Modeling and Analysis of Hybrid Systems

Timed automata

Prof. Dr. Erika Ábrahám

Informatik 2 - LuFG Theory of Hybrid Systems
RWTH Aachen University

Szeged, Hungary, 27 September - 06 October 2017

Literature

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

Contents

1 Motivation

2 Timed automata

3 Timed computation tree logic (TCTL)

4 TCTL model checking for timed automata

Motivation

Time-critical systems

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

Time-critical systems

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

Time-critical systems

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 modelling: 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 $\leadsto$ $k - 1$ 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

1 Motivation

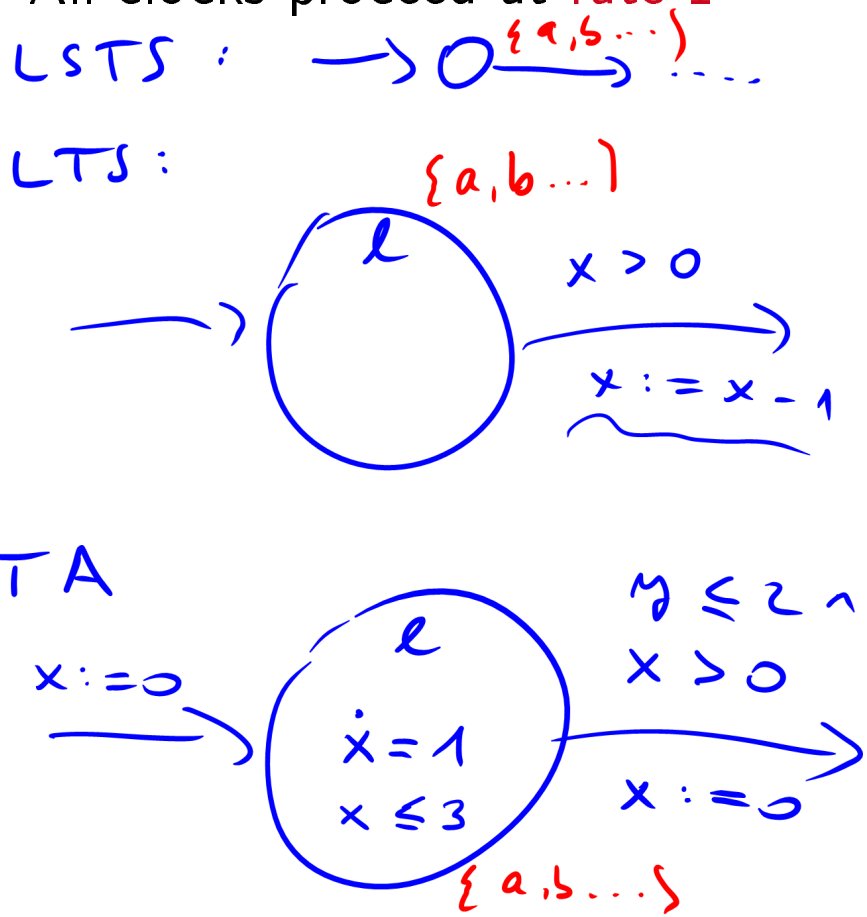
2 Timed automata

3 Timed computation tree logic (TCTL)

4 TCTL model checking for timed automata

Timed automata

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



Timed automata

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

  Read access:

  Atomic clock constraints:

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

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

  Clock constraints:

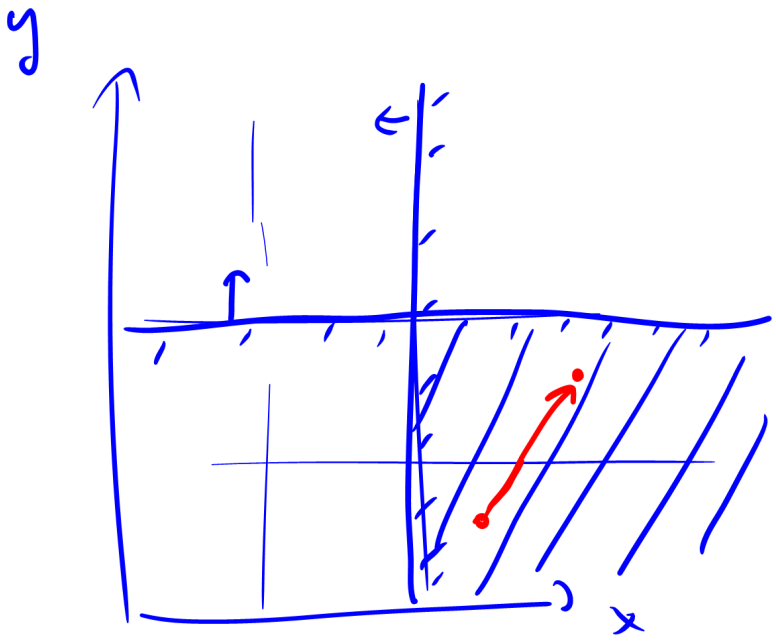
  $$g \quad ::= \quad acc \quad | \quad g \wedge g$$

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

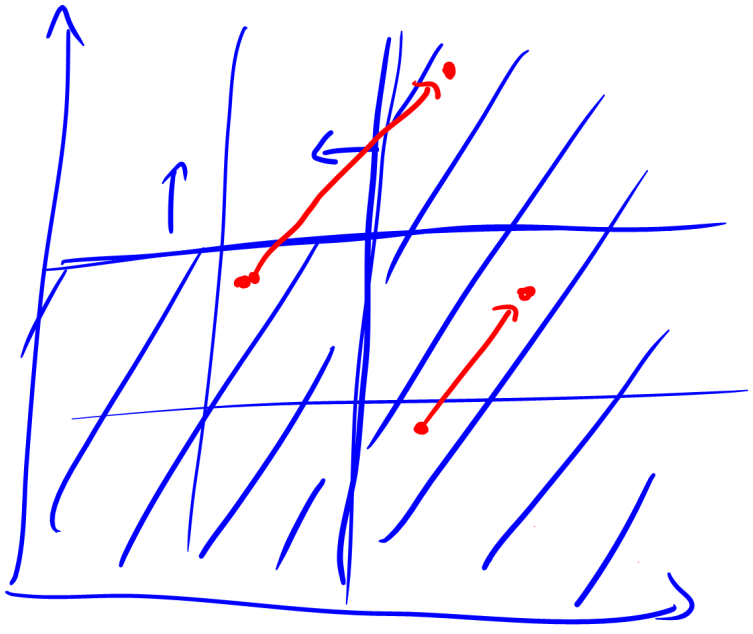
  $ACC(\mathcal{C})$: set of atomic clock constraints over $\mathcal{C}$
  $CC(\mathcal{C})$: set of clock constraints over $\mathcal{C}$

  Write access: Clock reset sets clock value to 0

$x \geq 2 \wedge y \leq 2$



$x \geq 2 \vee y \leq 2$



Semantics of clock constraints

Given a set $\mathcal{C}$ of clocks, a clock valuation

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 (Semantics of clock constraints)

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 (Semantics of clock constraints)

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 (Time delay, clock reset)

Semantics of clock access

Definition (Time delay, clock reset)

- 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

Semantics of clock access

Definition (Time delay, clock reset)

- 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 $x \in \mathcal{C}$.

Semantics of clock access

Definition (Time delay, clock reset)

- 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 set $R \subseteq \mathcal{C}$ we define *reset R in ν* to be

Semantics of clock access

Definition (Time delay, clock reset)

- 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 set $R \subseteq \mathcal{C}$ we define *reset R in ν* to be the valuation resulting from ν by resetting all clocks from R:

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

For a single clock $x \in \mathcal{C}$ we write *reset x in ν*.

Semantics of clock access

Definition (Time delay, clock reset)

- 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 set $R \subseteq \mathcal{C}$ we define *reset* R *in* ν to be the valuation resulting from ν by resetting all clocks from R:

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

For a single clock $x \in \mathcal{C}$ we write *reset* x *in* ν.

valuation for $\mathcal{C} = \{x, y\}$	value of x	value of y
ν	5	1
$\nu + 9$ *reset* x *in* $(\nu + 9)$ $(\textit{reset } x \textit{ in } \nu) + 9$ *reset* $\{x, y\}$ *in* ν		

Semantics of clock access

Definition (Time delay, clock reset)

- 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 set $R \subseteq \mathcal{C}$ we define *reset* R *in* ν to be the valuation resulting from ν by resetting all clocks from R:

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

For a single clock $x \in \mathcal{C}$ we write *reset* x *in* ν.

valuation for $\mathcal{C} = \{x, y\}$	value of x	value of y
ν	5	1
$\nu + 9$ *reset* x *in* $(\nu + 9)$ $(\textit{reset } x \textit{ in } \nu) + 9$ *reset* $\{x, y\}$ *in* ν	14	10

Semantics of clock access

Definition (Time delay, clock reset)

- 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 set $R \subseteq \mathcal{C}$ we define *reset R in ν* to be the valuation resulting from ν by resetting all clocks from R:

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

For a single clock $x \in \mathcal{C}$ we write *reset x in ν*.

valuation for $\mathcal{C} = \{x, y\}$	value of x	value of y
ν	5	1
$\nu + 9$	14	10
reset x *in* $(\nu + 9)$	0	10
$(\textit{reset } x \textit{ in } \nu) + 9$		
reset $\{x, y\}$ *in* ν		

Semantics of clock access

Definition (Time delay, clock reset)

- 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 set $R \subseteq \mathcal{C}$ we define *reset* R *in* ν to be the valuation resulting from ν by resetting all clocks from R:

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

For a single clock $x \in \mathcal{C}$ we write *reset* x *in* ν.

valuation for $\mathcal{C} = \{x, y\}$	value of x	value of y
ν	5	1
$\nu + 9$	14	10
reset x *in* $(\nu + 9)$	0	10
$(\textit{reset } x \textit{ in } \nu) + 9$	9	10
reset $\{x, y\}$ *in* ν		

Semantics of clock access

Definition (Time delay, clock reset)

- 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 set $R \subseteq \mathcal{C}$ we define *reset* R *in* ν to be the valuation resulting from ν by resetting all clocks from R:

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

For a single clock $x \in \mathcal{C}$ we write *reset* x *in* ν.

valuation for $\mathcal{C} = \{x, y\}$	value of x	value of y
ν	5	1
$\nu + 9$	14	10
reset x *in* $(\nu + 9)$	0	10
$(\textit{reset } x \textit{ in } \nu) + 9$	9	10
reset $\{x, y\}$ *in* ν	0	0

Timed automata

Reminder: LTS = (Loc, Var, Lab, Edge, Init)

Timed automata extend labeled transition systems with the notion of time:

- All variables are clocks $c \in \mathcal{C}$.
- States $\sigma \in \Sigma = Loc \times V$ are pairs of a location and a clock valuation.
- While the control stays in a location, time passes by. Time can pass in a location as long as its invariant, which is a clock constraint, is satisfied. During control stays in a location, the clock values evolve with derivative 1.
- Edges are defined by
  - source and target locations,
  - a label,
  - a guard: clock constraint specifying enabling,
  - a set of clocks to be reset to zero.

  Note: guard g and reset R define a transition relation $\mu = \{(\nu, \nu') \in V^2 \mid \nu \models g \wedge \nu' = \mathit{reset}\ R\ \mathit{in}\ \nu\}$. $\subseteq V \times V$

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 synchronisation labels,
- $Edge \subseteq Loc \times Lab \times (\mathcal{C} \times 2^{\mathcal{C}}) \times Loc$ is a finite set of edges,
- $Inv : Loc \to \mathcal{C}$ is a function assigning an invariant to each location,
- $Init \subseteq \Sigma$ 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 $l \overset{a:g,R}{\hookrightarrow} l'$ to state that there exists an edge $(l, a, (g, R), l') \in Edge$.

Note: (1) derivaties are not explicitly specified (2) restricted logic for constraints

Timed automaton

Analogously to Kripke structures, we can additionally define

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

to model further system properties.

Operational semantics

$$\frac{(\ell, a, (g, R), \ell') \in Edge \quad \nu \models g \quad \nu' = reset\ R\ in\ \nu \quad \nu' \models Inv(\ell')}{(\ell, \nu) \xrightarrow{a} (\ell', \nu')} \text{Rule}_d$$

$$\frac{\nu' = \nu + t \quad \nu' \models Inv(\ell) \quad t \geq 0 \quad t \in \mathbb{R}}{(\ell, \nu) \xrightarrow{t} (\ell, \nu')} \text{Rule}_{time}$$

Operational semantics

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

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

Operational semantics

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

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

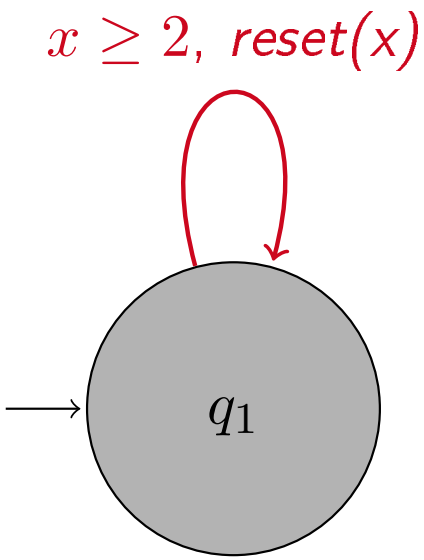
- Execution step: $\rightarrow = \xrightarrow{a} \cup \xrightarrow{t}$
- Path: $\sigma_0 \rightarrow \sigma_1 \rightarrow \sigma_2 \ldots$ with $\sigma_0 = (l_0, \nu_0)$ and $\nu_0 \in Inv(l_0)$
- Initial path: path $\sigma_0 \rightarrow \sigma_1 \rightarrow \sigma_2 \ldots$ with $\sigma_0 = (l_0, \nu_0)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

