NSWI101: System Behaviour Models and Verification

7. Timed Automata

Jan Kofroň

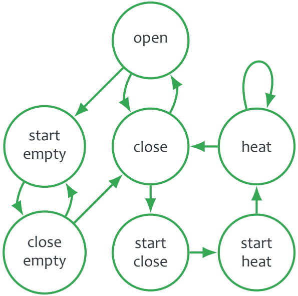
FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

Department of
Distributed and
Dependable
Systems

Today

- Timed Automata

Timed Automata



System model

AG (start → AF heat)

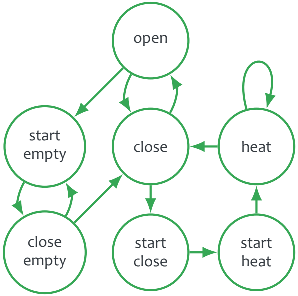
Property specification

Model Checker

Property satisfied

Property violated

Timed Automata



System model

Timed CTL

AG (start → AF heat)

Property specification

Model Checking

Model Checker

Property satisfied

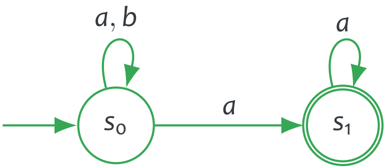
Property violated

Recall: Büchi Automata

Büchi automaton is finite automaton accepting infinite words

Word is accepted if:

- An accepting state is visited infinitely many times (standard case)
- A state from each accepting set is visited infinitely many times (generalized case)



$$L_{BA} = (a + b)^* a^\omega$$

Timed Languages

Timed sequence $t = t_1 t_2 t_3 \ldots$ is infinite sequence of values $t_i \in \mathbb{R}^+$ satisfying:

- monotonicity: $\forall i \geq 1 : t_i < t_{i+1}$
- progress: $\forall x \in \mathbb{R} : \exists i \geq 1 : t_i > x$

Timed word is a tuple (s, t) where s is infinite sequence of alphabet symbols and t is timed sequence.

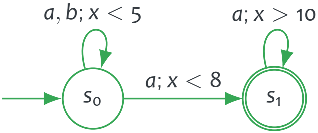
TIMED AUTOMATON

In addition to Büchi automaton, Timed automaton contains set of real variables representing *clocks*.

Each clock variable:

- is initially set to 0
- increments at the same speed as any other clock variable
- can be reset to 0 at any transition
- (co-)defines guard upon transitions

Timed automaton accepts timed language, i.e., (finite or infinite) set of timed words.



Clock Constraints

Given set of clock X, set $\Phi(X)$ of clock constraints δ is defined:

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

where $x \in X$, $c \in \mathbb{Q}$.

Timed Automaton

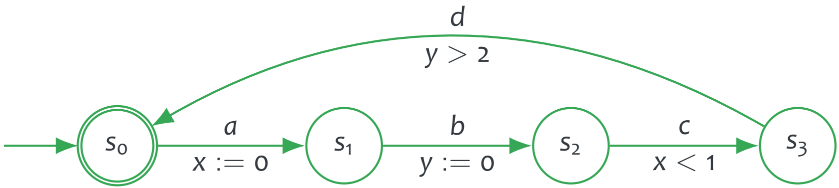
Nondeterministic timed automaton A is 6-tuple $(\Sigma, S, S_0, C, E, F)$:

- Σ is finite alphabet
- S is finite set of states
- $S_0 \subseteq S$ is set of initial states
- C is finite set of clocks
- $E \subseteq S \times S \times \Sigma \times 2^C \times \Phi(C)$ is transition relation
  - 2^C denotes the set of clocks to be reset at transition
  - $\Phi(C)$ is clock constraint over C
- $F \subseteq S$ is set of accepting states

Example

Automaton accepting language:

$$\left\{\left((abcd)^\omega, t\right) \middle| \forall j : \left((t_{4j+3} < t_{4j+1} + 1) \wedge (t_{4j+4} > t_{4j+2} + 2)\right)\right\}$$



Properties of Timed Automata

Question: Is class of timed regular languages closed under **finite union**?

- Yes

Proof: Since the TA are nondeterministic, union is represented by disjoint union of particular automata (similar to Büchi automata).

Properties of Timed Automata

Question: Is class of timed regular languages closed under **intersection**?

- Yes

Proof: Simple modification of intersection of Büchi automata: Let A be automaton accepting the intersection of languages of A_1 and A_2 and C_i be set of clocks. Transitions of A are $\big((s_1, s_2, i), (s'_1, s'_2, j), a, \lambda, \varphi\big)$:

- $(s_1, s_2, i), (s'_1, s'_2, j), a$ as in case of intersection of two Büchi automata
- $\lambda = \lambda_1 \cup \lambda_2$ is set of clocks to be reset
- $\varphi = \varphi_1 \wedge \varphi_2$ is transition constraint

Assuming disjunct sets of clocks, states, and transitions, alphabet is shared, though.

Question: Is class of timed regular languages closed under **complement**?

