

Modeling and analysis of hybrid systems

Timed automata

Prof. Dr. Erika Ábrahám

Informatik 2 - Theory of Hybrid Systems
RWTH Aachen

SS 2010

Christel Baier and Joost-Pieter Katoen:
Principles of Model Checking

1 Motivation

2 Timed automata

3 TCTL

*Correctness in **time-critical** systems not only depends on the logical result of the computation but also on the time at which the results are produced.*

Thus if we model such systems, we also need to model the time.
The first choice in modeling: discrete or continuous time?

Discrete-time systems

- conceptually simple
- each action lasts for a single **time unit (tick)**
- action α lasts $k > 0$ time units $\leadsto k - 1$ ticks followed by α
- leads to large transition systems
- minimal time between two actions is a multiple of the tick
- **logic**: CTL or LTL extended with syntactic sugar

$(\mathcal{X}\varphi)$ $\bigcirc\varphi$: φ holds after one tick

$(\mathcal{X}^k\varphi)$ $\bigcirc^k\varphi$: φ holds after k ticks

$(\mathcal{F}^{\leq k}\varphi)$ $\diamond^{\leq k}\varphi$: φ occurs within k ticks

We deal in this lecture with **continuous-time** models.

Contents

1 Motivation

2 Timed automata

3 TCTL

Timed automata

- Measure time: finite set \mathcal{C} of **clocks** x, y, z, \dots
- Clocks increase their value implicitly as time progresses
- All clocks proceed at **rate 1**
- Limited clock access:

Reading: **Clock constraints**

$g ::= x < c \mid x \leq c \mid x > c \mid x \geq c \mid g \wedge g$

with $c \in \mathbb{N}$ ($c \in \mathbb{Q}$) and $x \in \mathcal{C}$.

Syntactic sugar: $true, \quad x \in [c_1, c_2), \quad c_1 \leq x < c_2, \quad x = c, \dots$

$ACC(\mathcal{C})$: set of atomic clock constraints over \mathcal{C}

$CC(\mathcal{C})$: set of clock constraints over \mathcal{C}

Writing: **Clock reset** sets value to 0

Definition

For a set \mathcal{C} of clocks, $x \in \mathcal{C}$, $\nu \in V_{\mathcal{C}}$, $c \in \mathbb{N}$, and $g, g' \in CC(\mathcal{C})$, let $\models \subseteq V_{\mathcal{C}} \times CC(\mathcal{C})$ be defined by

$$\begin{aligned}\nu \models x < c & \text{ iff } \nu(x) < c \\ \nu \models x \leq c & \text{ iff } \nu(x) \leq c \\ \nu \models x > c & \text{ iff } \nu(x) > c \\ \nu \models x \geq c & \text{ iff } \nu(x) \geq c \\ \nu \models g \wedge g' & \text{ iff } \nu \models g \text{ and } \nu \models g'\end{aligned}$$

Definition

- For a set \mathcal{C} of clocks, $\nu \in V_{\mathcal{C}}$, and $c \in \mathbb{N}$ we denote by $\nu + c$ the valuation with $(\nu + c)(x) = \nu(x) + c$ for all $x \in \mathcal{C}$.
- For a valuation $\nu \in V_{\mathcal{C}}$ and a clock $x \in \mathcal{C}$ we define *reset x in ν* to be the valuation which equals ν except on x whose value is 0:

$$(\text{reset } x \text{ in } \nu)(y) = \begin{cases} \nu(y) & \text{if } y \neq x \\ 0 & \text{else} \end{cases}$$

Definition

- For a set \mathcal{C} of clocks, $\nu \in V_{\mathcal{C}}$, and $c \in \mathbb{N}$ we denote by $\nu + c$ the valuation with $(\nu + c)(x) = \nu(x) + c$ for all $x \in \mathcal{C}$.
- For a valuation $\nu \in V_{\mathcal{C}}$ and a clock $x \in \mathcal{C}$ we define *reset x in ν* to be the valuation which equals ν except on x whose value is 0:

$$(\text{reset } x \text{ in } \nu)(y) = \begin{cases} \nu(y) & \text{if } y \neq x \\ 0 & \text{else} \end{cases}$$

What does it mean?

- $\nu + 9$
- *reset x in $(\nu + 9)$*
- $(\text{reset } x \text{ in } \nu) + 9$
- *reset x in (reset y in ν)*

A **timed automaton** is a special hybrid system:

- **Variables are clocks** to model real-time behaviour and express real-time assumptions
- **Edges** are defined by
 - source and target locations,
 - a label,
 - a **guard**: clock constraint specifying enabling,
 - a set of clocks to be **reset**.
- **Invariants** are clock constraints.