LSTS semantics of timed automata

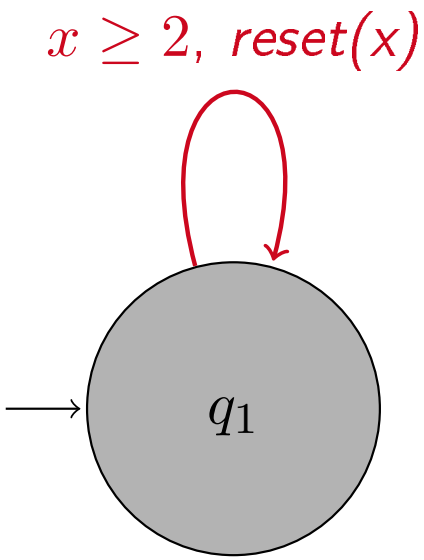
Each timed automaton $\mathcal{T} = (Loc, \mathcal{C}, Lab, Edge, Inv, Init)$ induces a labeled state transition system $\mathcal{LSTS} = (\Sigma, Lab', Edge', Init)$ with

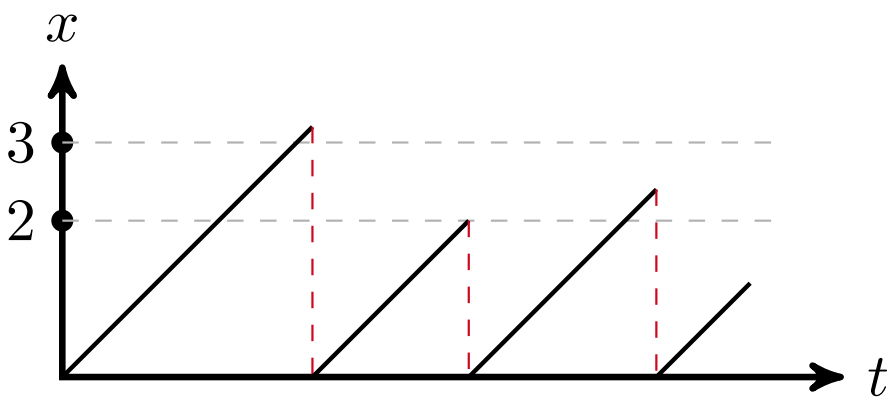
- $\Sigma = Loc \times V$,
- $Lab' = Lab \cup \mathbb{R}_{\geq 0}$
- $Edge' = \{(\nu, \alpha, \nu') \mid \nu \xrightarrow{\alpha} \nu'\}$. $\{ (\sigma, \alpha, \sigma') \mid \sigma \xrightarrow{\alpha} \sigma' \}$