- No

Even worse—inclusion of timed languages $L(A) \subseteq L(B)$ is **undecidable** problem.

Model Checking of Timed Automata

- Verification of properties realized by checking of language emptiness, similarly to LTL model checking.
- Systematic exploration of timed automata not feasible due to infinite (usually even uncountable) number of possible clock valuations.
- **Idea:** Constructing "equivalent" Büchi automaton accepting the same language up to timing as original timed automaton.
  - corresponding Büchi automaton is called *region automaton*.

Region Automaton

- States of region automaton are *regions*.
- Each region corresponds to state and set of equivalent clock valuations of original timed automaton.
- Transformation to region automaton solves the problem of uncountable many clock valuations disallowing systematic exploration of state space.

Clock Regions

Given timed automaton A, (s, n) denotes *extended state*

- s is state of A
- n is clock interpretation (i.e., valuation of clock variables)

For $t \in \mathbb{R} : t = \lfloor t \rfloor + \text{fract}(t)$.

Clock Regions

Let $A = (\Sigma, S, S_0, C, E, F)$ be timed automaton.

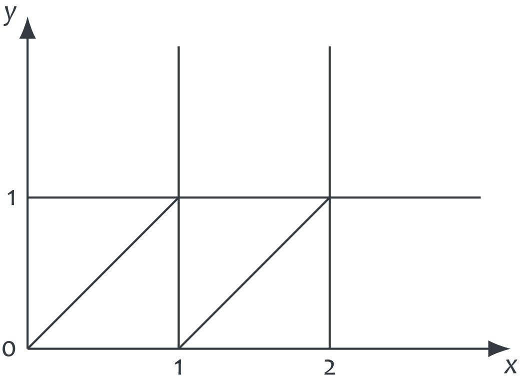
For $x \in C$, by c_x denote largest c such that $x \leq c$ or $c \leq x$ is sub-formula of some clock constraints in F

Clock valuations n, n' are equivalent ($n \sim n'$) iff:

1. $\forall x \in C : \lfloor n(x) \rfloor = \lfloor n'(x) \rfloor \vee (\lfloor n(x) \rfloor > c_x \wedge \lfloor n'(x) \rfloor > c_x)$ and
2. $\forall x, y \in C : n(x) \leq c_x \wedge n(y) \leq c_y \implies$
   $\mathrm{fract}(n(x)) \leq \mathrm{fract}(n(y)) \Leftrightarrow \mathrm{fract}(n'(x)) \leq \mathrm{fract}(n'(y))$ and
3. $\forall x \in C : \lfloor n(x) \rfloor \leq c_x \implies \mathrm{fract}(n(x)) = 0 \Leftrightarrow \mathrm{fract}(n'(x)) = 0$

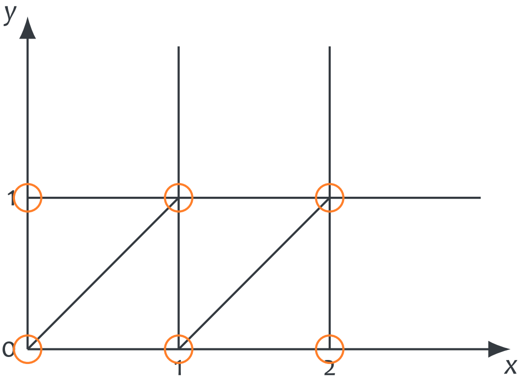
Clock region for A is equivalence class induced by $\sim$.

Example of Clock Regions



Example of Clock Regions

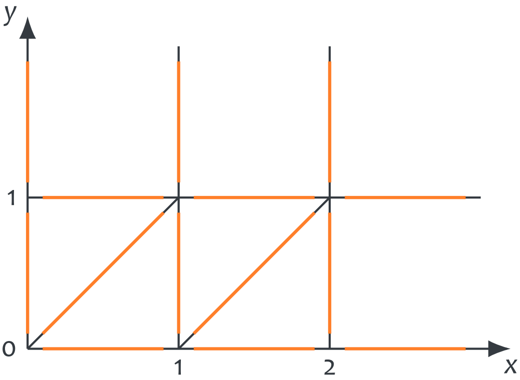
6 corner regions



Example of Clock Regions

6 corner regions

14 open line segments

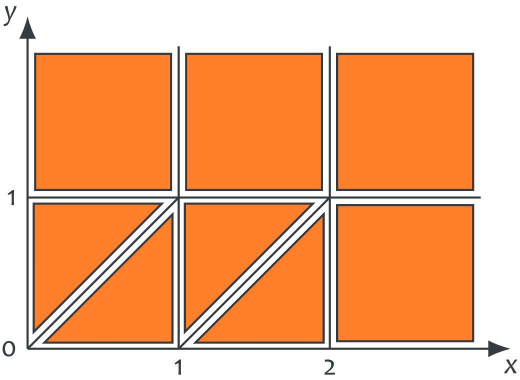


Example of Clock Regions

6 corner regions

14 open line segments

8 open regions



Region Successors

Clock region b is successor of clock region a ($b \in succ(a)$) iff
$\forall n \in a : \exists t \in \mathbb{R}^+ : n + t \in b$.

