Modeling and Analysis of Hybrid Systems

Timed automata

Prof. Dr. Erika Ábrahám

Informatik 2 - Theory of Hybrid Systems
RWTH Aachen University

SS 2013

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.

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.

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

Discrete-time systems

Advantages:

- 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

Discrete-time systems

Advantages:

- 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

Disadvantages:

- leads to large transition systems
- minimal time between two actions is a multiple of the tick

Discrete-time systems

Advantages:

- 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

Disadvantages:

- leads to large transition systems
- minimal time between two actions is a multiple of the tick

Logic: CTL or LTL extended with syntactic sugar

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

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

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

Discrete-time systems

Advantages:

- 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

Disadvantages:

- leads to large transition systems
- minimal time between two actions is a multiple of the tick

Logic: CTL or LTL extended with syntactic sugar

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

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

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

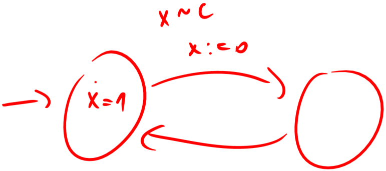
We deal in this lecture with continuous-time models.

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

$\sim \in \{\leq, <, =, >, \geq\}$



LTS

C

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

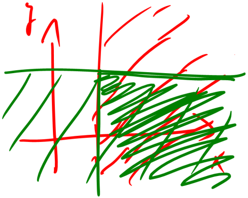
$$g \quad ::= \quad x < c \quad | \quad x \leq c \quad | \quad x > c \quad | \quad x \geq c \quad | \quad g \wedge g$$

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

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

$ACC(\mathcal{C})$: 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

Given a set $\mathcal{C}$ of clocks, a clock valuation $\nu : \mathcal{C} \to \mathbb{R}_{\geq 0}$ assigns a non-negative value to each clock. We use $V_{\mathcal{C}}$ to denote the set of clock valuations for the clock set $\mathcal{C}$.

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:

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

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:

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

What does it mean?

- $(\nu + 9)(x) = \nu(x) + 9$

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:

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

What does it mean?

- $\nu + 9$
- $($*reset* x *in* $(\nu + 9))(x) = 0$

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:

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

What does it mean?

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

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:

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

What does it mean?

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

$\nu(x)$		$\nu(x) = 0$
0	9	$\nu \not\models x \geq 2$
0	0	$\nu + 2 \models x \geq 2$
0	9	$\nu + 3 \models x \geq 2$
0	0	reset x in $\nu \not\models x \geq 2$

Timed automata

A timed automaton is a special hybrid automaton:

- All variables are clocks.
- 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.

Timed automaton

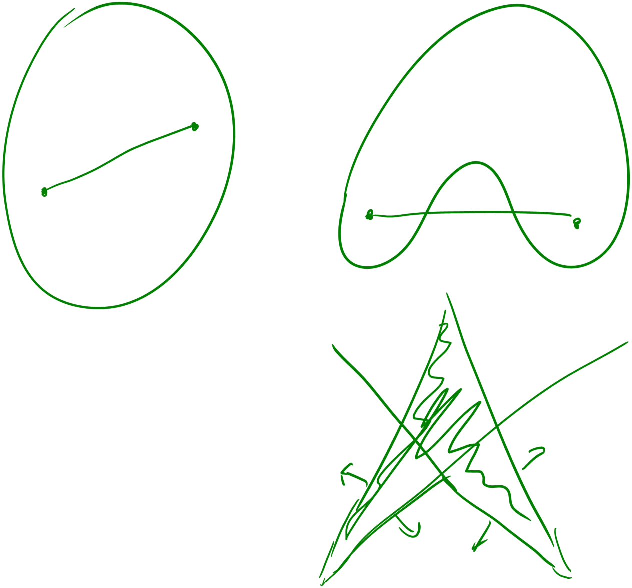
Definition (Syntax of timed automata)

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

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

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

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



Timed automaton

Analogously to Kripke structures, we can additionally define

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

to model further system properties.

