Recall: Timed Büchi Automata

A Timed Büchi Automaton (TBA) is defined as a pair 
$$ \langle R, \tau, \nu \rangle $$

where:
- $R$ is a rational constant.
- $\tau$ is a sequence of the form
  $$ \nu = \langle \nu_1, \nu_2, \ldots, \nu_i \rangle $$
  such that
  $$ \nu_i = \langle \sigma, \tau \rangle $$
  for some $\sigma \in \Sigma$, $\tau \in \mathbb{N}$. 
- $\nu$ is a rational constant.

Recall: Timed Languages

The Universality Problem is undecidable for TBA.
Recall: The Language of a TBA

Definition. For a TBA \( A \), the language \( L(A) \) of timed words it accepts is defined to be the set \( \{ (\sigma, \tau) | A \text{ has an accepting run over } (\sigma, \tau) \} \).

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

Definition. A timed language \( L \) is a timed regular language if and only if \( L = L(A) \) for some TBA \( A \).

The Universality Problem is Undecidable for TBA

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

Theorem 5.2. The problem of deciding whether a timed automaton over alphabet \( \Sigma \) accepts all timed words over \( \Sigma \) is \( \Pi_{11} \)-hard.

("The class \( \Pi_{11} \) consists of highly undecidable problems, including some non-arithmetic sets; 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 \( L_{\text{undec}} = \emptyset \). . . 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 \( \langle i, c, d \rangle \):
  - 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, \( \langle i_j, c_j, d_j \rangle \) is a result executing instruction \( i_j \) at \( \langle i_j, c_j, d_j \rangle \).
• A computation of \( M \) is called recurring if \( i_j = 1 \) 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 \( \langle i, c, d \rangle \) of \( M \) by the sequence \( b_i a_1 \ldots a_1 \ldots c \times a_2 \ldots a_2 \ldots d \times \ldots = b_1 a_1 c \times a_2 d \times \ldots \).
Step 2: Construct "Observer" for $L$.

The Language of Recurring Computations

(i) The prefix of the timed word with times of instruction doesn't encode $t < 0$, i.e. $i$ has only finitely many $\langle \sigma, \tau \rangle$.

(ii) The configuration encoded in $W$ doesn't encode $t < 0$, i.e. $i$ has only finitely many $\langle \sigma, \tau \rangle$.

(iii) The configuration encoded in $W$ doesn't encode $t < 0$, i.e. $i$ has only finitely many $\langle \sigma, \tau \rangle$.

(iv) The configuration encoded in $W$ doesn't encode $t < 0$, i.e. $i$ has only finitely many $\langle \sigma, \tau \rangle$.

It is easy to construct such a timed automaton.
Step 2.(ii): Construct

The prefix of the timed word with times $0 \leq t < 1$ doesn't encode $\langle 1, 0, 0 \rangle$.

• It accepts $\{ (\sigma_j, \tau_j) | j \in \mathbb{N}_0 | (\sigma_0 \neq b_1) \lor (\tau_0 \neq 0) \lor (\tau_1 \neq 1) \}$.

Step 2.(iii): Construct

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

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 7 is:

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

Once again: stepwise.

$A_7$ is $A_{17} \cup \cdots \cup A_{67}$.

• $A_{17}$ accepts words with $b_7$ at time $j$ but neither $b_3$ nor $b_5$ at time $j+1$.

"Easy to construct."

• $A_{27}$ is $\ell_0 \ell_1 \ell_2^* b_7 x := 0^* a_1 x < 1 x := 0 \neg a_1, x \neq 1 x \neq 1$.

• $A_{37}$ accepts words which encode unexpected increment of counter $C$.

• $A_{47}, \ldots, A_{67}$ accept words with missing decrement of $D$.

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:

• Characterize the allowed behavior as $A_2$ and model the design as $A_1$.

• Automatically check whether the behavior of the design is a subset of the allowed behavior.

• 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 $W$ timed regular?

Possible applications of a decision procedure:

• Characterize the allowed behavior 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 behavior.

• 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 \( W \) 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 \( \Pi_{11} \)-hardness of the inclusion problem." [Alur and Dill, 1994]

An non-complementable TBA \( A \):

\[
\begin{align*}
\ell_1 &:= 0 \\
\ell_0 &:= a, x := 0 \\
\ell_2 &:= b, y := 0 \\
2x &:= 3 \\
y &:= 0 \\
\end{align*}
\]

\[L(A) = \{(a\omega, (t_i)i \in \mathbb{N}_0) | \exists i \in \mathbb{N}_0 \exists j > i : (t_j = t_i + 1)\}\].

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 timed languages that are not timed regular.

• There exist timed languages \( L \) such that there exists no \( A \) with \( L(A) = L \).

Example:

\[
\begin{align*}
(1+1-1) &:= (1+1-1) x \quad \text{for } x := 0 \\
(1+1-1) &:= (1+1-1) y \\
(1+1-1) &:= (1+1-1) z \\
\end{align*}
\]

\[L(A) = \{(1+1-1)\omega, \tau \} | \forall j. (\tau_3 j - \tau_3 j - 1) = 2(\tau_3 j - 1 - \tau_3 j - 2)\}\].

References