Example: Timed Automaton

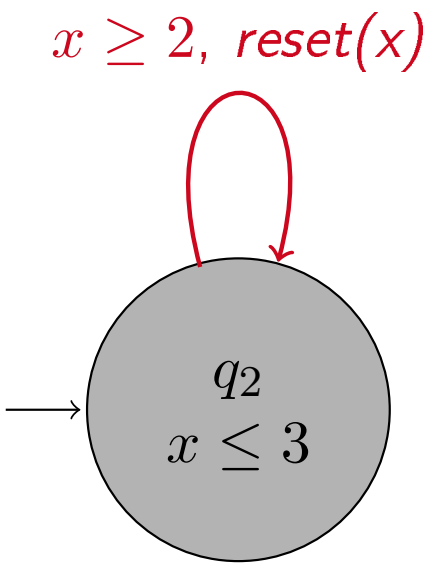


Example: Timed Automaton

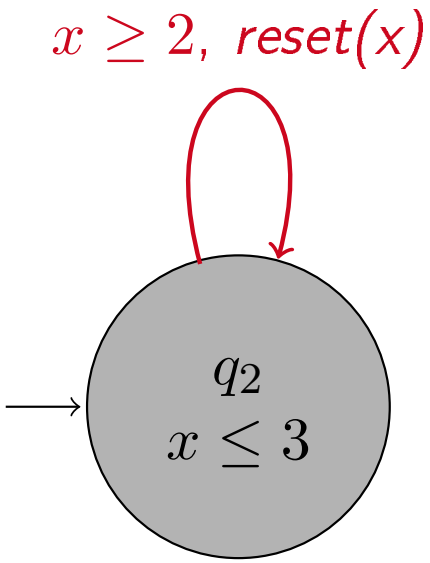


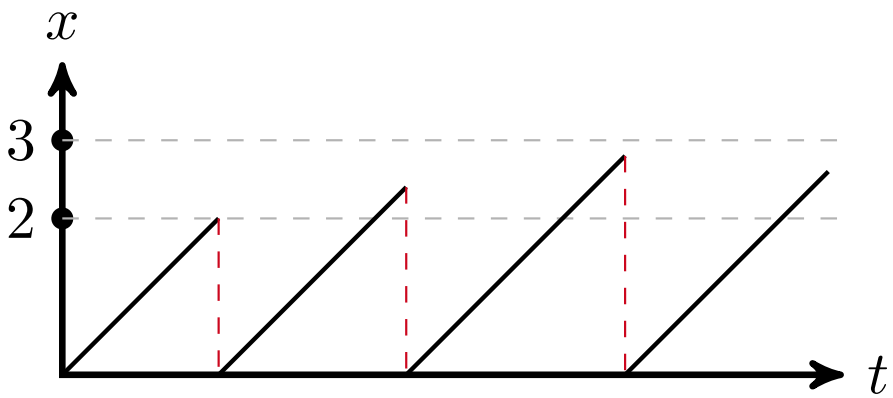


Example: Timed Automaton

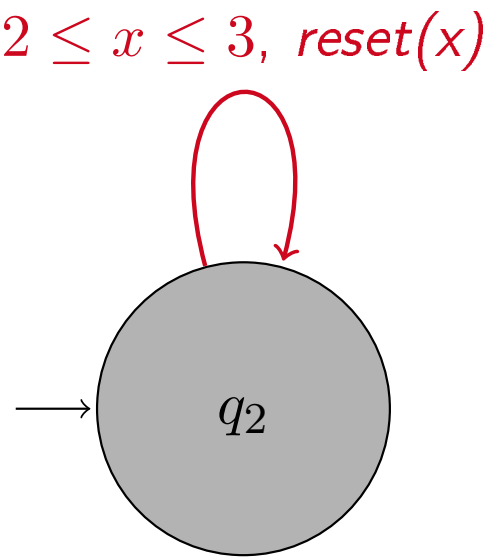


Example: Timed Automaton

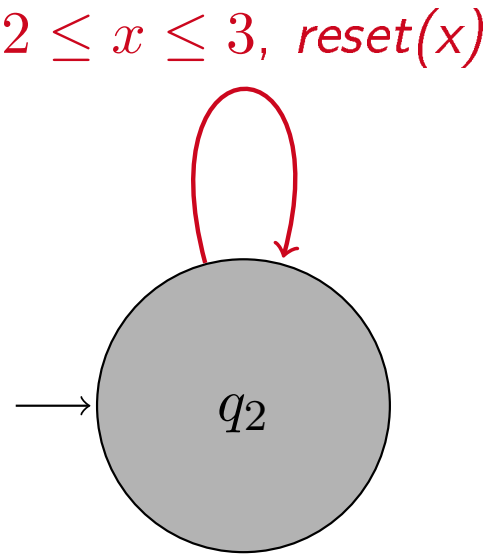


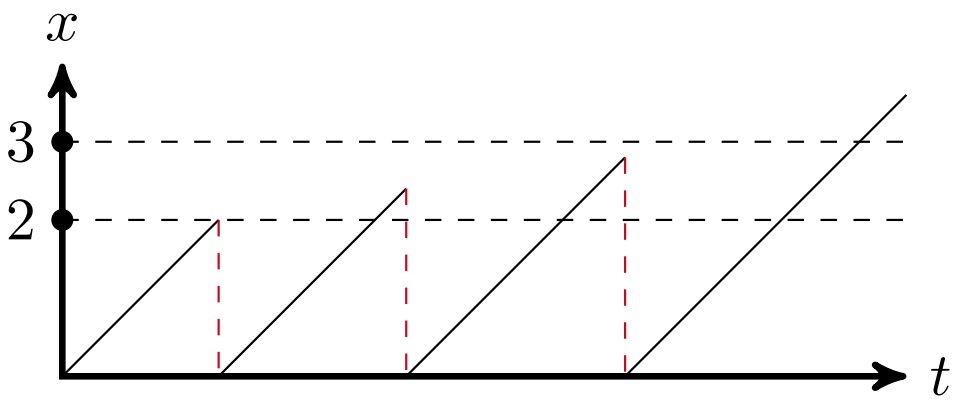


Example: Timed Automaton

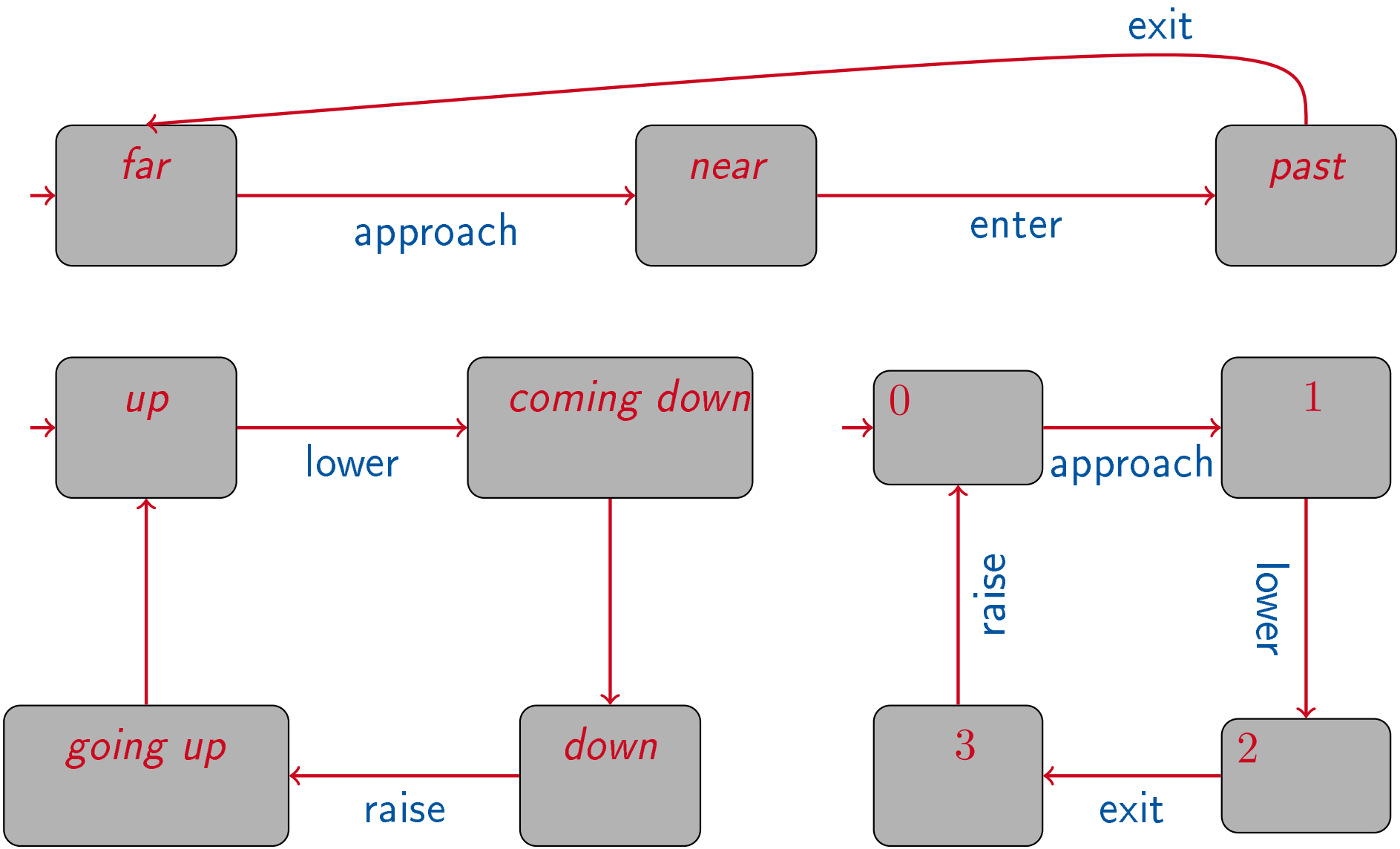


Example: Timed Automaton

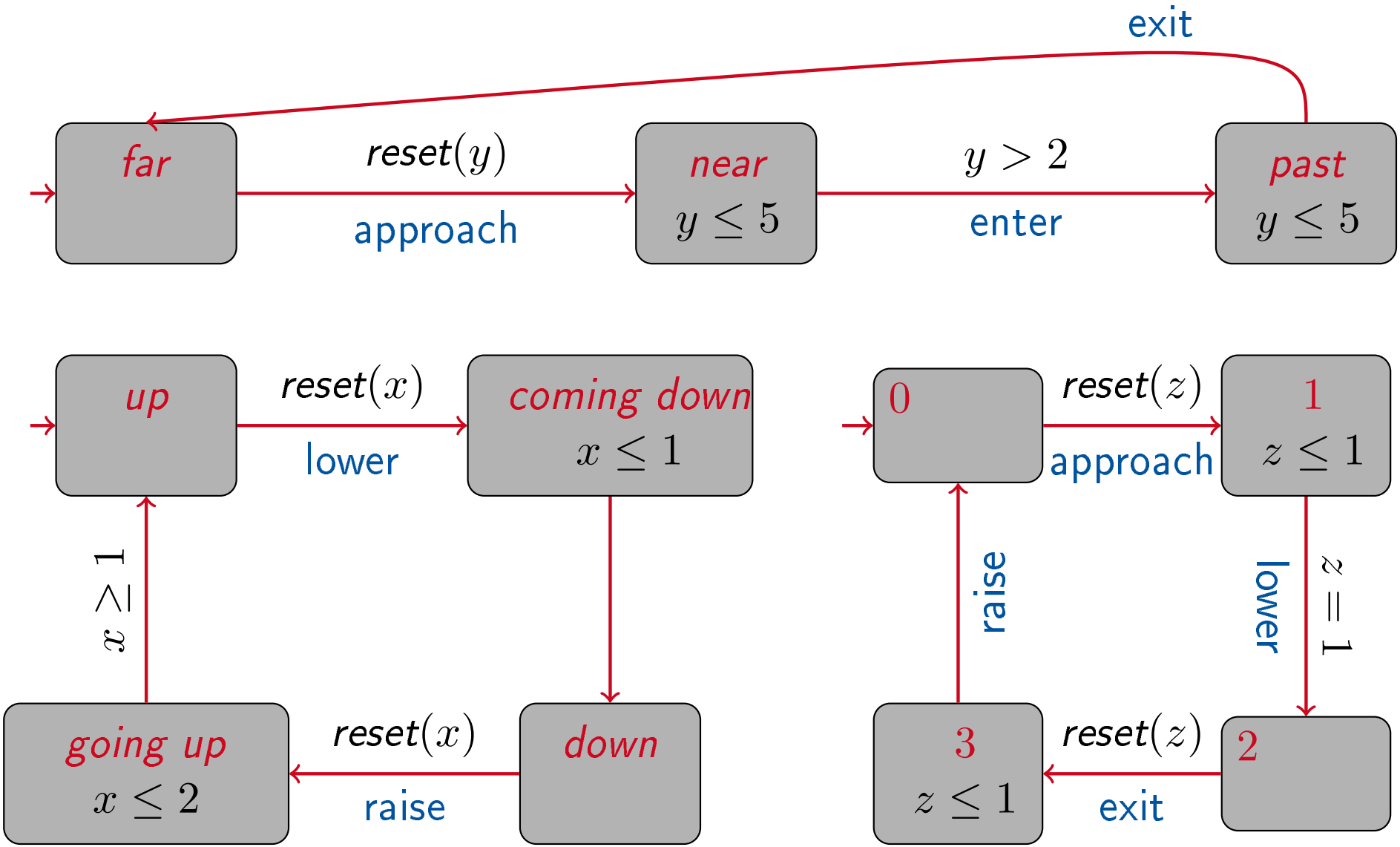




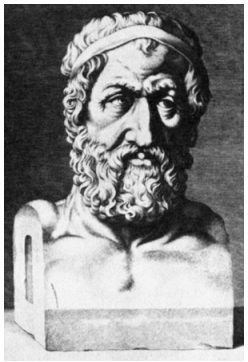
Example: Railroad Crossing



Example: Railroad Crossing

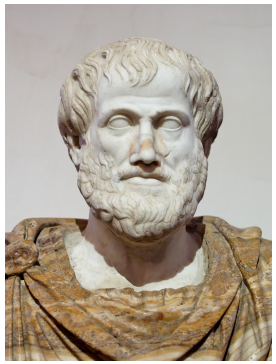


Time convergence, timelock, and Zenoness



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.

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 $\rho = s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} s_2 \xrightarrow{\alpha_2} \ldots$ we define

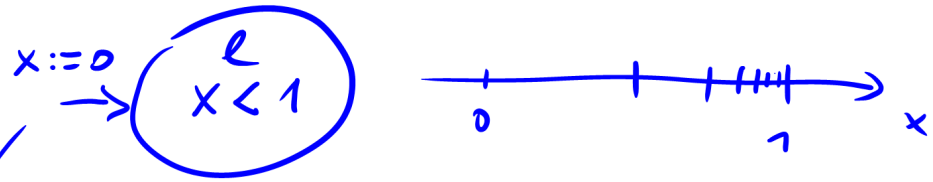
infinite

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

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).

Timelock



Definition

For a state $\sigma \in \Sigma$ let $Paths_{div}(\sigma)$ be the set of time-divergent paths starting in σ.
A state $\sigma \in \Sigma$ has a timelock iff $Paths_{div}(\sigma) = \emptyset$.
A timed automaton has a timelock iff at least one of its reachable states has a timelock.
If a timed automaton does not have a timelock then it is called timelock-free.

Timelocks are modelling flaws and should be avoided.

Zenoness

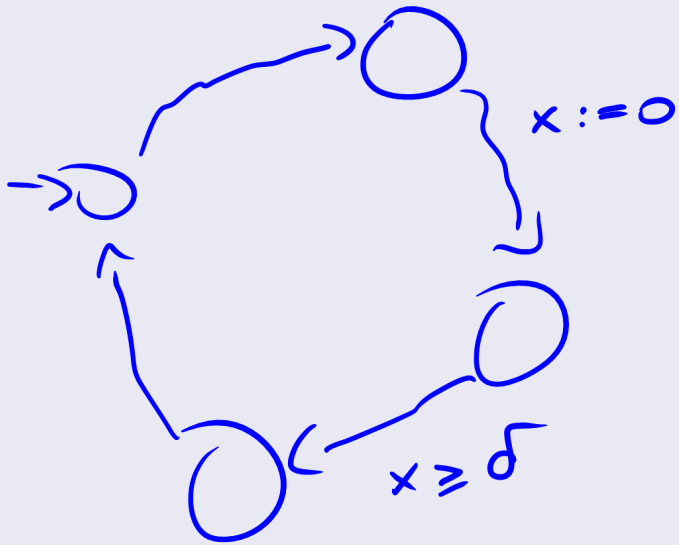
Definition

An infinite path 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 non-realisable behaviour, since their execution would require infinitely fast processors.
- Though Zeno paths are modelling flaws, they are not always easy to avoid.
- 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)



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

$$l_0 \overset{a_1:g_1,R_1}{\hookrightarrow} l_1 \overset{a_2:g_2,R_2}{\hookrightarrow} l_2 \ldots \overset{a_n:g_n,R_n}{\hookrightarrow} l_n = l_0$$

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

- *$x \in R_i$ for some $1 \leq i \leq n$, and*
- *for all valuations $\nu \in V$ there exist some $1 \leq j \leq n$ and $d \in \mathbb{N}^{>0}$ with*

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

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

Contents

1 Motivation

2 Timed automata

3 Timed computation tree logic (TCTL)

4 TCTL model checking for timed automata

TCTL

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

  State formulae

  $$\psi \quad ::= \quad \mathit{true} \;\mid\; a \;\mid\; g \;\mid\; \psi \wedge \psi \;\mid\; \neg\psi \;\mid\; \mathbf{E}\varphi \;\mid\; \mathbf{A}\varphi$$

  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 ∞).
- Note: no next-time operator

TCTL syntax

Syntactic sugar:

$$
\begin{aligned}
\mathcal{F}^J\psi &:= \text{true}\ \mathcal{U}^J\ \psi \\
\mathbf{E}\mathcal{G}^J\psi &:= \neg\mathbf{A}\mathcal{F}^J\neg\psi \\
\mathbf{A}\mathcal{G}^J\psi &:= \neg\mathbf{E}\mathcal{F}^J\neg\psi \\
\psi_1\ \mathcal{U}\ \psi_1 &:= \psi_1\ \mathcal{U}^{[0,\infty)}\ \psi_2 \\
\mathcal{F}\psi &:= \mathcal{F}^{[0,\infty)}\psi \\
\mathcal{G}\psi &:= \mathcal{G}^{[0,\infty)}\psi
\end{aligned}
$$

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 labelling function.The function $\models$ assigns a truth value to each TCTL state and path formulae as follows:

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

semantics of clock constraints

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 ψ_1 is satisfied.

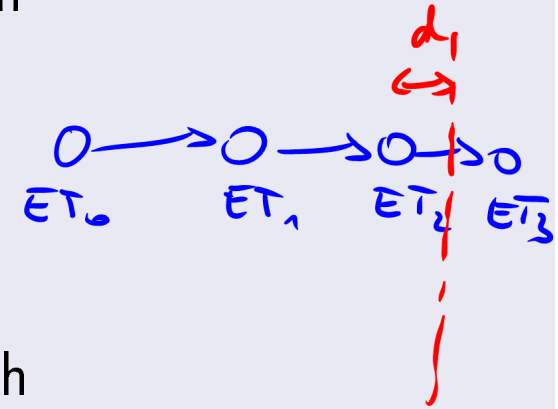
TCTL semantics

Definition (TCTL continuous semantics)

For a time-divergent path $\pi = (\ell_0, \nu_0) \xrightarrow{\alpha_0} (\ell_1, \nu_1) \xrightarrow{\alpha_1} \ldots$ we define $\pi \models \psi_1 \; \mathcal{U}^J \; \psi_2$ iff

- $\exists i \geq 0.\ (\ell_i, \nu_i + d) \models \psi_2$ for some $d \in [0, d_i]$ with

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



- $\forall j \leq i.\ (\ell_j, \nu_j + d') \models \psi_1$ 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

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

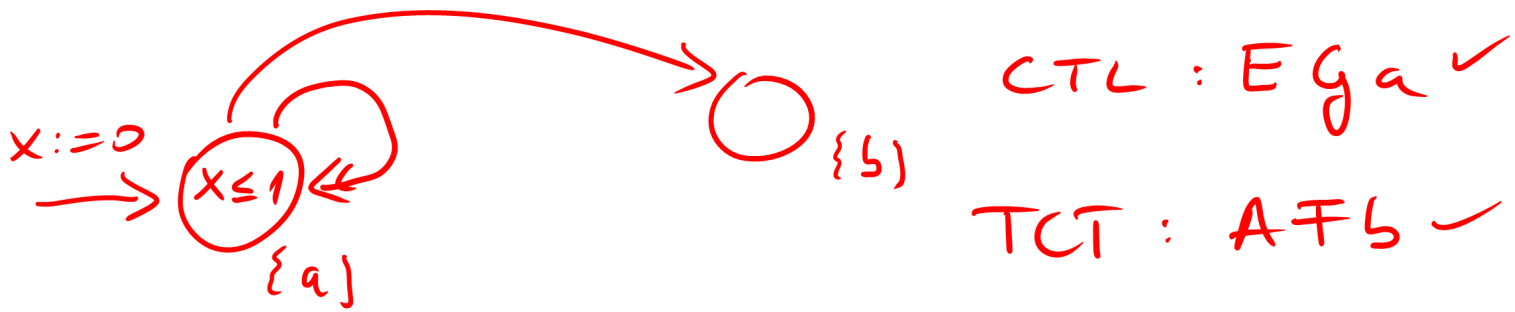
$\mathcal{T}$ satisfies ψ iff ψ holds in all initial states:

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

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!



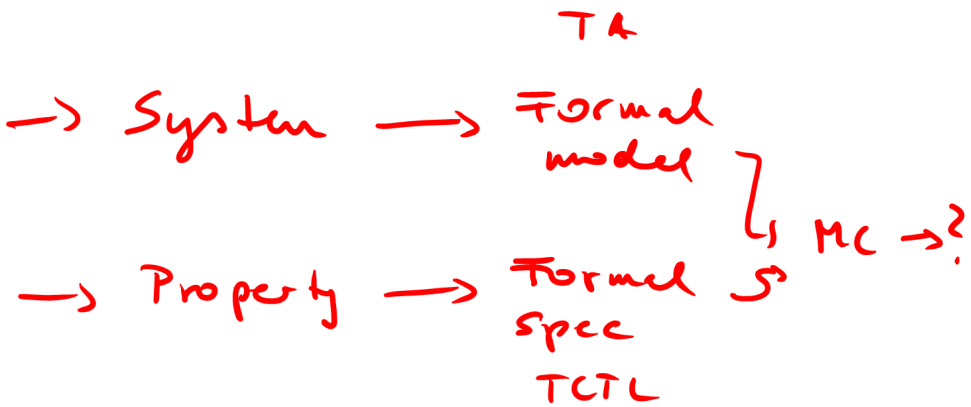
Contents

1 Motivation

2 Timed automata

3 Timed computation tree logic (TCTL)

4 TCTL model checking for timed automata



Basic method: Abstraction

- Given: a concrete system
  (here: a timed automaton)
- Goal: reduce the size of the system by abstraction
  (here: reduce the infinite state space to a finite one)
- Result: abstract system
  (here: region transition system)

Basic method: Abstraction

- Given: a concrete system
  (here: a timed automaton)
- Goal: reduce the size of the system by abstraction
  (here: reduce the infinite state space to a finite one)
- Result: abstract system
  (here: region transition system)
- Behaviorally equivalent abstraction: If we see both the concrete and the abstract system as black boxes and make experiments with them, we cannot distinguish between their observable behavior.

Basic method: Abstraction

- Given: a concrete system
  (here: a timed automaton)
- Goal: reduce the size of the system by abstraction
  (here: reduce the infinite state space to a finite one)
- Result: abstract system
  (here: region transition system)
- Behaviorally equivalent abstraction: If we see both the concrete and the abstract system as black boxes and make experiments with them, we cannot distinguish between their observable behavior.
- Two systems P and P' have the same observable behaviour iff for each context C we have that $[\![C[P]]\!] = [\![C[P']]\!]$.
  ($C[P]$: the composition of C and P, $[\![\cdot]\!]$: (global) semantics)
- E.g., for programs it could mean the same input-output behaviour. For model checking we require that they satisfy the same formulas of the underlying logic.
  (here: TCTL)

```
m( int x )
    return 2 · x
}
```

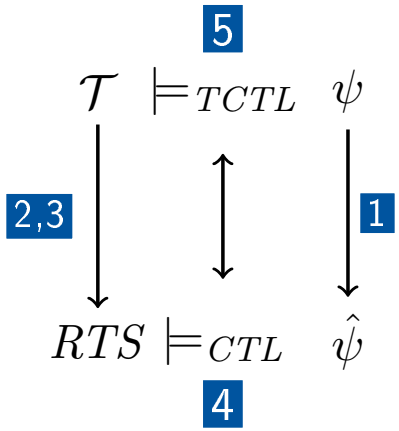
```
m'(int x) {
    int u;
    u := x
    u := u + x
    return u
}
```

TCTL model checking

Input: timed automaton $\mathcal{T}$, TCTL formula ψ

Output: the answer to the question whether $\mathcal{T} \models \psi$

1. Eliminate the timing parameters from the TCTL formula ψ, resulting in a CTL formula $\hat{\psi}$ (with clock constraints as atomic propositions).
2. Make a finite abstraction of the state space of $\mathcal{T}$.
3. Construct abstract state transition system RTS (with the states labeled by clock constraints as atomic propositions) such that $\mathcal{T} \models_{TCTL} \psi$ iff $RTS \models_{CTL} \hat{\psi}$.
4. Apply CTL model checking to check whether $RTS \models_{CTL} \hat{\psi}$.
5. Return the model checking result.