Operational semantics

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

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

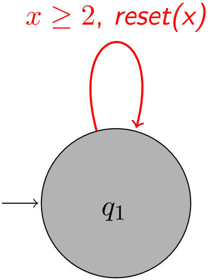
Operational semantics

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

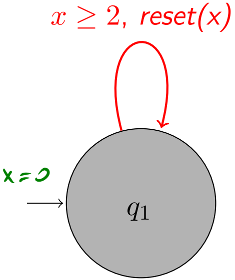
$$\frac{t>0 \quad \nu' = \nu + t \quad \nu' \models Inv(l)}{(l, \nu) \xrightarrow{t} (l, \nu')} \quad \texttt{Rule}_{\texttt{ Time}}$$

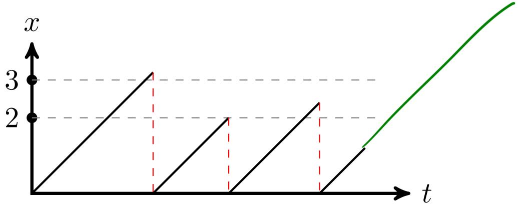
- Execution step: $\rightarrow \; = \; \xrightarrow{a} \cup \xrightarrow{t}$
- Path: $\sigma_0 \rightarrow \sigma_1 \rightarrow \sigma_2 \ldots$ with $\sigma_0 = (l_0, \nu_0)$ and $\nu_0 \in Inv(l_0)$
- Initial path: path $\sigma_0 \rightarrow \sigma_1 \rightarrow \sigma_2 \ldots$ with $\sigma_0 = (l_0, \nu_0)$, $l_0 \in Init$ and $\nu_0(x) = 0$ f.a. $x \in \mathcal{C}$
- Reachability of a state: exists an initial path leading to the state