Definition (Syntax of timed automata)

A *timed automaton* $\mathcal{T} = (Loc, Clocks, Lab, Edge, Inv, Init)$ is a tuple with

- Loc is a finite set of locations,
- $Clocks$ is a finite set of clocks,
- Lab is a finite set of synchronization labels,
- $Edge \subseteq Loc \times Lab \times (CC(Clocks) \times 2^{Clocks}) \times Loc$ is a finite set of edges,
- $Inv : Loc \rightarrow CC(Clocks)$ is a function assigning an invariant to each location, and
- $Init \subseteq \Sigma$ with $\nu(x) = 0$ for all $x \in Clocks$ and all $(l, \nu) \in Init$.

We call the variables in $Clocks$ *clocks*. We also use the notation $l \xrightarrow{a:g,C} l'$ to state that there exists an edge $(l, a, (g, C), l') \in Edge$.

Note: (1) no explicit activities given (2) fixed logic for constraints

Analogously to Kripke structures, we can additionally define

- a set of atomic propositions AP and
- a labeling function $L : Loc \rightarrow 2^{AP}$

to model further system properties.

$$\frac{\begin{array}{l} (l, a, (g, \mathcal{R}), l') \in Edge \\ \nu \models g \quad \nu' = \text{reset } \mathcal{R} \text{ in } \nu \quad \nu' \models \text{Inv}(l') \end{array}}{(l, \nu) \xrightarrow{a} (l', \nu')} \quad \text{Rule}_{\text{Discrete}}$$

$$\frac{t > 0 \quad \nu' = \nu + t \quad \nu' \models \text{Inv}(l)}{(l, \nu) \xrightarrow{t} (l, \nu')} \quad \text{Rule}_{\text{Time}}$$

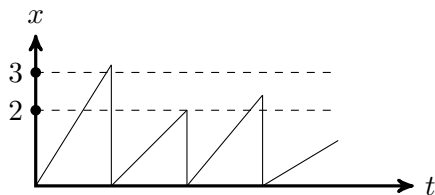
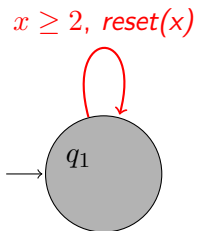
- execution step: $\rightarrow = \xrightarrow{a} \cup \xrightarrow{t}$
- path: $\sigma_0 \rightarrow \sigma_1 \rightarrow \sigma_2 \dots$
- run: path $\sigma_0 \rightarrow \sigma_1 \rightarrow \sigma_2 \dots$ with $\sigma_0 = (l_0, \nu_0)$, $l_0 \in \text{Init}$, $\nu_0(x) = 0$ f.a. $x \in \mathcal{C}$ (and $\nu_0 \in \text{Inv}(l_0)$)
- reachability of a state: exists a run leading to the state

Examples:

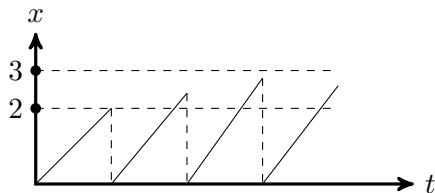
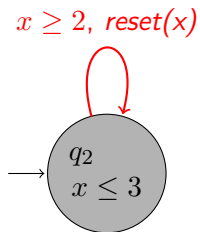
- Simple example: guards and invariants
- Light switch
- Controller from the railroad crossing example
- Simplified railroad crossing
- Parallel composition for the simplified railroad crossing

Discussion: Timed (hybrid) automata vs. transition systems

Example: Timed Automaton

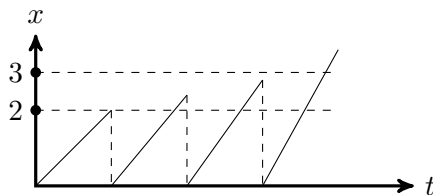
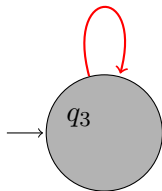