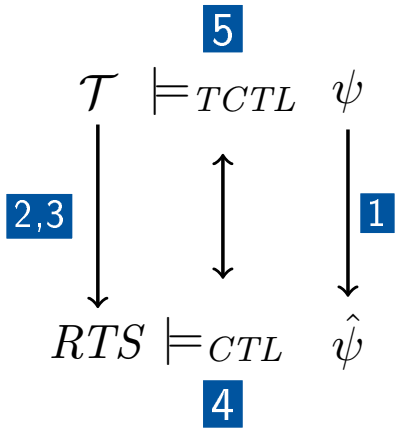


TCTL model checking

Input: timed automaton $\mathcal{T}$, TCTL formula ψ

Output: the answer to the question whether $\mathcal{T} \models \psi$

1. Eliminate the timing parameters from the TCTL formula ψ, resulting in a CTL formula $\hat{\psi}$ (with clock constraints as atomic propositions).
2. Make a finite abstraction of the state space of $\mathcal{T}$.
3. Construct abstract state transition system RTS (with the states labeled by clock constraints as atomic propositions) such that $\mathcal{T} \models_{TCTL} \psi$ iff $RTS \models_{CTL} \hat{\psi}$.
4. Apply CTL model checking to check whether $RTS \models_{CTL} \hat{\psi}$.
5. Return the model checking result.



1. Eliminating timing parameters

Let $\mathcal{T}$ be a timed automaton with clock set $\mathcal{C}$ and atomic propositions AP.
Let $\mathcal{T}'$ result from $\mathcal{T}$ by adding a fresh clock z which never gets reset.

1. Eliminating timing parameters

Let $\mathcal{T}$ be a timed automaton with clock set $\mathcal{C}$ and atomic propositions AP.
Let $\mathcal{T}'$ result from $\mathcal{T}$ by adding a fresh clock z which never gets reset.

For any state σ of $\mathcal{T}$ it holds that

1. Eliminating timing parameters

Let $\mathcal{T}$ be a timed automaton with clock set $\mathcal{C}$ and atomic propositions AP.
Let $\mathcal{T}'$ result from $\mathcal{T}$ by adding a fresh clock z which never gets reset.

For any state σ of $\mathcal{T}$ it holds that

1 $\sigma \models_{TCTL} \mathbf{E}(\psi_1 \ \mathcal{U}^J \ \psi_2)$ *iff*

1. Eliminating timing parameters

Let $\mathcal{T}$ be a timed automaton with clock set $\mathcal{C}$ and atomic propositions AP.
Let $\mathcal{T}'$ result from $\mathcal{T}$ by adding a fresh clock z which never gets reset.

For any state σ of $\mathcal{T}$ it holds that

1. $$\begin{array}{lll} \sigma & \models_{TCTL} & \mathbf{E}(\psi_1 \quad \mathcal{U}^J \quad \psi_2) \textit{ iff} \\ \textit{reset}(z) \textit{ in } \sigma & \models_{TCTL} & \mathbf{E}\,(\psi_1 \quad \mathcal{U} \quad ((z \in J) \wedge \psi_2)\,. \end{array}$$

1. Eliminating timing parameters

Let $\mathcal{T}$ be a timed automaton with clock set $\mathcal{C}$ and atomic propositions AP.
Let $\mathcal{T}'$ result from $\mathcal{T}$ by adding a fresh clock z which never gets reset.

For any state σ of $\mathcal{T}$ it holds that

1. $\sigma \models_{TCTL} \mathbf{E}(\psi_1 \ \mathcal{U}^J \ \psi_2)$ *iff*
   $\mathit{reset}(z)\ \mathit{in}\ \sigma \models_{TCTL} \mathbf{E}\,(\psi_1 \ \mathcal{U} \ ((z \in J) \wedge \psi_2))$.

2. $\sigma \models_{TCTL} \mathbf{A}(\psi_1 \ \mathcal{U}^J \ \psi_2)$ *iff*

1. Eliminating timing parameters

Let $\mathcal{T}$ be a timed automaton with clock set $\mathcal{C}$ and atomic propositions AP.
Let $\mathcal{T}'$ result from $\mathcal{T}$ by adding a fresh clock z which never gets reset.

For any state σ of $\mathcal{T}$ it holds that

1. $\sigma \models_{TCTL} \mathbf{E}(\psi_1 \ \mathcal{U}^J \ \psi_2)$ *iff* $\mathit{reset}(z) \text{ in } \sigma \models_{TCTL} \mathbf{E}\,(\psi_1 \ \mathcal{U} \ ((z \in J) \wedge \psi_2))$.

2. $\sigma \models_{TCTL} \mathbf{A}(\psi_1 \ \mathcal{U}^J \ \psi_2)$ *iff* $\mathit{reset}(z) \text{ in } \sigma \models_{TCTL} \mathbf{A}\,(\psi_1 \ \mathcal{U} \ ((z \in J) \wedge \psi_2))$.

1. Eliminating timing parameters

Let $\mathcal{T}$ be a timed automaton with clock set $\mathcal{C}$ and atomic propositions AP.
Let $\mathcal{T}'$ result from $\mathcal{T}$ by adding a fresh clock z which never gets reset.

For any state σ of $\mathcal{T}$ it holds that

1. $\sigma \models_{TCTL} \mathbf{E}(\psi_1 \ \mathcal{U}^J \ \psi_2)$ *iff*
   $\mathit{reset}(z) \text{ in } \sigma \models_{TCTL} \mathbf{E}\,(\psi_1 \ \mathcal{U} \ ((z \in J) \wedge \psi_2))$.

2. $\sigma \models_{TCTL} \mathbf{A}(\psi_1 \ \mathcal{U}^J \ \psi_2)$ *iff*
   $\mathit{reset}(z) \text{ in } \sigma \models_{TCTL} \mathbf{A}\,(\psi_1 \ \mathcal{U} \ ((z \in J) \wedge \psi_2))$.

3. $\sigma \models_{TCTL} \mathbf{E}\mathcal{F}^{\leq 2}\psi_1$ *iff*

1. Eliminating timing parameters

Let $\mathcal{T}$ be a timed automaton with clock set $\mathcal{C}$ and atomic propositions AP.
Let $\mathcal{T}'$ result from $\mathcal{T}$ by adding a fresh clock z which never gets reset.

For any state σ of $\mathcal{T}$ it holds that

1. $\sigma \models_{TCTL} \mathbf{E}(\psi_1 \ \mathcal{U}^J \ \psi_2)$ *iff* $\mathit{reset}(z) \text{ in } \sigma \models_{TCTL} \mathbf{E}(\psi_1 \ \mathcal{U} \ ((z \in J) \wedge \psi_2))$.

2. $\sigma \models_{TCTL} \mathbf{A}(\psi_1 \ \mathcal{U}^J \ \psi_2)$ *iff* $\mathit{reset}(z) \text{ in } \sigma \models_{TCTL} \mathbf{A}(\psi_1 \ \mathcal{U} \ ((z \in J) \wedge \psi_2))$.

3. $\sigma \models_{TCTL} \mathbf{E}\mathcal{F}^{\leq 2}\psi_1$ *iff* $\mathit{reset}(z) \text{ in } \sigma \models_{TCTL} \mathbf{E}\mathcal{F}((z \leq 2) \wedge \psi_1)$

1. Eliminating timing parameters

Let $\mathcal{T}$ be a timed automaton with clock set $\mathcal{C}$ and atomic propositions AP.
Let $\mathcal{T}'$ result from $\mathcal{T}$ by adding a fresh clock z which never gets reset.

For any state σ of $\mathcal{T}$ it holds that

1. $\sigma \models_{TCTL} \mathbf{E}(\psi_1 \ \mathcal{U}^J \ \psi_2)$ *iff* $\mathit{reset}(z) \text{ in } \sigma \models_{TCTL} \mathbf{E}(\psi_1 \ \mathcal{U} \ ((z \in J) \wedge \psi_2))$.
2. $\sigma \models_{TCTL} \mathbf{A}(\psi_1 \ \mathcal{U}^J \ \psi_2)$ *iff* $\mathit{reset}(z) \text{ in } \sigma \models_{TCTL} \mathbf{A}(\psi_1 \ \mathcal{U} \ ((z \in J) \wedge \psi_2))$.
3. $\sigma \models_{TCTL} \mathbf{E}\mathcal{F}^{\leq 2}\psi_1$ *iff* $\mathit{reset}(z) \text{ in } \sigma \models_{TCTL} \mathbf{E}\mathcal{F}((z \leq 2) \wedge \psi_1)$
4. $\sigma \models_{TCTL} \mathbf{E}\mathcal{G}^{\leq 2}\psi_1$ *iff*

1. Eliminating timing parameters

Let $\mathcal{T}$ be a timed automaton with clock set $\mathcal{C}$ and atomic propositions AP.
Let $\mathcal{T}'$ result from $\mathcal{T}$ by adding a fresh clock z which never gets reset.

For any state σ of $\mathcal{T}$ it holds that

1. $$\begin{array}{lll} \sigma & \models_{TCTL} & \mathbf{E}(\psi_1 \quad \mathcal{U}^J \quad \psi_2) \text{ iff} \\ \textit{reset}(z) \text{ in } \sigma & \models_{TCTL} & \mathbf{E}\,(\psi_1 \quad \mathcal{U} \quad ((z \in J) \wedge \psi_2)\,. \end{array}$$

2. $$\begin{array}{lll} \sigma & \models_{TCTL} & \mathbf{A}(\psi_1 \quad \mathcal{U}^J \quad \psi_2) \text{ iff} \\ \textit{reset}(z) \text{ in } \sigma & \models_{TCTL} & \mathbf{A}\,(\psi_1 \quad \mathcal{U} \quad ((z \in J) \wedge \psi_2)\,. \end{array}$$

3. $\sigma \models_{TCTL} \mathbf{E}\mathcal{F}^{\leq 2}\psi_1 \quad \textit{iff} \quad \textit{reset}(z) \text{ in } \sigma \models_{TCTL} \mathbf{E}\mathcal{F}((z \leq 2) \wedge \psi_1)$

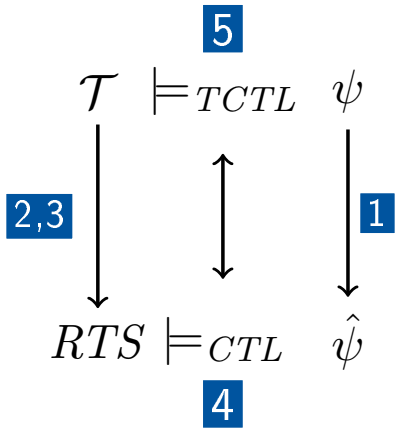
4. $\sigma \models_{TCTL} \mathbf{E}\mathcal{G}^{\leq 2}\psi_1 \quad \textit{iff} \quad \textit{reset}(z) \text{ in } \sigma \models_{TCTL} \mathbf{E}\mathcal{G}((z \leq 2) \rightarrow \psi_1)$

TCTL model checking

Input: timed automaton $\mathcal{T}$, TCTL formula ψ

Output: the answer to the question whether $\mathcal{T} \models \psi$

1. Eliminate the timing parameters from the TCTL formula ψ, resulting in a CTL formula $\hat{\psi}$ (with clock constraints as atomic propositions).
2. Make a finite abstraction of the state space of $\mathcal{T}$.
3. Construct abstract state transition system RTS (with the states labeled by clock constraints as atomic propositions) such that $\mathcal{T} \models_{TCTL} \psi$ iff $RTS \models_{CTL} \hat{\psi}$.
4. Apply CTL model checking to check whether $RTS \models_{CTL} \hat{\psi}$.
5. Return the model checking result.



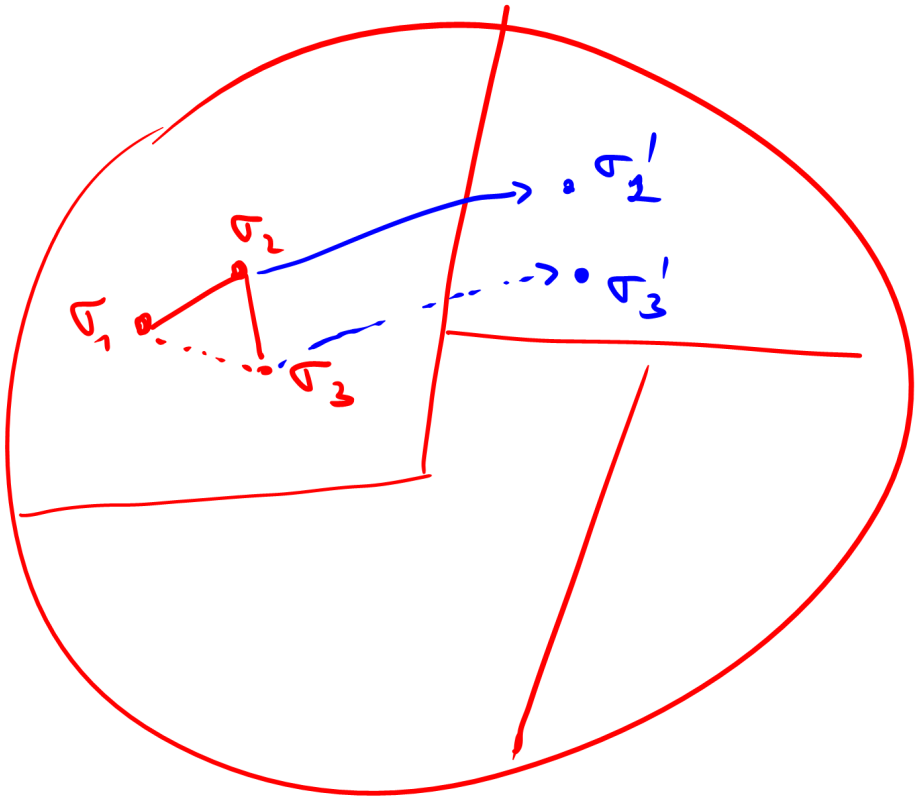
Keywords:

Finite abstraction

Equivalence relation, equivalence classes

Bisimulation

And what does it mean in our context?



Eq. rel

$R \subseteq \Sigma \times \Sigma$

① $(\sigma_1, \sigma_2) \in R \rightarrow$
$(\sigma_2, \sigma_1) \in R$
(symmetry)

② $(\sigma_1, \sigma_1) \in R$
(refl.)