Example: Timed Automaton

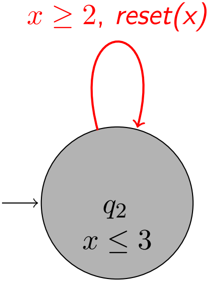


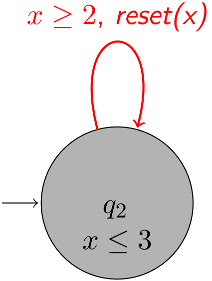
Example: Timed Automaton

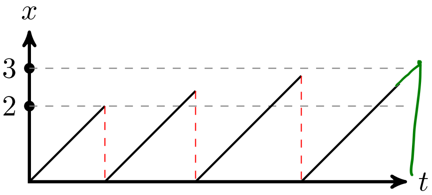




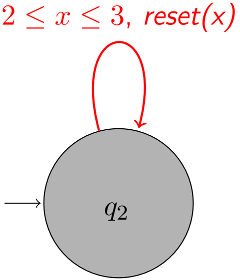
Example: Timed Automaton

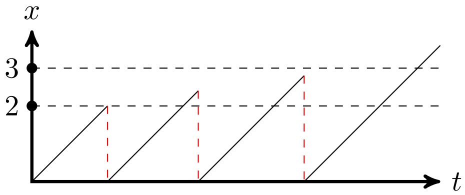



$x \geq 2$, reset(x)
q_2
$x \leq 3$

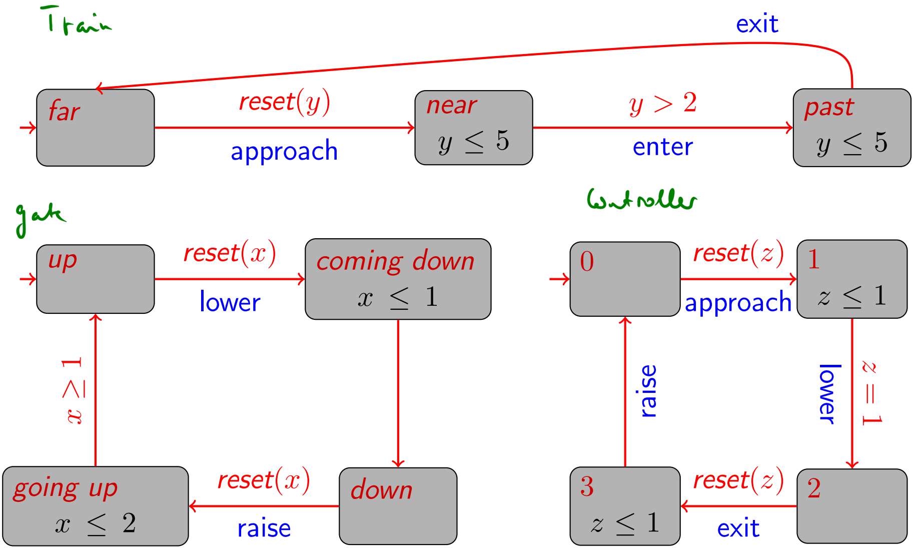

x
3
2
t

Example: Timed Automaton

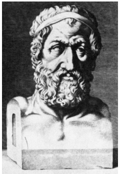




Example: Railroad Crossing

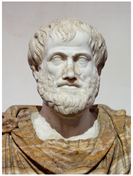


Time divergence, timelock, and zenoness



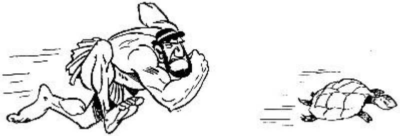
Zeno of Elea

(ca.490 BC-ca.430 BC)



Aristotle

(384 BC-322 BC)

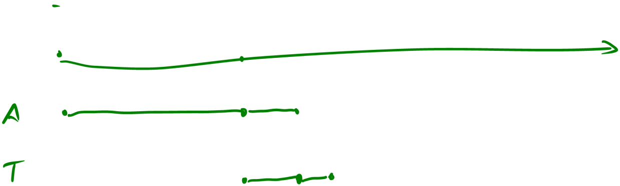


Paradox:
Achilles and the tortoise

(Achilles was the great Greek hero of Homer's The Iliad.)

"In a race, the quickest runner can never overtake the slowest, since the pursuer must first reach the point where 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 convergence, timelock, zenoness.


A
T

Time convergence

Definition

For a timed automaton $\mathcal{T} = (Loc, \mathcal{C}, Lab, Edge, Inv, Init)$. 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 $\rho = s_0 \stackrel{\alpha_0}{\to} s_1 \stackrel{\alpha_1}{\to} s_2 \stackrel{\alpha_2}{\to} \ldots$ we define

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

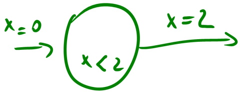
A path is time-divergent iff $ExecTime(\rho) = \infty$, and time-convergent otherwise.

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

$\pi : \bullet \xrightarrow{1} \bullet \xrightarrow{1/2} \xrightarrow{1/4} \xrightarrow{1/8} \ldots$

$$\text{ExecTime}(\pi) = \sum_{i=1}^{\infty} \frac{1}{i} = 2$$

Timelock



Definition

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

Zenoness

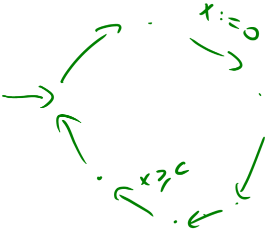
Definition

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 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.

$$x = 0 \xrightarrow{a} x = 0 \xrightarrow{a} x = 0 \xrightarrow{a}$$

$$x = 0 \xrightarrow{1} x = 1 \xrightarrow{a} x = 0 \xrightarrow{1/2} x = \tfrac{1}{2} \xrightarrow{a} x = 0 \xrightarrow{1/4} x = \tfrac{1}{4} \xrightarrow{a} \ldots$$



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

$$l_0 \stackrel{\alpha_1:g_1,C_1}{\longrightarrow} l_1 \stackrel{\alpha_2:g_2,C_2}{\longrightarrow} l_2 \ldots \stackrel{\alpha_n:g_n,C_n}{\longrightarrow} l_n = l_0$$

