Contents & Goals

Last Lecture:
- Timed Büchi Automata and timed regular languages [Alur and Dill, 1994].

This Lecture:
- Educational Objectives: Capabilities for following tasks/questions.
  - What’s a TBA and what’s the difference to (extended) TA?
  - What’s undecidable for timed (Büchi) automata?
  - What’s the idea of the proof?

- Content:
  - The Universality Problem is undecidable for TBA [Alur and Dill, 1994] Cont’d
  - Timed regular languages are not everything.
Recall: Timed Languages

**Definition.** A **time sequence** $\tau = \tau_1, \tau_2, \ldots$ is an infinite sequence of time values $\tau_i \in \mathbb{R}_0^+$, 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 $t \in \mathbb{R}_0^+$, 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 $\Sigma$. 
Recall: Timed Büchi Automata

Definition. The set $\Phi(X)$ of clock constraints over $X$ is defined inductively by

$$\delta ::= x \leq c \mid c \leq x \mid -\delta \mid \delta_1 \land \delta_2$$

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 $\lambda$ 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.

Recall: (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

$$r : (s_0, \nu_0) \overset{\sigma_1}{\longrightarrow} (s_1, \nu_1) \overset{\sigma_2}{\longrightarrow} (s_2, \nu_2) \overset{\sigma_3}{\longrightarrow} \ldots$$

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 $\text{inf}(r) \subseteq S$ 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 (an) accepting (run) if and only if $\text{inf}(r) \cap F \neq \emptyset$. 
Recall: 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 $\mathcal{A}$.

The Universality Problem is Undecidable for TBA

[Alur and Dill, 1994]
Recall: The Universality Problem

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

**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 nonarithmetic sets
for an exposition of the analytical hierarchy consult, for instance [Rogers, 1967].")

**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 $L(A) = L_{\text{undec}}$.
- 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: Two Counter Machines (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 \( (i, c, d) \):
  
  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 such that \( L_{\text{undec}} = \emptyset \) 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 \( (i, c, d) \) of \( M \) by the sequence

\[
b_1a_1\ldots a_1a_2\ldots a_2 = b_1^{c}a_1^{d}a_2^{d}
\]

\( c \) times \( d \) times
Step 1: The Language of Recurring Computations

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

- $\sigma$ is of the form $b_i a_1^{c_i} a_2^{d_i} b_i a_1^{c_i} a_2^{d_i} \ldots$
- $\langle i_1, c_1, d_1 \rangle, \langle i_2, c_2, d_2 \rangle, \ldots$ is a recurring computation of $M$.
- For all $j \in \mathbb{N}_0$,
  - the time of $b_{ij}$ is $j$.
  - if $c_{j+1} = c_j$ then for every $a_1$ at time $t$ in the interval $[j, j+1]$ there is an $a_1$ at time $t + 1$,
  - if $c_{j+1} = c_j + 1$ then 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 $c_{j+1} = c_j - 1$ then 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 $A$ such that

$$L(A) = L_{\text{undec}}$$

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

1. The $b_i$ at time $j \in \mathbb{N}$ is missing, or there is a spurious $b_i$ at time $t \in ]j, j + 1[.$
2. The prefix of the timed word with times $0 \leq t < 1$ doesn’t encode $(1, 0, 0)$.
3. The timed word is not recurring, i.e. it has only finitely many $b_i$.
4. The configuration encoded in $]j + 1, j + 2[$ doesn’t faithfully represent the effect of instruction $b_i$ on the configuration encoded in $]j, j + 1[$.
Step 2: Construct “Observer” for $L_{\text{undec}}$

**Wanted:** A TBA $A$ such that

$$L(A) = L_{\text{undec}}$$

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

(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[.

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

(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 $A_0$ for case (i), a TBA $A_{\text{init}}$ for case (ii), a TBA $A_{\text{recur}}$ for case (iii), and one TBA $A_i$ for each instruction for case (iv).

Then set

$$A = A_0 \cup A_{\text{init}} \cup A_{\text{recur}} \cup \bigcup_{1 \leq i \leq n} A_i$$

---

**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 $A_{\text{init}}$

(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 $b_\#$.

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

---
Step 2.(iv): Construct $A_i$

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

Example: assume instruction $b_7$ is:
Increment counter $D$ and jump non-deterministically to instruction 3 or 5.

Once again: stepwise. $A_7$ is $A_1 \cup \cdots \cup A_6 \cup A_7$.

- $A_7^1$ 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

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

Aha, And...?
Consequences: Language Inclusion

- **Given**: Two TBAs $A_1$ and $A_2$ over alphabet $B$.
- **Question**: Is $L(A_1) \subseteq L(A_2)$?

Possible applications of a decision procedure:

- Characterise the allowed behaviour as $A_2$ and model the design as $A_1$.
- 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 $A$ by checking
  $$L(A_{\text{univ}}) \subseteq L(A)$$
  where $A_{\text{univ}}$ 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 $L(A_3) = \overline{L(A_2)}$ 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 $A$:

\[
\begin{align*}
    &a & a & a \\
    &x := 0 & x = 1 & \\
    &\emptyset & \emptyset & \emptyset
\end{align*}
\]

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

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
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 timed languages than timed regular languages.
- There exists timed languages \( L \) such that there exists no \( A \) with \( L(A) = L \).

**Example:**

\[
\{(abc)^\omega, \tau) \mid \forall j, (\tau_{3j} - \tau_{3j-1}) = 2(\tau_{3j-1} - \tau_{3j-2}) \}
\]

References
References
