decidable. (Proof by construction of region automaton [Alur and Dill, 1994].)

Consequences: Complementation

- **Given**: A timed regular language $W$ over $B$ (that is, there is a TBA $A$ such that $L(A) = W$).
- **Question**: Is $\overline{W}$ timed regular?

- If the class of timed regular languages were closed under complementation, "the complement of the inclusion problem is recursively enumerable. This contradicts the $\Pi_1^1$-hardness of the inclusion problem." [Alur and Dill, 1994]

A non-complementable TBA $A$:

$$L(A) = \{(a^\omega, (t_i)_{i \in \mathbb{N}_0}) \mid \exists i \in \mathbb{N}_0 \exists j > i : (t_j = t_i + 1)\}$$

Complement language:

$$\overline{L(A)} = \{(a^\omega, (t_i)_{i \in \mathbb{N}_0}) \mid \text{no two } a \text{ are separated by distance } 1\}.$$
Beyond Timed Regular

With clock constraints of the form

\[ x + y \leq x' + y' \]

we can describe timed languages which are not timed regular.

**In other words:**
- There are strictly more timed languages than timed regular languages.
- There exists timed languages \( L \) such that there exists no \( A \) with \( L(A) = L \).

**Example:**

\[
\left\{ ((abc)^{\omega}, \tau) \mid \forall j. (\tau_{3j} - \tau_{3j-1}) = 2(\tau_{3j-1} - \tau_{3j-2}) \right\}
\]
**hat is a 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.
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.