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

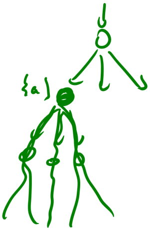
- $x \in C_i$ *for some* $0 < i \leq n$*, and*
- *for all evaluations* $\nu \in V$ *there exist some* $0 < j \leq n$ *and* $d \in \mathbb{N}^{>0}$ *with*

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

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

Contents

TCTL



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

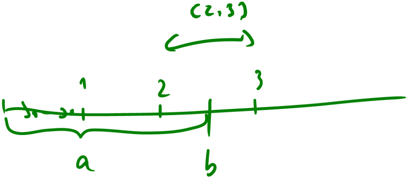
State formulae

$$\psi \quad ::= \quad \textit{true} \quad | \quad a \quad | \quad g \quad | \quad \psi \wedge \psi \quad | \quad \neg\psi \quad | \quad \mathbf{E}\varphi \quad | \quad \mathbf{A}\varphi$$

Path formulae:

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

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



$a \; \mathcal{U}^{[2,3]} \; b$

request $\mathcal{U}$ response

- Syntactic sugar:

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

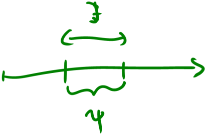
$$\mathbf{E}\mathcal{G}^J \psi \quad := \quad \neg \mathbf{A}\mathcal{F}^J \neg \psi$$

$$\mathbf{A}\mathcal{G}^J \psi \quad := \quad \neg \mathbf{E}\mathcal{F}^J \neg \psi$$

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

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

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



- Note: no next-time operator

TCTL semantics

Definition (TCTL continuous semantics)

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

$$
\begin{array}{lll}
\sigma & \models \mathit{true} & \\
\sigma & \models a & \textit{iff} \quad a \in L(\sigma) \\
\sigma & \models g & \textit{iff} \quad \sigma \models g \\
\sigma & \models \neg\psi & \textit{iff} \quad \sigma \not\models \psi \\
\sigma & \models \psi_1 \wedge \psi_2 & \textit{iff} \quad \sigma \models \psi_1 \textit{ and } \sigma \models \psi_2 \\
 & & \\
\sigma & \models \mathbf{E}\varphi & \textit{iff} \quad \pi \models \varphi \textit{ for some } \pi \in Paths_{div}(\sigma) \\
\sigma & \models \mathbf{A}\varphi & \textit{iff} \quad \pi \models \varphi \textit{ for all } \pi \in Paths_{div}(\sigma).
\end{array}
$$

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.

TCTL semantics

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.

TCTL semantics

Definition (TCTL continuous semantics)

For a time-divergent path $\pi = \sigma_0 \stackrel{\alpha_1}{\rightarrow} \sigma_1 \stackrel{\alpha_2}{\rightarrow} \ldots$ 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
$$(\sum_{k=0}^{i-1} d_k) + d \in J, \ \textit{and}$$
- $\forall j \leq i.\ \sigma_j + d' \models \psi_1 \vee \psi_2$ for any $d' \in [0, d_j]$ with
$$(\sum_{k=0}^{j-1} d_k) + d' \leq (\sum_{k=0}^{i-1} d_k) + d$$

where $d_i = ExecTime(\alpha_i)$.

Satisfaction set

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

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

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

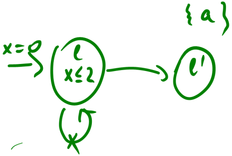
$$\mathcal{T} \models \psi \textit{ iff } \forall l_0 \in Init.\ (l_0, \nu_0) \models \psi$$

$\forall x \in C .\ \nu_0(x) = 0$

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

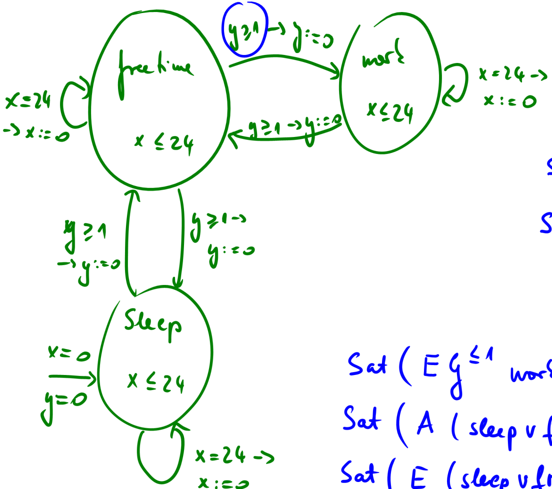
TCTL vs. CTL

- 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!