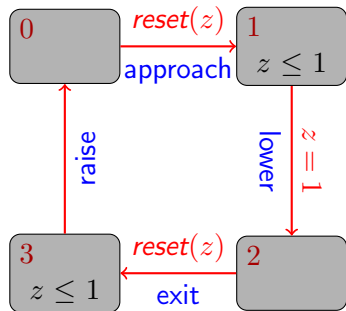
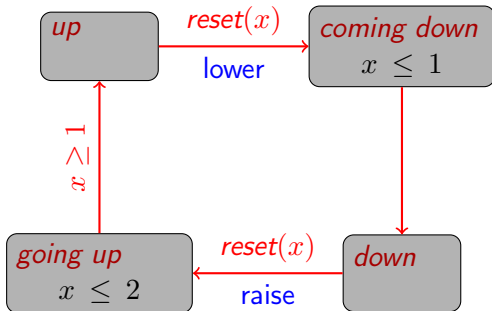
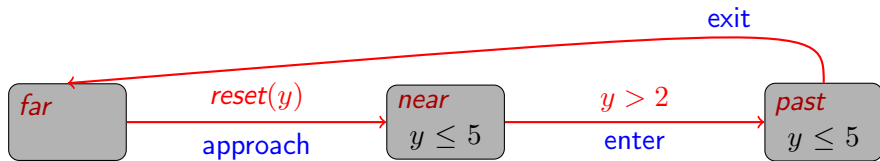
Example: Timed Automaton



Example: Timed Automaton

$2 \leq x \leq 3, \text{reset}(x)$





Time divergence, timelock, and zenoness

Zeno of Elea (c.490–c.430 BC)

Zeno's paradoxes: Achilles and the tortoise

"In a race, the quickest runner can never overtake the slowest, since the pursuer must first reach the point whence the pursued started, so that the slower must always hold a lead." —Aristotle, Physics VI:9, 239b15



- Not all paths of a timed automata represent realistic behaviour.
- Three essential phenomena: **time divergence**, **timelock**, **zenoness**.

Definition

For a timed automaton $\mathcal{T} = (Loc, \mathcal{C}, Lab, Edge, Inv, Init)$. we define *ExecTime* : $(Lab \cup \mathbb{R}^{\geq 0}) \rightarrow \mathbb{R}^{\geq 0}$ with

- $ExecTime(a) = 0$ for $a \in Lab$ and
- $ExecTime(d) = d$ for $d \in \mathbb{R}^{\geq 0}$.

Furthermore, for $\rho = s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} s_2 \xrightarrow{\alpha_2} \dots$ we define

$$ExecTime(\rho) = \sum_{i=0}^{\infty} ExecTime(\alpha_i).$$

A path is **time-divergent** iff $ExecTime(\rho) = \infty$, and **time-convergent** otherwise.

- Time-convergent paths are not realistic, and are not considered in the semantics.
- Note: their existence cannot be avoided (in general).

Definition

For a state $\sigma \in \Sigma$ let $Paths_{div}(\sigma)$ be the set of time-divergent paths starting in s .

A state $\sigma \in \Sigma$ contains a **timelock** iff $Paths_{div}(\sigma) = \emptyset$.

A timed automaton is **timelock-free** iff none of its *reachable* states contains a timelock.

Timelocks are modeling flows and should be avoided.

Definition

An infinite path fragment π is **zeno** iff it is time-convergent and infinitely many *discrete* actions are executed within π .

A timed automaton is **non-zeno** iff no zeno path starts in an *initial* state.

- Zeno paths represent **nonrealizable** behaviour, since their execution would require infinitely fast processors.
- Thus zeno paths are modeling flows and should be avoided.
- To **check** whether a timed automaton is non-zeno is algorithmically difficult.
- Instead, **sufficient** conditions are considered that are simple to check, e.g., by static analysis.

Checking non-zenoness

Theorem (Sufficient condition for non-zenoness)

Let \mathcal{T} be a timed automaton with clocks \mathcal{C} such that for every control cycle

$$l_0 \xrightarrow{\alpha_1:g_1,C_1} l_1 \xrightarrow{\alpha_2:g_2,C_2} l_2 \dots \xrightarrow{\alpha_n:g_n,C_n} l_n$$

in \mathcal{T} there exists a clock $x \in \mathcal{C}$ such that

- $x \in \mathcal{C}_i$ for some $0 < i \leq n$, and
- for all evaluations $\nu \in V$ there exists $d \in \mathbb{N}^{>0}$ with

$$\nu(x) < d \quad \text{implies} \quad (\nu \not\models g_j \text{ or } \nu \not\models \text{Inv}(l_j))$$

for some $0 < j \leq n$.