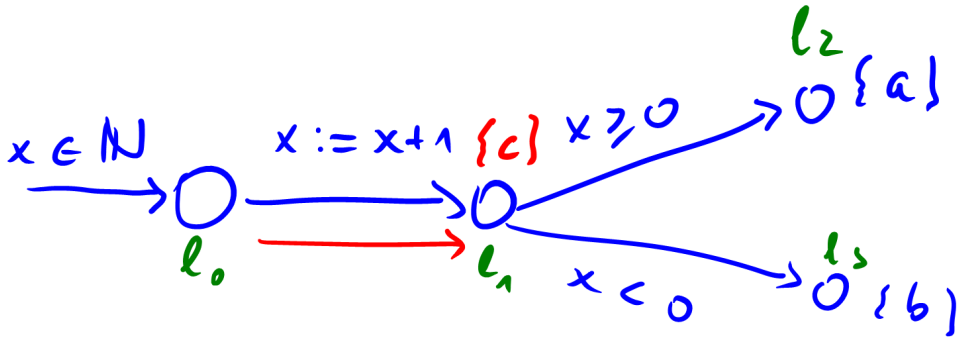
③ $(\sigma_1, \sigma_2) \in R \wedge$
$(\sigma_2, \sigma_3) \in R \rightarrow$
$(\sigma_1, \sigma_3) \in R$

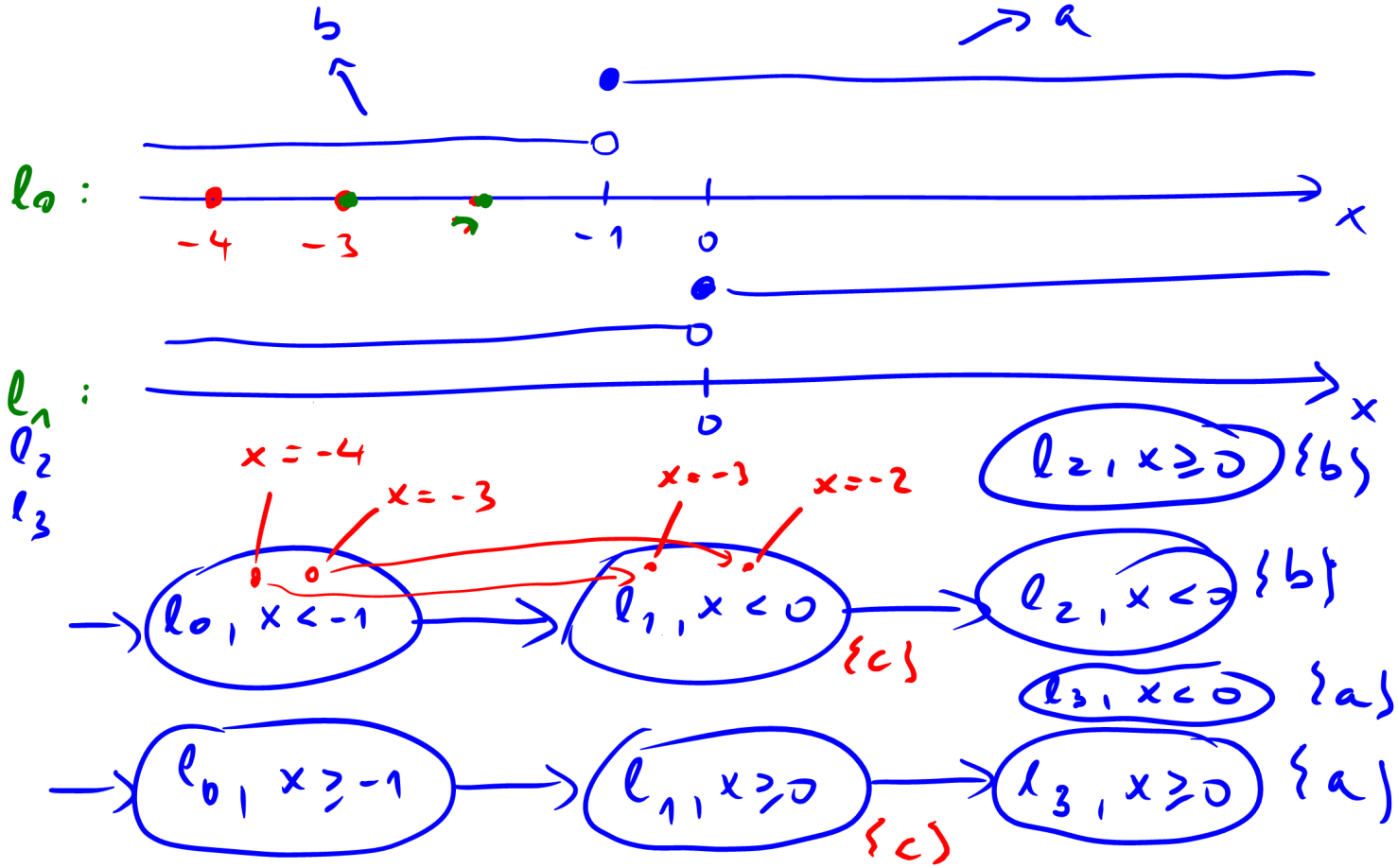
2. Finite state space abstraction

We search for an equivalence relation $\cong$ on states, such that equivalent states satisfy the same (sub)formulae ψ' occurring in the timed automaton $\mathcal{T}$ or in the specification ψ:

$$\sigma \cong \sigma' \quad \Rightarrow \quad \left(\sigma \models \psi' \quad \textit{iff} \quad \sigma' \models \psi'\right).$$

We strive for a finite number of equivalence classes.


$x \in \mathbb{N}$
$x := x+1$ {c}
$x \geq 0$
$x < 0$
ℓ_0
ℓ_1
ℓ_2 {a}
ℓ_3 {b}


b
a
ℓ_0 :
ℓ_1 :
ℓ_2
ℓ_3
-4
-3
-1
0
x
x = -4
x = -3
x = -3
x = -2
$\ell_0, x < -1$
$\ell_1, x < 0$
$\ell_2, x \geq 0$ {b}
$\ell_2, x < 0$ {b}
$\ell_3, x < 0$ {a}
$\ell_0, x \geq -1$
$\ell_1, x \geq 0$
$\ell_3, x \geq 0$ {a}
{c}
{c}

2. Finite state space abstraction

We search for an equivalence relation $\cong$ on states, such that equivalent states satisfy the same (sub)formulae ψ' occurring in the timed automaton $\mathcal{T}$ or in the specification ψ:

$$\sigma \cong \sigma' \quad \Rightarrow \quad \left(\sigma \models \psi' \quad \textit{iff} \quad \sigma' \models \psi'\right).$$

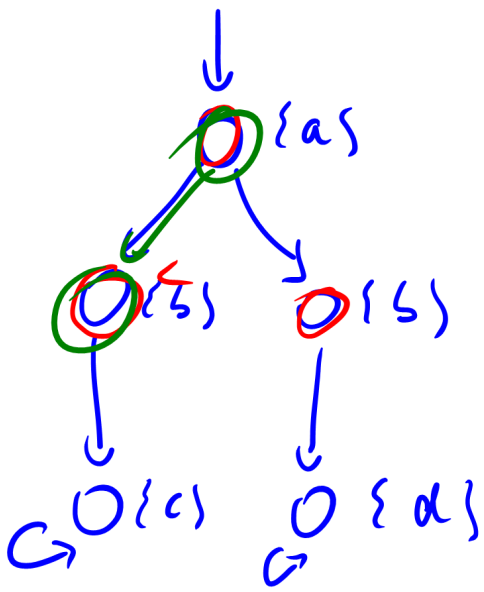
We strive for a finite number of equivalence classes.

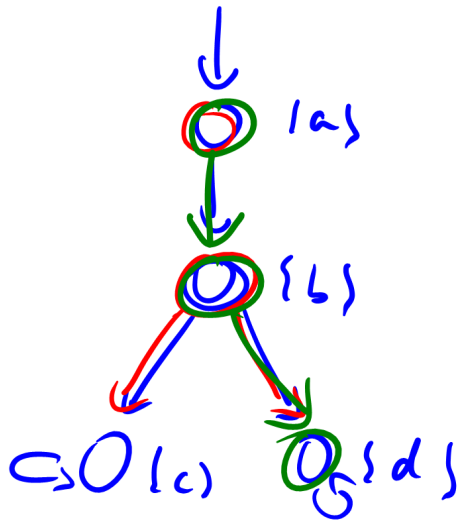
Definition

Let $\mathcal{LSTS} = (\Sigma, Lab, Edge, Init)$ be a labeled state transition system, AP a set of atomic propositions, and $L : \Sigma \to 2^{AP}$ a labeling function for $\mathcal{LSTS}$ over AP.

A bisimulation for $\mathcal{LSTS}$ and L is an equivalence relation $\approx \subseteq \Sigma \times \Sigma$ such that for all $\sigma_1 \approx \sigma_2$

1. $L(\sigma_1) = L(\sigma_2)$
2. for all $\sigma_1' \in \Sigma$ with $\sigma_1 \xrightarrow{a} \sigma_1'$ there exists $\sigma_2' \in \Sigma$ such that $\sigma_2 \xrightarrow{a} \sigma_2'$ and $\sigma_1' \approx \sigma_2'$.


{a}
{b}
{b}
{c}
{d}


{a}
{b}
{c}
{d}

Time abstract bisimulation

Definition

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 labeling function for $\mathcal{T}$ over AP. A time abstract bisimulation for $\mathcal{T}$ and L is an equivalence relation $\approx \subseteq \Sigma \times \Sigma$ such that for all $\sigma_1, \sigma_2 \in \Sigma$ satisfying $\sigma_1 \approx \sigma_2$

- $L(\sigma_1) = L(\sigma_2)$
- for all $\sigma_1' \in \Sigma$ with $\sigma_1 \xrightarrow{a} \sigma_1'$ there is a $\sigma_2' \in \Sigma$ such that $\sigma_2 \xrightarrow{a} \sigma_2'$ and $\sigma_1' \approx \sigma_2'$
- for all $\sigma_1' \in \Sigma$ with $\sigma_1 \xrightarrow{t_1} \sigma_1'$ there is a $\sigma_2' \in \Sigma$ such that $\sigma_2 \xrightarrow{t_2} \sigma_2'$ and $\sigma_1' \approx \sigma_2'$.

Time abstract bisimulation

Definition

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 labeling function for $\mathcal{T}$ over AP. A time abstract bisimulation for $\mathcal{T}$ and L is an equivalence relation $\approx \subseteq \Sigma \times \Sigma$ such that for all $\sigma_1, \sigma_2 \in \Sigma$ satisfying $\sigma_1 \approx \sigma_2$

- $L(\sigma_1) = L(\sigma_2)$
- for all $\sigma_1' \in \Sigma$ with $\sigma_1 \xrightarrow{a} \sigma_1'$ there is a $\sigma_2' \in \Sigma$ such that $\sigma_2 \xrightarrow{a} \sigma_2'$ and $\sigma_1' \approx \sigma_2'$
- for all $\sigma_1' \in \Sigma$ with $\sigma_1 \xrightarrow{t_1} \sigma_1'$ there is a $\sigma_2' \in \Sigma$ such that $\sigma_2 \xrightarrow{t_2} \sigma_2'$ and $\sigma_1' \approx \sigma_2'$.

Time-abstract bisimulation is sufficient for CTL-equivalence (i.e., two states are in the same equivalence class iff they satisfy the same CTL properties), but not yet sufficient for TCTL-equivalence (as we will see).

Bisimulation

Lemma

Let $\mathcal{T} = (Loc, \mathcal{C}, Lab, Edge, Inv, Init)$ *be a timed automaton with state space* $\Sigma = Loc \times V$, AP *a set of atomic propositions, and* $L : Loc \to 2^{AP}$ *a labeling function for* $\mathcal{T}$ *over* AP. *Assume furthermore a time-abstract bisimulation* $\approx \subseteq \Sigma \times \Sigma$ *for* $\mathcal{T}$ *and* L.
Then for all $\sigma, \sigma' \in \Sigma$ *with* $\sigma \approx \sigma'$ *we have that for each path*

$$\pi : \sigma \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \sigma_2 \xrightarrow{\alpha_3} \dots$$

of $\mathcal{T}$ *there exists a path*

$$\pi' : \sigma' \xrightarrow{\alpha'_1} \sigma'_1 \xrightarrow{\alpha'_2} \sigma'_2 \xrightarrow{\alpha'_3} \dots$$

of $\mathcal{T}$ *such that for all* i

- $\sigma_i \approx \sigma'_i$ *(implying* $L(\sigma_i) = L(\sigma'_i)$*,*
- $\alpha_i = \alpha'_i$ *if* $\alpha_i \in Lab$ *and*
- $\alpha_i, \alpha'_i \in \mathbb{R}_{\geq 0}$ *otherwise.*

2. Finite state space abstraction

How could a time-abstract bisimulation for a timed automaton look like?

2. Finite state space abstraction

TCTL : atomic prop.
atomic clock constr.
(+ temp.)