$AF\,a$

$$\rightarrow (\ell, x=0) \xrightarrow{1} (\ell, x=1) \xrightarrow{1/2} (\ell, x=1{,}5) \xrightarrow{1/4} (\ell, x=1{,}75) \rightarrow \ldots$$



$Sat(AF\, work) = \{work\} \times V$

$Sat(EF\, work) = Loc \times V = \Sigma$

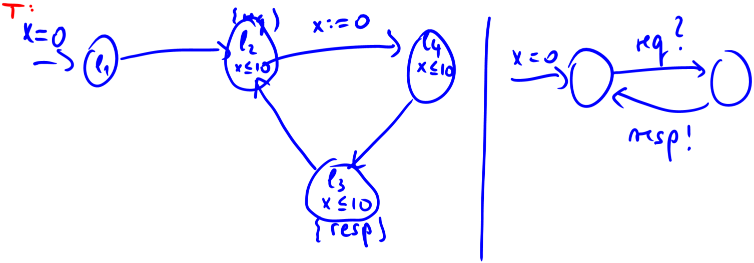
$Sat(EF^{\leq 2}\, work) = \Sigma$

$Sat(EF^{\leq 1}\, work) = \{(\ell, \nu) \in \Sigma \mid \ell = work \vee \ell = freetime \vee (\ell = sleep \wedge y \geq 1)\}$

$Sat(EG^{\leq 1}\, work) = \{(\ell, \nu) \in \Sigma \mid \ell = work \wedge y = 0\}$

$Sat(A\, (sleep \vee freetime)\, \mathcal{U}\, work) = \{work\} \times V$

$Sat(E\, (sleep \vee freetime)\, \mathcal{U}^{[1,2]}\, work) = \Sigma$



$Sat(AG\,AF\,req) = \{\ell_2, \ell_3, \ell_4\} \times V$

$(\ell_1, x=0) \xrightarrow{0} (\ell_2, x=0)$

$\downarrow 10$

$(\ell_4, x=0) \xleftarrow{0} (\ell_2, x=10)$

$\downarrow 10$

$(\ell_4, x=10) \xrightarrow{0} (\ell_3, x=10)$

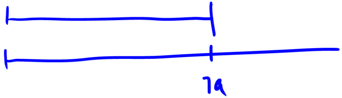
$Sat(AF\,req) =$

$\{\ell_2, \ell_3, \ell_4\} \times V$

$Sat(AG(req \rightarrow AF^{\leq 20} resp))$

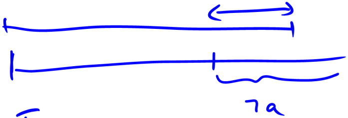
$AG((req \wedge x \geq 10) \rightarrow AF^{\leq 10} resp)$

TF – 11 –



$G\,a$ — safety

violation: finite path



$G F a$ — liveness

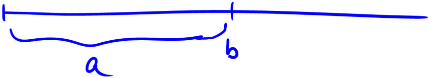
violation: infinite path

$F a$ — violation:

$F G a$ — liveness

violation:

$a\, U\, b$



violation: i) $G \neg b$ — safety

ii)

	safety	liveness
witness	infinite	finite
⇒ counter-example	finite	infinite

$$a \, \mathcal{U} \, b = a \, \mathcal{U}_{weak} \, b \wedge F b$$

$a \, \mathcal{U}_{weak} \, b$ ↑ safety

$F b$ ↑ liveness