Successor regions are those that can be reached by incrementing time, including resetting clocks.

Region Successors

1. $\forall x \in C : x > x_c \implies succ(a) = \{a\}$
2. Let C_0 be set of clocks such that $(x = c) \in a$. Values of $x \in C$ for $b = succ(a)$ satisfy:
   - if $x = c_x$, then b satisfies $x > x_c$, otherwise $c < x < c + 1$.
   - if $x \notin C_0$, constraint for x in a equals to the one in b.
3. Let C_1 be set of clocks such that region a does **not** satisfy $x > c_x$ and $\forall y \in C_1 : \text{fract}(y) \leq \text{fract}(x)$. Define clock region b as:
   - $\big(\forall x \in C_1 : (c - 1 < x < c) \in a \implies (x = c) \in b\big) \wedge$ $\big(\forall x \notin C_1 : \text{constraint for } x \text{ in } a \text{ equals to the one in } b\big)$.
   - $\forall x, y \notin C_1 : \text{fract}(x) \leq \text{fract}(y)$ in $a \Leftrightarrow \text{fract}(x) \leq \text{fract}(y)$ in b.

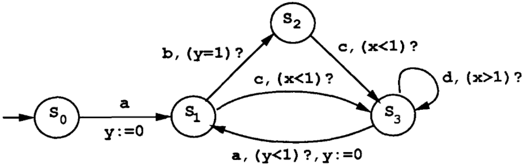
   $succ(a) = \{a, b, succ(b)\}$

Region Automaton

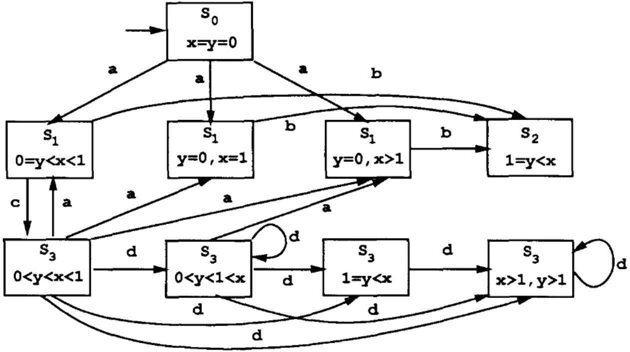
For timed automaton $A = (\Sigma, S, S_0, C, E, F)$, corresponding region automaton $R(A)$ is defined as follows:

- States of $R(A)$ are (s, a) where $s \in S$ and a is clock region.
- Initial states are $(s_0, [n_0])$ where $s_0 \in S_0$ and $n_0(x) = 0$ for $x \in C$.
- $\big((s, a), (s', a'), m\big)$ is transition of $R(A)$ iff $\exists (s, s', m, \lambda, \varphi) \in E$ and there exists region a'' such that:
  - a'' is successor of a
  - a'' satisfies φ
  - $a' = [\lambda \rightarrow 0]a''$

EXAMPLE



Figures from: Rajeev Alur, David L. Dill, A theory of timed automata, Theoretical Computer Science, Volume 126, Issue 2, 1994



Lemma: If r is *progressive* run of $R(A)$ over s, then there exists time sequence t and run r' of A over (s, t) such that $r = [r']$.

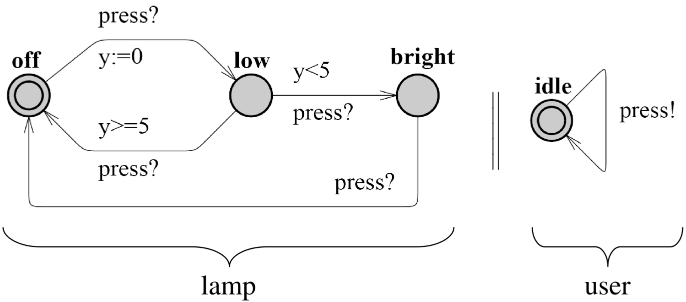
- *progressive*—no bound for any clock
- w.l.o.g. we can assume just progressive runs
- $[r]$ means "untiming" r

Theorem: For timed automaton A, there exists Büchi automaton $R(A)$ that accepts $Untime\big(L(A)\big)$.

Idea: Construct region automaton $R(A)$ with accepting states $\{(s, a) | s \in F\}$. $R(A)$ is special case of Büchi automaton.

NETWORK OF TIMED AUTOMATA

- For modelling communicating parts of system in independent way
- Each part represented by a single TA, which Communicates with other parts through input/output actions
- Composition realized by parallel synchronous product



Example from: Enoiu, E.P., Čaušević, A., Ostrand, T.J. et al. Automated test generation using model checking: an industrial evaluation. Int J Softw Tools Technol Transfer 18, 2016´

UPPAAL

- Tool for verification of TA models
- Academic, but quite well established and used in industry nowadays
- Supports modelling, verification, simulation
- Successfully applied on communication protocols, multimedia applications, ...
- Available at `http://www.uppaal.org/` and `http://www.uppaal.com`