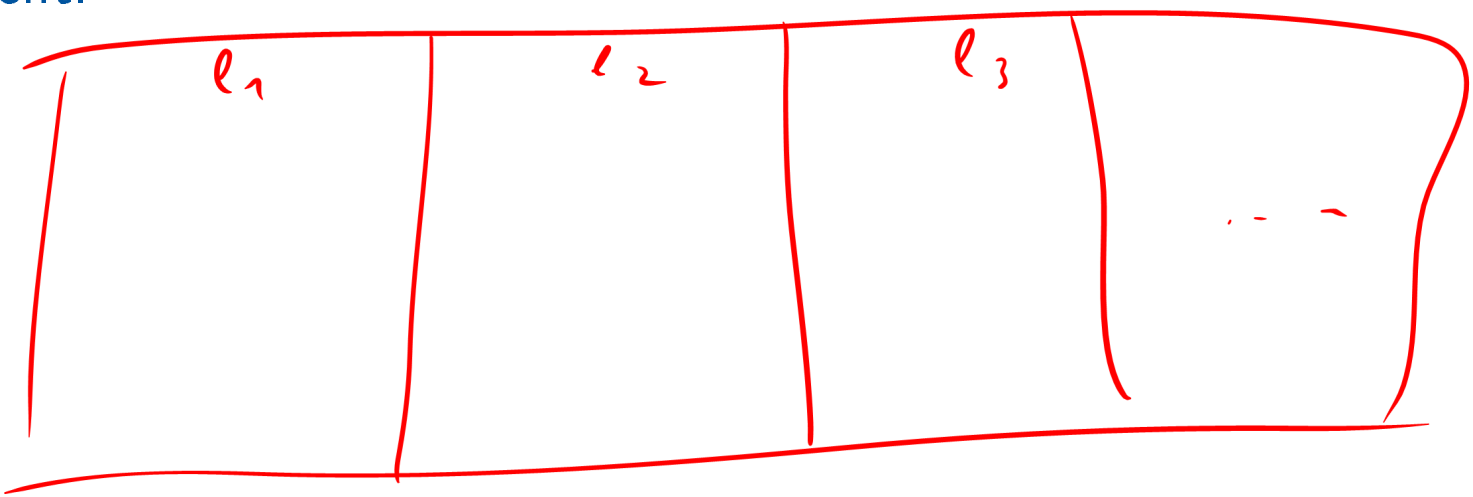
How could a time-abstract bisimulation for a timed automaton look like?

Since, in general, the atomic propositions assigned to different locations in $\mathcal{T}$ are different,

2. Finite state space abstraction

How could a time-abstract bisimulation for a timed automaton look like?

Since, in general, the atomic propositions assigned to different locations in $\mathcal{T}$ are different, only states (l, ν) and (l', ν') satisfying $l = l'$ should be equivalent.



2. Finite state space abstraction

Equivalent states should satisfy the same atomic clock constraints.

2. Finite state space abstraction

Equivalent states should satisfy the same atomic clock constraints.
Notation:

- Integer part of $r \in \mathbb{R}$: $\lfloor r \rfloor = \max \{c \in \mathbb{N} \mid c \leq r\}$
- Fractional part of $r \in \mathbb{R}$: $frac(r) = r - \lfloor r \rfloor$

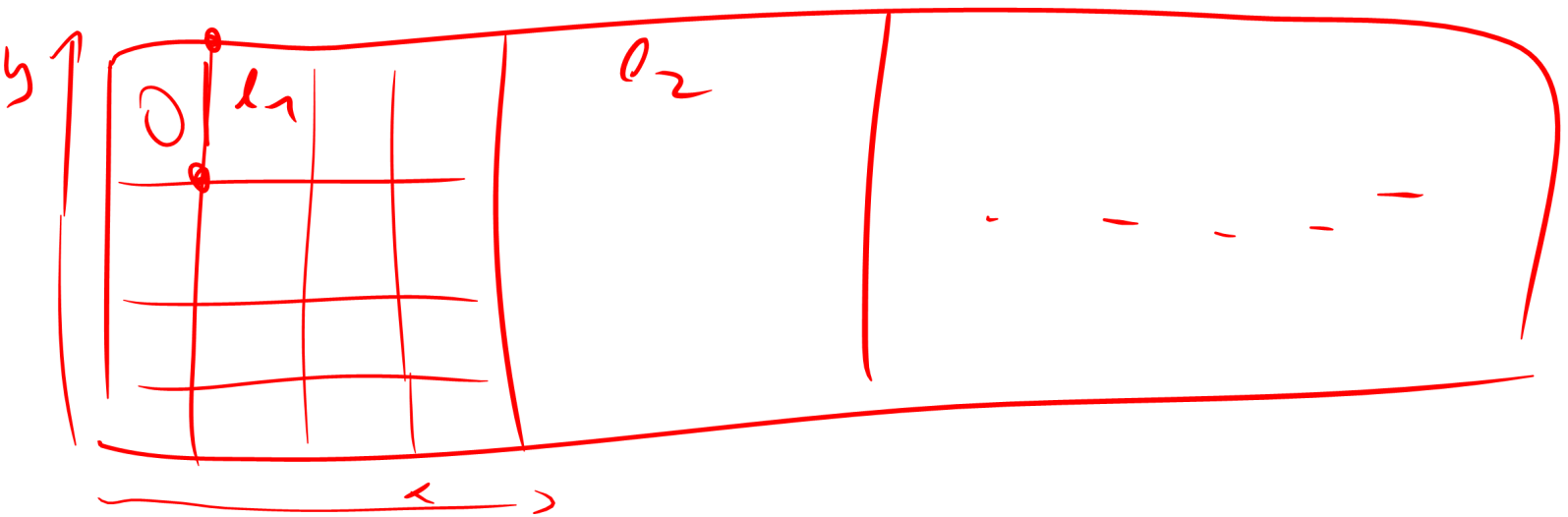
2. Finite state space abstraction

Equivalent states should satisfy the same atomic clock constraints.
Notation:

- Integer part of $r \in \mathbb{R}$: $\lfloor r \rfloor = \max\{c \in \mathbb{N} \mid c \leq r\}$
- Fractional part of $r \in \mathbb{R}$: $frac(r) = r - \lfloor r \rfloor$

$x \geq 2 \wedge x < 3$

Only states (l, ν) and (l, ν') with equal integer part $\lfloor \nu(x) \rfloor = \lfloor \nu'(x) \rfloor$ for each clock $x \in \mathcal{C}$ can be equivalent for arbitrary $\mathcal{T}$. (Assume w.l.o.g. $c = \lfloor \nu(x) \rfloor > \lfloor \nu'(x) \rfloor = c'$ and assume that $\mathcal{T}$ has an edge with source location l, guard $x \geq c$, and that the invariant in both source and target locations is *true*. Then the edge is enabled in (l, ν) but not in (l, ν').)



2. Finite state space abstraction

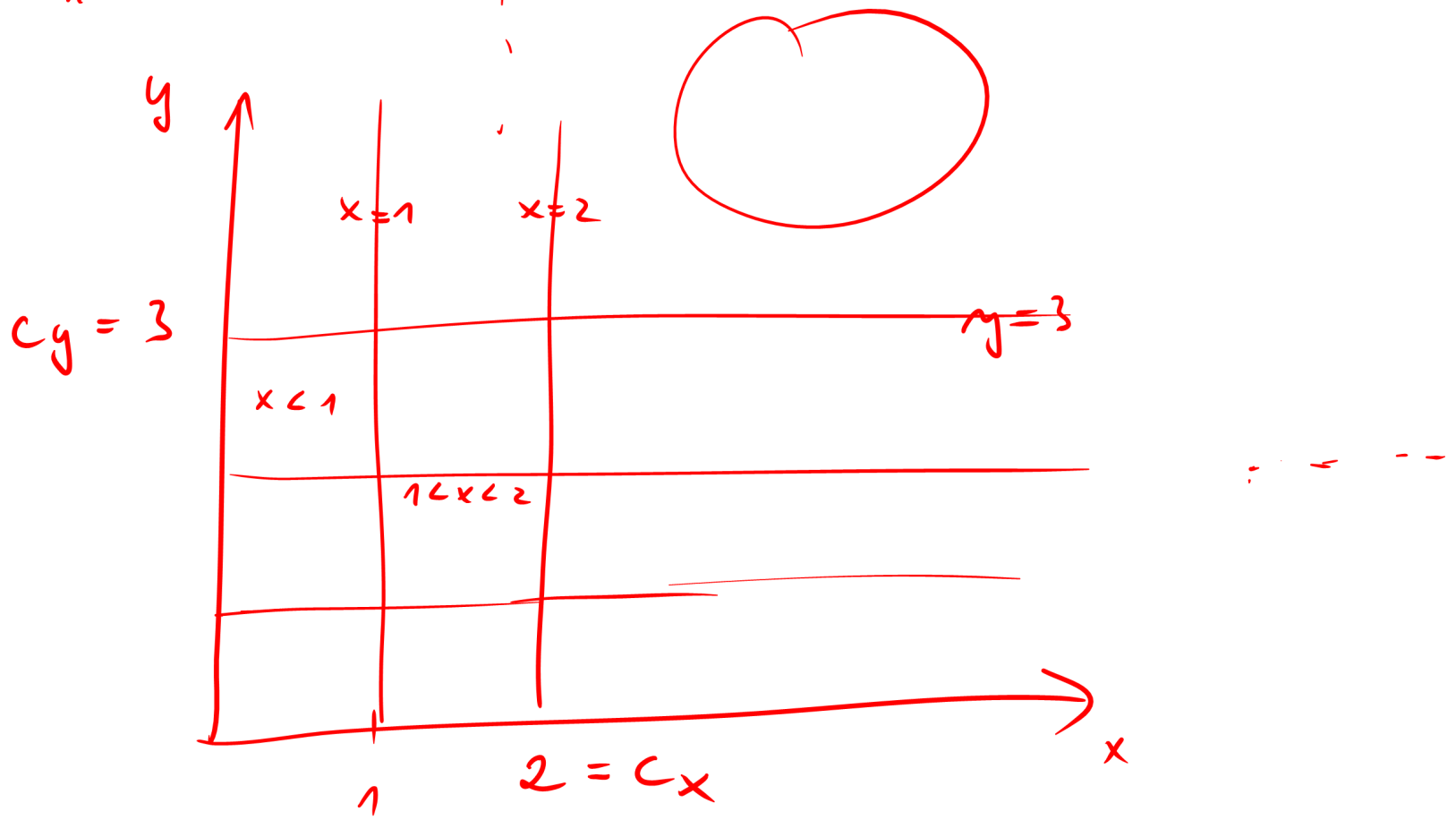
Equivalent states should satisfy the same atomic clock constraints.
Notation:

- Integer part of $r \in \mathbb{R}$: $\lfloor r \rfloor = \max\{c \in \mathbb{N} \mid c \leq r\}$
- Fractional part of $r \in \mathbb{R}$: $frac(r) = r - \lfloor r \rfloor$

Only states (l, ν) and (l, ν') with equal integer part $\lfloor \nu(x) \rfloor = \lfloor \nu'(x) \rfloor$ for each clock $x \in \mathcal{C}$ can be equivalent for arbitrary $\mathcal{T}$. (Assume w.l.o.g. $c = \lfloor \nu(x) \rfloor > \lfloor \nu'(x) \rfloor = c'$ and assume that $\mathcal{T}$ has an edge with source location l, guard $x \geq c$, and that the invariant in both source and target locations is *true*. Then the edge is enabled in (l, ν) but not in (l, ν').) $x = 2$

Only states (l, ν) and (l, ν') with $frac(\nu(x)) = 0$ iff $frac(\nu'(x)) = 0$ for all clocks $x \in \mathcal{C}$ can be equivalent for arbitrary $\mathcal{T}$. (Assume w.l.o.g. $\nu(x) = c \in \mathbb{N}$, implying $frac(\nu(x)) = 0$. If $frac(\nu'(x)) \neq 0$ then $\nu(x) \neq c$. Now if $\mathcal{T}$ has an edge with source location l and guard $x = c$, and if the invariant in both source and target locations is *true*, then the edge is enabled in (l, ν) but not in (l, ν').)

l_1 :



2. Finite state space abstraction

Equivalent states should satisfy the same atomic clock constraints.
Notation:

- Integer part of $r \in \mathbb{R}$: $\lfloor r \rfloor = \max\{c \in \mathbb{N} \mid c \leq r\}$
- Fractional part of $r \in \mathbb{R}$: $frac(r) = r - \lfloor r \rfloor$

Only states (l, ν) and (l, ν') with equal integer part $\lfloor \nu(x) \rfloor = \lfloor \nu'(x) \rfloor$ for each clock $x \in \mathcal{C}$ can be equivalent for arbitrary $\mathcal{T}$. (Assume w.l.o.g. $c = \lfloor \nu(x) \rfloor > \lfloor \nu'(x) \rfloor = c'$ and assume that $\mathcal{T}$ has an edge with source location l, guard $x \geq c$, and that the invariant in both source and target locations is *true*. Then the edge is enabled in (l, ν) but not in (l, ν').)
Only states (l, ν) and (l, ν') with $frac(\nu(x)) = 0$ iff $frac(\nu'(x)) = 0$ for all clocks $x \in \mathcal{C}$ can be equivalent for arbitrary $\mathcal{T}$. (Assume w.l.o.g. $\nu(x) = c \in \mathbb{N}$, implying $frac(\nu(x)) = 0$. If $frac(\nu'(x)) \neq 0$ then $\nu(x) \neq c$. Now if $\mathcal{T}$ has an edge with source location l and guard $x = c$, and if the invariant in both source and target locations is *true*, then the edge is enabled in (l, ν) but not in (l, ν').)
I.e., only states (l, ν) and (l, ν') satisfying

$$\lfloor \nu(x) \rfloor = \lfloor \nu'(x) \rfloor \; and \; (frac(\nu(x)) = 0 \leftrightarrow frac(\nu'(x)) = 0)$$

for all $x \in \mathcal{C}$ should be equivalent.

2. Finite state space abstraction

Problem: It would generate infinitely many equivalence classes!

