Timed Automata

Prof. Dr. Erika Ábrahám

Informatik 2 - Theory of Hybrid Systems
RWTH Aachen

SS09

Literature

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

Contents

Motivation

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 $\rightsquigarrow$ $k-1$ 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

$\bigcirc\varphi$	$(\mathcal{X}\varphi)$	:	φ holds after one tick
$\bigcirc^k\varphi$	$(\mathcal{X}^k\varphi)$	:	φ holds after k ticks
$\Diamond^{\leq k}\varphi$	$(\mathcal{F}^{\leq k}\varphi)$	:	φ occurs within k ticks

We deal in this lecture with continuous-time models.

Contents

Timed automata

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

Reading: Clock constraints

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

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

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

$ACC(\mathcal{C})$: 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

Semantics of clock constraints

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

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

Semantics of clock access

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:

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

What does it mean?

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

Timed automata

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.

Reminder: Hybrid automaton

Definition

A *hybrid automaton* $\mathcal{H}$ is a tuple $\mathcal{H} = (Loc, Var, Lab, Edge, Act, Inv, Init)$ with

- finite set of locations Loc,
- finite set of real-valued variables Var,
- finite set of synchronization labels Lab, $\tau \in Lab$ (stutter label)
- finite set of edges $Edge \subseteq Loc \times Lab \times 2^{V^2} \times Loc$ (including stutter transitions $(l, \tau, \mathtt{Id}, l)$ for each location $l \in Loc$),
- Act is a function assigning a set of activities $f : \mathbb{R}^+ \to V$ to each location; the activity sets are time-invariant, i.e., $f \in Act(l)$ implies $(f + t) \in Act(l)$, where $(f + t)(t') = f(t + t')$ f.a. $t' \in \mathbb{R}^+$,
- a function Inv assigning an invariant $Inv(l) \subseteq V$ to each location $l \in Loc$,
- initial states $Init \subseteq \Sigma$.

with

- valuations $\nu : Var \to \mathbb{R}$, V is the set of valuations
- state: $(l, \nu) \in Loc \times V$, Σ is the set of states
- transitions: discrete and time

Timed automaton

Definition

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

- finite set of locations Loc,
- finite set of clocks $\mathcal{C}$,
- finite set of synchronization labels Lab, $\tau \in Lab$ (stutter label)
- finite set of edges $Edge \subseteq Loc \times Lab \times (CC(\mathcal{C}) \times 2^{\mathcal{C}}) \times Loc$ (including stutter transitions $(l, \tau, \mathit{true}, \mathtt{Id}, l)$ for each location $l \in Loc$),
- a function Inv assigning an invariant $Inv(l) \in CC(\mathcal{C})$ to each location $l \in Loc$,
- initial locations $Init \subseteq Loc$ (with $\nu \models Inv(l)$ for all $l \in Init$ and $\nu(x) = 0$ f.a. $x \in \mathcal{C}$).

with

- valuations $\nu : \mathcal{C} \to \mathbb{R}^{\geq 0}$, V is the set of valuations
- state: $(l, \nu) \in Loc \times V$, Σ is the set of states
- transitions: discrete and time

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

Timed automaton

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.

Operational semantics

$$\frac{(l, a, (g, \mathcal{R}), l') \in Edge \qquad \nu \models g \quad \nu' = \textit{reset } \mathcal{R} \textit{ in } \nu \quad \nu' \models Inv(l')}{(l, \nu) \xrightarrow{a} (l', \nu')} \quad \texttt{Rule}_{\texttt{Discrete}}$$

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

- execution step: $\rightarrow = \xrightarrow{a} \cup \xrightarrow{t}$
- path: $\sigma_0 \rightarrow \sigma_1 \rightarrow \sigma_2 \ldots$
- run: path $\sigma_0 \rightarrow \sigma_1 \rightarrow \sigma_2 \ldots$ with $\sigma_0 = (l_0, \nu_0)$, $l_0 \in Init$, $\nu_0(x) = 0$ f.a. $x \in \mathcal{C}$ (and $\nu_0 \in Inv(l_0)$)
- 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

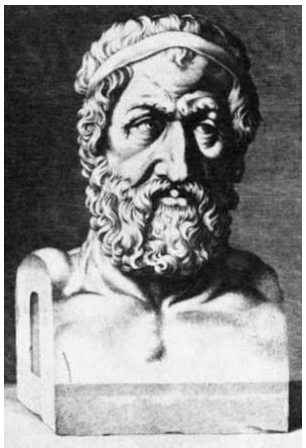
Discussion: Timed (hybrid) automata vs. transition systems

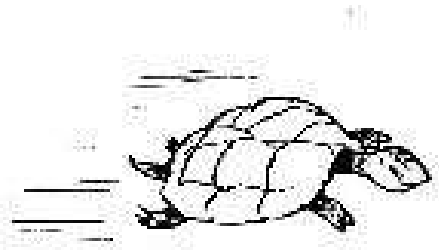
Time divergence, timelock, and zenoness

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 pathes of a timed automata represent realistic behaviour.
- Three essential phenomena: time divergence, timelock, zenoness.