Then \mathcal{T} is non-zeno.

Note: the above condition is compositional

1 Motivation

2 Timed automata

3 TCTL

- How to describe the behaviour of timed automata?
- Logic: **TCTL**, a real-time variant of CTL
- **Syntax:**

State formulae

$$\psi ::= \text{true} \mid a \mid g \mid \psi \wedge \psi \mid \neg \psi \mid \exists \varphi \mid \forall \varphi$$

Path formulae:

$$\varphi ::= \psi \mathcal{U}^J \psi$$

with $J \subseteq \mathbb{R}^{\geq 0}$ is an interval with integer bounds (open right bound may be ∞).

- Syntactic sugar:

$$\mathcal{F}^J \psi \quad := \quad \text{true } \mathcal{U}^J \psi$$

$$\exists \mathcal{G}^J \psi \quad := \quad \neg \forall \mathcal{F}^J \neg \psi$$

$$\forall \mathcal{G}^J \psi \quad := \quad \neg \exists \mathcal{F}^J \neg \psi$$

$$\psi_1 \mathcal{U} \psi_2 \quad := \quad \psi_1 \mathcal{U}^{[0,\infty)} \psi_2$$

$$\mathcal{F} \psi \quad := \quad \mathcal{F}^{[0,\infty)} \psi$$

$$\mathcal{G} \psi \quad := \quad \mathcal{G}^{[0,\infty)} \psi$$

- Note: no next-time operator

Definition (TCTL semantics)

Let $\mathcal{T} = (Loc, \mathcal{C}, Lab, Edge, Inv, Init)$ be a timed automaton, AP a set of atomic propositions, and $L : Loc \rightarrow 2^{AP}$ a state labeling function. The function \models assigns a truth value to each TCTL state and path formulae as follows:

$$\begin{array}{ll} \sigma \models true & \\ \sigma \models a & \text{iff } a \in L(\sigma) \\ \sigma \models g & \text{iff } \sigma \models g \\ \sigma \models \neg\psi & \text{iff } \sigma \not\models \psi \\ \sigma \models \psi_1 \wedge \psi_2 & \text{iff } \sigma \models \psi_1 \text{ and } \sigma \models \psi_2 \\ \sigma \models \exists\varphi & \text{iff } \pi \models \varphi \text{ for some } \pi \in Paths_{div}(\sigma) \\ \sigma \models \forall\varphi & \text{iff } \pi \models \varphi \text{ for all } \pi \in Paths_{div}(\sigma). \end{array}$$

where $\sigma \in \Sigma$, $a \in AP$, $g \in ACC(\mathcal{C})$, ψ , ψ_1 and ψ_2 are TCTL state formulae, and φ is a TCTL path formula.

Meaning of \mathcal{U} : a time-divergent path satisfies $\psi_1 \mathcal{U}^J \psi_2$ whenever at some time point in J property ψ_2 holds and at all previous time instants $\psi_1 \vee \psi_2$ is satisfied.

Definition (TCTL semantics)

For a time-divergent path $\pi = \sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \dots$ we define $\pi \models \psi_1 \mathcal{U}^J \psi_2$ iff

- $\exists i \geq 0. \sigma_i + d \models \psi_1$ for some $d \in [0, d_i]$ with

$$\left(\sum_{k=0}^{i-1} d_k \right) + d \in J, \text{ and}$$

- $\forall j \leq i. \sigma_j + d' \models \psi_1 \vee \psi_2$ for any $d' \in [0, d_j]$ with

$$\left(\sum_{k=0}^{j-1} d_k \right) + d' \leq \left(\sum_{k=0}^{i-1} d_k \right) + d$$

where $d_i = ExecTime(\alpha_i)$.

Definition

For a timed automaton \mathcal{T} with clocks \mathcal{C} and locations Loc , and a TCTL state formula ψ the satisfaction set $Sat(\psi)$ is defined by

$$Sat(\psi) = \{s \in \Sigma \mid s \models \psi\}.$$

\mathcal{T} satisfies ψ iff ψ holds in all initial states:

$$\mathcal{T} \models \psi \text{ iff } \forall l_0 \in Init. (l_0, \nu_0) \models \psi$$

where $\nu_0(x) = 0$ for all $x \in \mathcal{C}$.

- TCTL formulae with intervals $[0, \infty)$ may be considered as CTL formulae
- However, there is a difference due to time convergent paths
- TCTL ranges over time-divergent paths, whereas CTL over all paths!