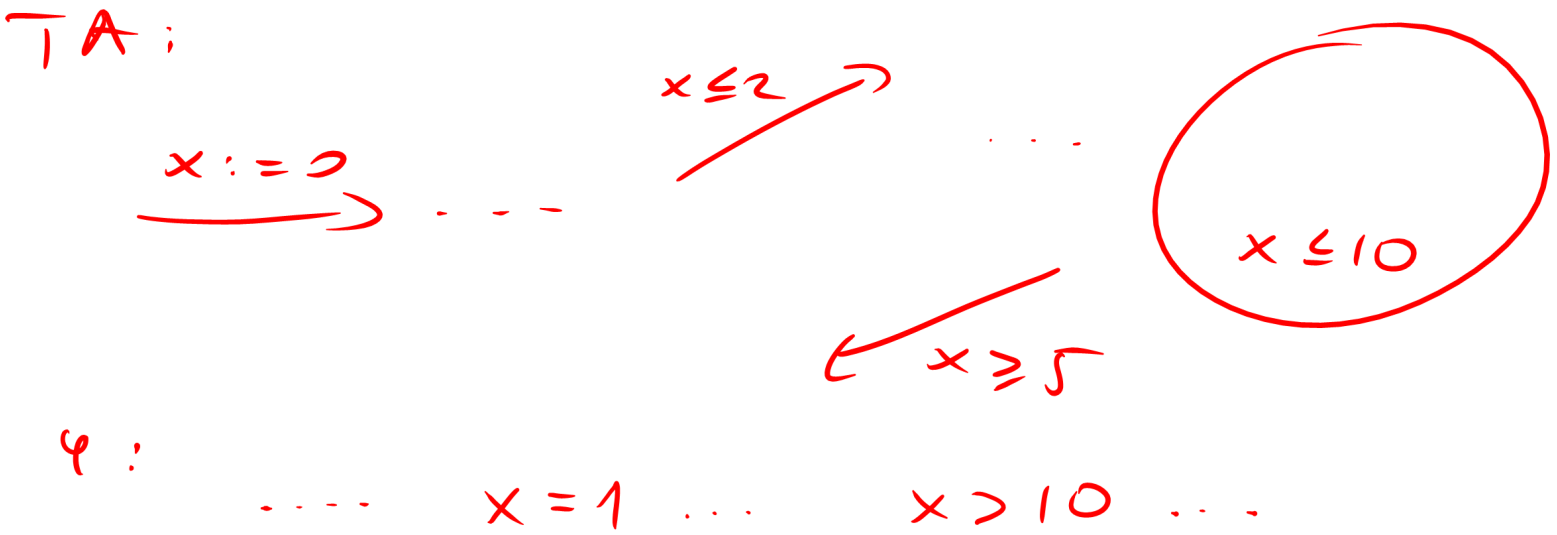
2. Finite state space abstraction

Problem: It would generate infinitely many equivalence classes!

Solution: Require equivalence not in general but only for a given $\mathcal{T}$.

Problem: It would generate infinitely many equivalence classes!
Solution: Require equivalence not in general but only for a given $\mathcal{T}$.

Let c_x be the largest constant to which a clock x is compared in either $\mathcal{T}$ or in ψ. Then there is no observation which could distinguish between the x-values in (l, ν) and (l, ν') if $\nu(x) > c_x$ and $\nu'(x) > c_x$.



Problem: It would generate infinitely many equivalence classes!
Solution: Require equivalence not in general but only for a given $\mathcal{T}$.

Let c_x be the largest constant to which a clock x is compared in either $\mathcal{T}$ or in ψ. Then there is no observation which could distinguish between the x-values in (l, ν) and (l, ν') if $\nu(x) > c_x$ and $\nu'(x) > c_x$.
I.e., only states (l, ν) and (l, ν') satisfying

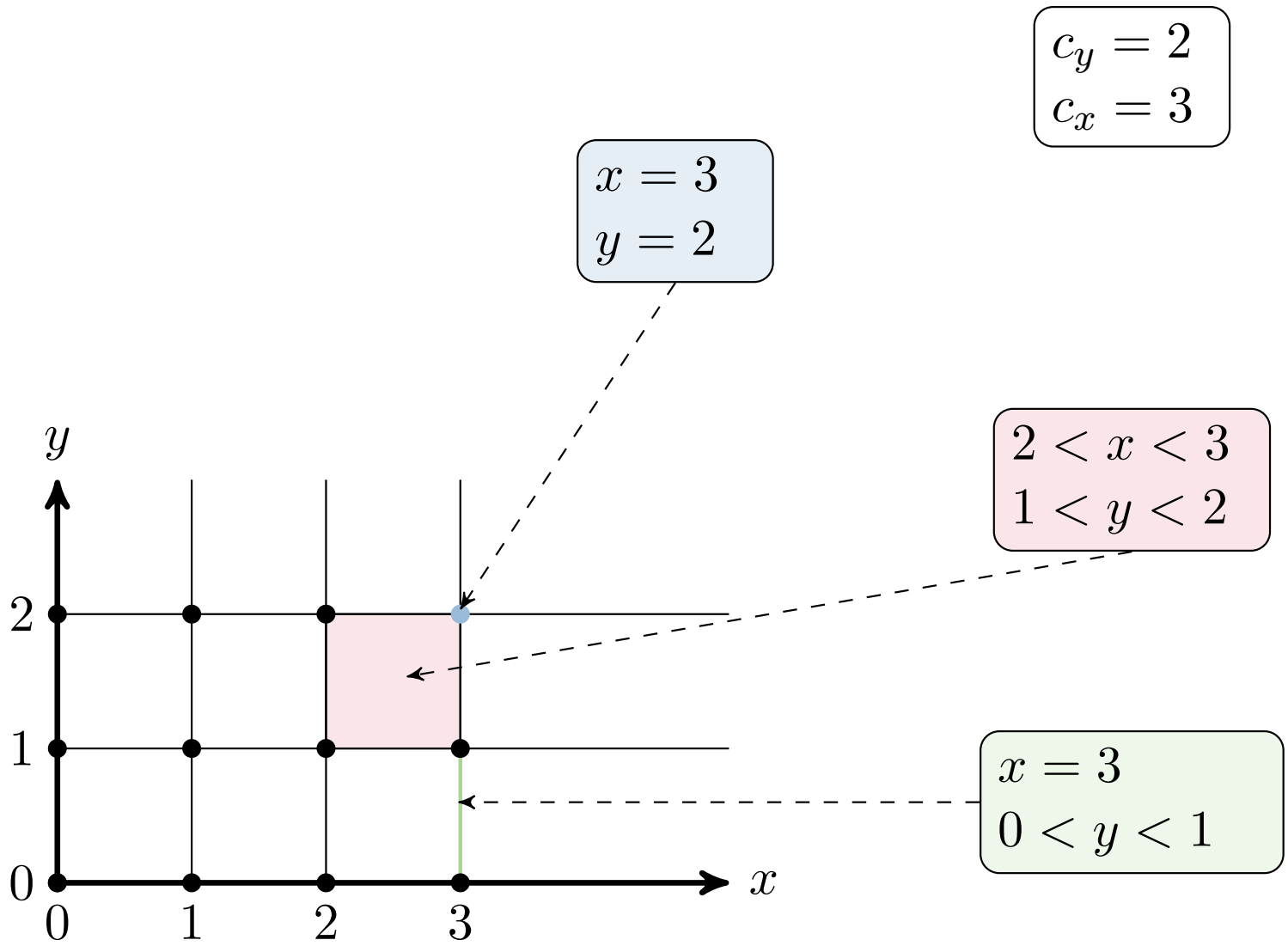
Problem: It would generate infinitely many equivalence classes!
Solution: Require equivalence not in general but only for a given $\mathcal{T}$.

Let c_x be the largest constant to which a clock x is compared in either $\mathcal{T}$ or in ψ. Then there is no observation which could distinguish between the x-values in (l, ν) and (l, ν') if $\nu(x) > c_x$ and $\nu'(x) > c_x$.
I.e., only states (l, ν) and (l, ν') satisfying

$$(\nu(x) > c_x \wedge \nu'(x) > c_x) \quad \vee$$
$$(\lfloor \nu(x) \rfloor = \lfloor \nu'(x) \rfloor \ \wedge \ (frac(\nu(x)) = 0 \ \leftrightarrow \ frac(\nu'(x)) = 0))$$

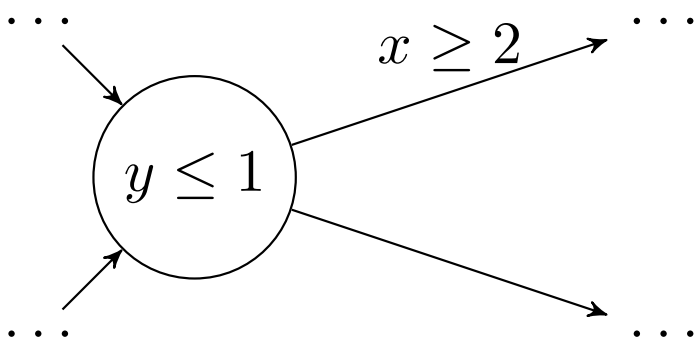
for all $x \in \mathcal{C}$ should be equivalent.

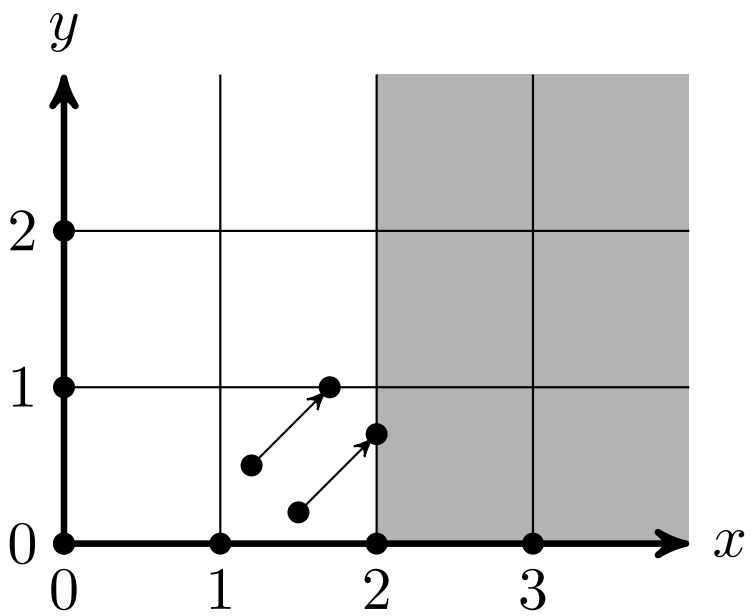
2. Finite state space abstraction



2. Finite state space abstraction

As the following example illustrates, we must make a further refinement of the abstraction, since it does not distinguish between states satisfying different formulae.





2. Finite state space abstraction

What we need is a refinement taking the order of the fractional parts of the clock values into account. However, again only for values below the largest constants to which the clocks get compared.
I.e., only states (l, ν) and (l, ν') satisfying

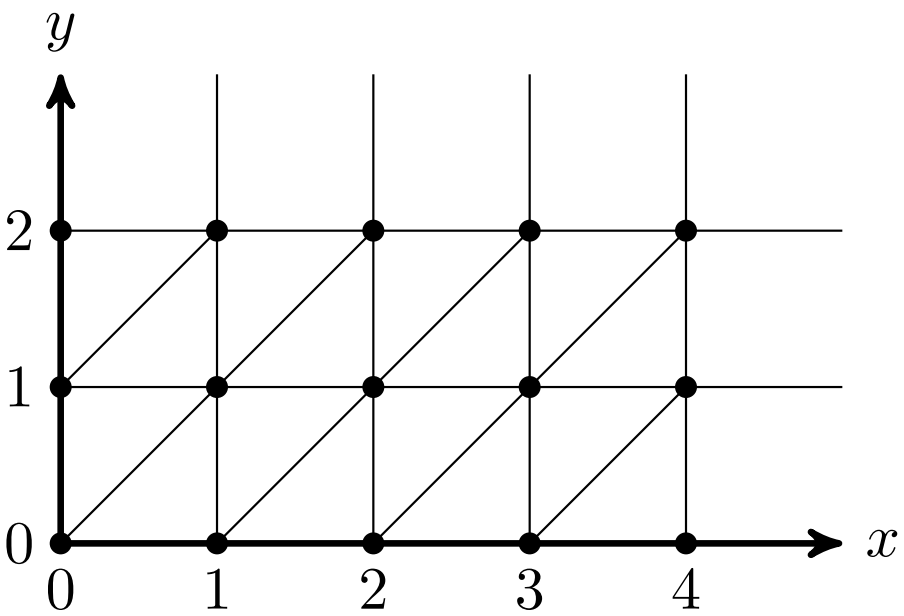
$$
\begin{aligned}
&(\nu(x) > c_x \wedge \nu'(x) > c_x) \vee (\nu(y) > c_y \wedge \nu'(y) > c_y) \quad \vee \\
&\quad (\quad frac(\nu(x)) < frac(\nu(y)) \quad \leftrightarrow \quad frac(\nu'(x)) < frac(\nu'(y)) \quad \wedge \\
&\qquad frac(\nu(x)) = frac(\nu(y)) \quad \leftrightarrow \quad frac(\nu'(x)) = frac(\nu'(y)) \quad \wedge \\
&\qquad frac(\nu(x)) > frac(\nu(y)) \quad \leftrightarrow \quad frac(\nu'(x)) > frac(\nu'(y)))
\end{aligned}
$$

for all $x, y \in \mathcal{C}$ should be equivalent.
Because of symmetry the following is sufficient:

$$
\begin{aligned}
&(\nu(x) > c_x \wedge \nu'(x) > c_x) \vee (\nu(y) > c_y \wedge \nu'(y) > c_y) \quad \vee \\
&\quad (frac(\nu(x)) \leq frac(\nu(y)) \quad \leftrightarrow \quad frac(\nu'(x)) \leq frac(\nu'(y)))
\end{aligned}
$$

for all $x, y \in \mathcal{C}$.

2. Finite state space abstraction



$$c_x = 4$$
$$c_y = 2$$

finite index

Definition

Assume a timed automaton $\mathcal{T} = (Loc, \mathcal{C}, Lab, Edge, Inv, Init)$ and a TCTL formula ψ, both over a clock set $\mathcal{C}$, and for each $x \in \mathcal{C}$ let c_x be the largest constant to which x is compared to in $\mathcal{T}$ and ψ. We define the clock equivalence relation $\cong \subseteq \Sigma \times \Sigma$ for $\mathcal{T}$ and ψ by $(l, \nu) \cong (l', \nu')$ iff $l = l'$ and

- for all $x \in \mathcal{C}$, either $\nu(x) > c_x \wedge \nu'(x) > c_x$ or

$$\lfloor \nu(x) \rfloor = \lfloor \nu'(x) \rfloor \ \wedge \ (frac(\nu(x)) = 0 \quad \leftrightarrow \quad frac(\nu'(x)) = 0)$$

- for all $x, y \in \mathcal{C}$ if $\nu(x), \nu'(x) \leq c_x$ and $\nu(y), \nu'(y) \leq c_y$ then

$$frac(\nu(x)) \leq frac(\nu(y)) \quad \leftrightarrow \quad frac(\nu'(x)) \leq frac(\nu'(y)).$$

The clock region of an evaluation $\nu \in V$ is the set $[\nu] = \{\nu' \in V \mid \nu \cong \nu'\}$. The clock region of a state $\sigma = (l, \nu) \in \Sigma$ is the set $[\sigma] = \{(l, \nu') \in \Sigma \mid \nu \cong \nu'\}$.