Time divergence

Definition

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

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

Furthermore, for $\rho = s_0 \xrightarrow{\tau_0} s_1 \xrightarrow{\tau_1} s_2 \xrightarrow{\tau_2} \ldots$ we define

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

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

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

Timelock

Definition

For a state $s \in \Sigma$ let $Paths_{div}(s)$ be the set of time-divergent paths starting in s.
A state $s \in \Sigma$ contains a timelock iff $Paths_{div}(s) = \emptyset$.
A timed automaton is timelock-free iff none of its *reachable* states contains a timelock.

Timelocks are modeling flows and should be avoided.

Zenoness

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 pathes 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 $\mathcal{TA}$ be a timed automaton with clocks $\mathcal{C}$ such that for every control cycle

$$l_0 \xrightarrow{g_1:\alpha_1,R_1} l_1 \xrightarrow{g_2:\alpha_2,R_2} l_2 \ldots \xrightarrow{g_n:\alpha_n,R_n} l_n$$

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

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

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

for some $0 < j \leq n$.

Then $\mathcal{TA}$ is non-zeno.

Note: the above condition is compositional

Uniqueness of paths

- There are equivalent paths in the sense of reachability, since two subsequent time transisitons can be represented by a single time transition, e.g.:

$$\sigma_0 \xrightarrow{d_1} \sigma_1 \xrightarrow{d_2} \sigma_2 \quad \textit{and} \quad \sigma_0 \xrightarrow{d_1+d_2} \sigma_1$$

  can be considered equivalent.

- Equivalence classes:

$$\sigma_0 \overset{d_1,a_1}{\Longrightarrow} \sigma_1 \overset{d_2,a_2}{\Longrightarrow} \sigma_2 \overset{d_3,a_3}{\Longrightarrow} \dots$$

Contents

TCTL

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

  State formulae

$$\Phi \quad ::= \quad \mathit{true} \;\mid\; a \;\mid\; g \;\mid\; \Phi \wedge \Phi \;\mid\; \neg\Phi \;\mid\; \exists\varphi \;\mid\; \forall\varphi$$

  Path formulae:

$$\varphi \quad ::= \quad \Phi \, \mathcal{U}^{J} \, \Phi$$

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

TCTL syntax

- Syntactic sugar:

$$
\begin{aligned}
\Diamond^J \Phi &:= \mathit{true}\, \mathcal{U}^J\ \Phi \\
\exists \Box^J \Phi &:= \neg \forall \Diamond^J \neg \Phi \\
\forall \Box^J \Phi &:= \neg \exists \Diamond^J \neg \Phi \\
\Phi\ \mathcal{U}\ \Psi &:= \Phi\ \mathcal{U}^{[0,\infty)}\ \Psi \\
\Diamond \Phi &:= \Diamond^{[0,\infty)} \Phi \\
\Box \Phi &:= \Box^{[0,\infty)} \Phi
\end{aligned}
$$

- Note: no next-time operator

TCTL semantics

Definition (TCTL semantics)

Let $\mathcal{TA} = (Loc, \mathcal{C}, Lab, Edge, Inv, Init, AP, L_{AP})$ be a timed automaton, $a \in AP$, $g \in ACC(\mathcal{C})$, and $J \subseteq \mathbb{R}^{\geq 0}$. For state $s = (l, \nu) \in \Sigma$, TCTL state formulae Φ and Ψ, and path formula φ, we define $\models$ by

$$
\begin{array}{lll}
s \models \mathit{true} & & \\
s \models a & \mathit{iff} & a \in L_{AP}(l) \\
s \models g & \mathit{iff} & \nu \models g \\
s \models \neg\Phi & \mathit{iff} & \mathit{not}\ s \models \Phi \\
s \models \Phi \wedge \Psi & \mathit{iff} & s \models \Phi\ \mathit{and}\ s \models \Psi \\
s \models \exists\varphi & \mathit{iff} & \pi \models \varphi\ \mathit{for\ some}\ \pi \in Paths_{\mathit{div}}(s) \\
s \models \forall\varphi & \mathit{iff} & \pi \models \varphi\ \mathit{for\ all}\ \pi \in Paths_{\mathit{div}}(s).
\end{array}
$$

TCTL semantics

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

TCTL semantics (cont.)

Definition (TCTL semantics)

For a time-divergent path $\pi \in \sigma_0 \overset{d_1,a_1}{\Longrightarrow} \sigma_1 \overset{d_2,a_2}{\Longrightarrow} \ldots$ we define $s \models \Phi \; \mathcal{U}^J \; \Psi$ by

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

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

- $\forall j \leq i.\ s_j + d' \models \Phi \vee \Psi$ for any $d' \in [0, d_j]$ with

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

Satisfaction set

Definition

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

$$Sat(\Phi) = \{s \in \Sigma | s \models \Phi\}.$$

$\mathcal{TA}$ satisfies Φ iff Φ holds in all initial states:

$$\mathcal{TA} \models \Phi \textit{ iff } \forall l_0 \in Init.\ (l_0, \nu_0) \models \Phi$$

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

TCTL vs. CTL

- 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!