Lemma

Assume a timed automaton $\mathcal{T} = (Loc, \mathcal{C}, Lab, Edge, Inv, Init)$*, a finite set* AP *of atomic propositions, a labeling function* $L : Loc \to 2^{AP}$ *and a TCTL formula* ψ *over* $\mathcal{C}$*.*

Let $\mathcal{LSTS} = (\Sigma, Lab', Edge', Init)$ *be the labeled state transition system induced by* $\mathcal{T}$*.*

Let $AP' = AP \cup ACC(\mathcal{T}) \cup ACC(\psi)$ *and* $L' : \Sigma \to 2^{AP'}$ *with* $L'((l, \nu)) = L(l) \cup \{g \in ACC(\mathcal{T}) \cup ACC(\psi) \mid \nu \models g\}$*.*

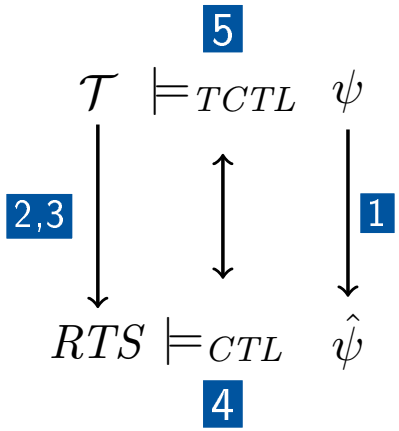
Then the clock equivalence relation $\cong \subseteq \Sigma \times \Sigma$ *for* $\mathcal{T}$ *and* ψ *is a bisimulation for* $\mathcal{LSTS}$ *and* L'*.*

TCTL model checking

Input: timed automaton $\mathcal{T}$, TCTL formula ψ

Output: the answer to the question whether $\mathcal{T} \models \psi$

1. Eliminate the timing parameters from the TCTL formula ψ, resulting in a CTL formula $\hat{\psi}$ (with clock constraints as atomic propositions).
2. Make a finite abstraction of the state space of $\mathcal{T}$.
3. Construct abstract state transition system RTS (with the states labeled by clock constraints as atomic propositions) such that $\mathcal{T} \models_{TCTL} \psi$ iff $RTS \models_{CTL} \hat{\psi}$.
4. Apply CTL model checking to check whether $RTS \models_{CTL} \hat{\psi}$.
5. Return the model checking result.



3. The abstract transition system

We have defined regions as abstract states,
now we connect them by abstract transitions.

Two kinds of transitions:
time and discrete

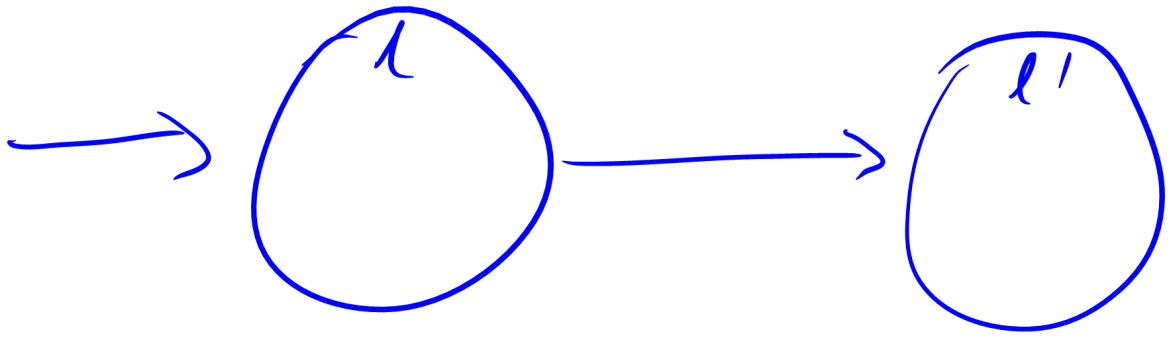
Definition

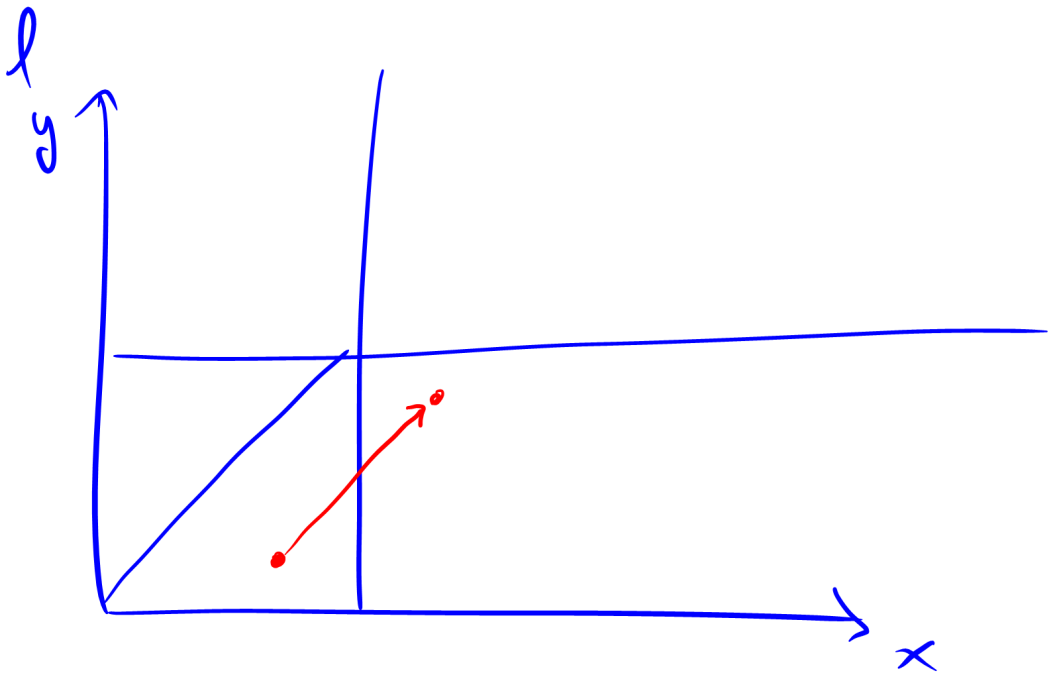
The clock region $r_\infty = \{\nu \in V \mid \forall x \in \mathcal{C}.\ \nu(x) > c_x\}$ is called unbounded. Let r, r' be two clock regions. The region r' is the successor clock region of r, denoted by $r' = succ(r)$, if either

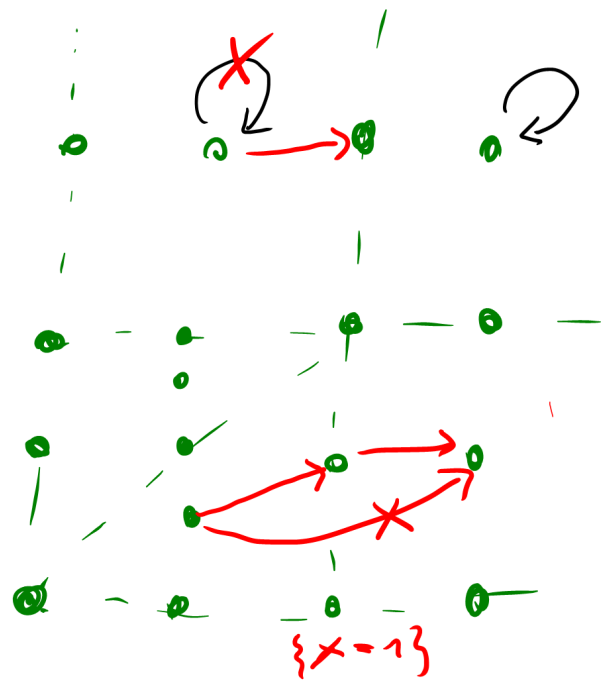
- $r = r' = r_\infty$, or
- $r \neq r_\infty$, $r \neq r'$, and for all $\nu \in r$:

$$\exists d \in \mathbb{R}_{>0}.\ (\nu + d \in r' \ \wedge\ \forall 0 \leq d' \leq d.\ \nu + d' \in (r \cup r')).$$

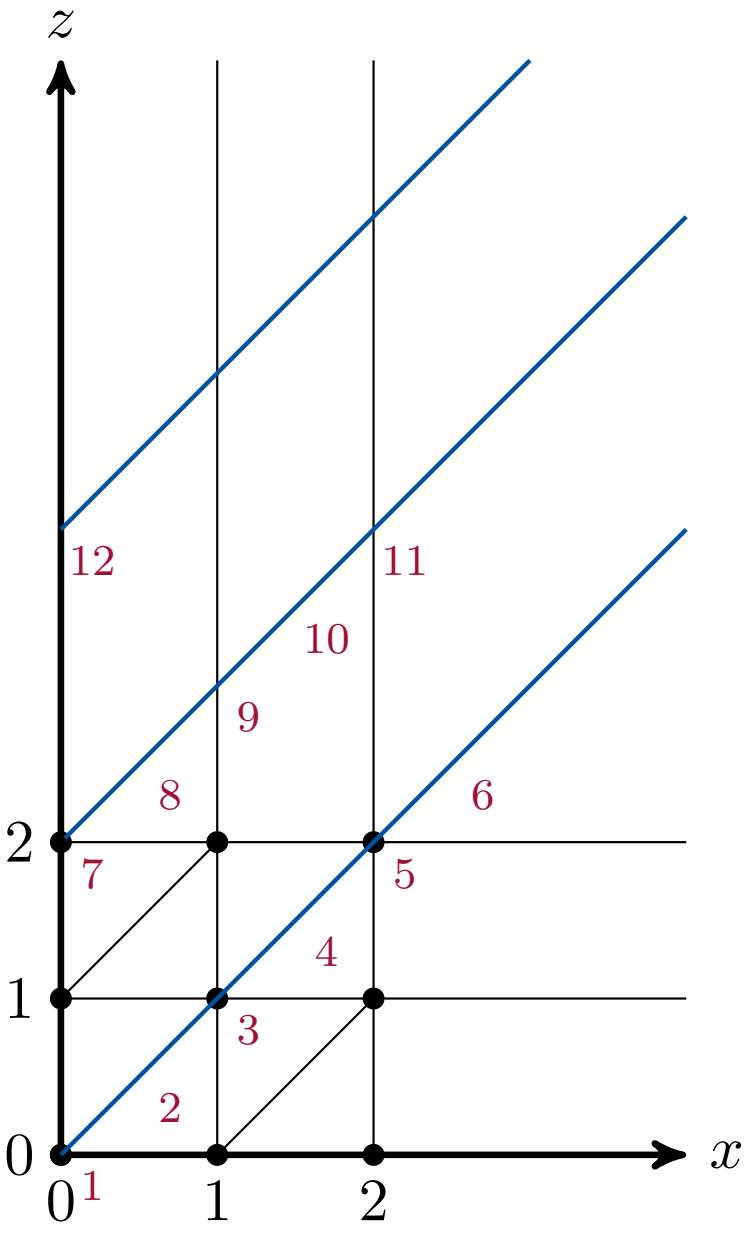
The successor state region is defined as $succ((l, r)) = (l, succ(r))$.



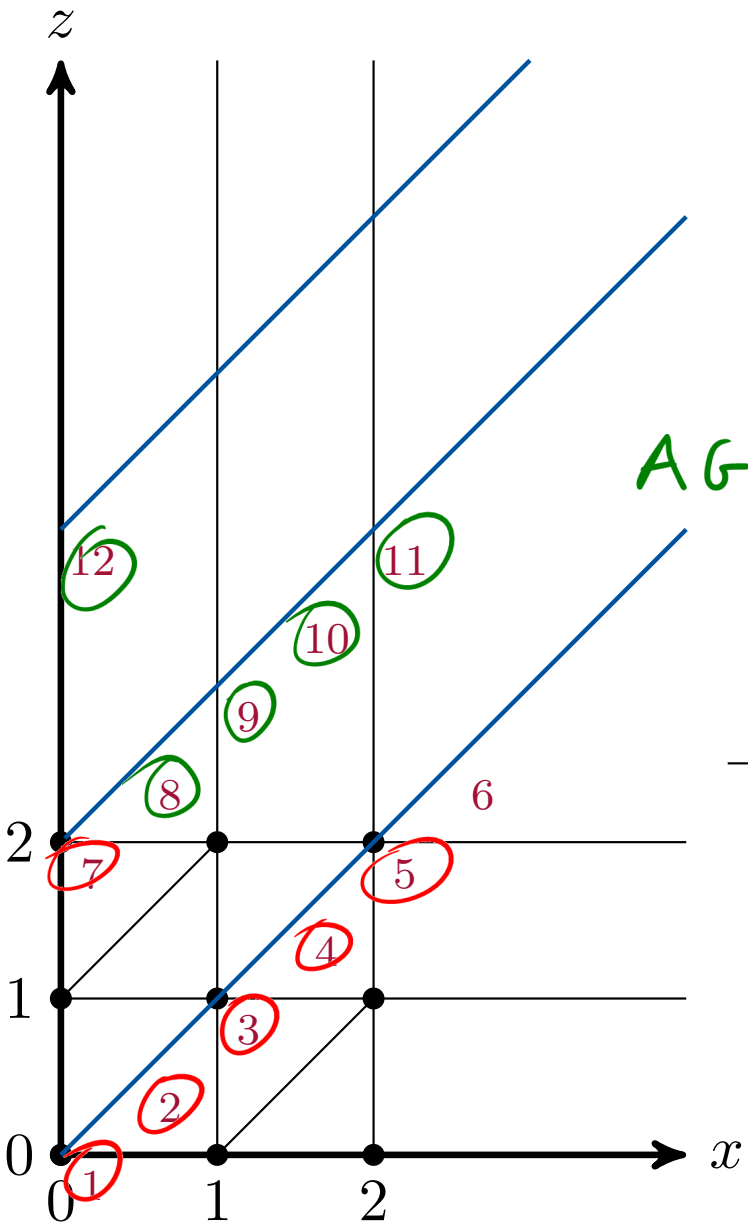




AF x=1

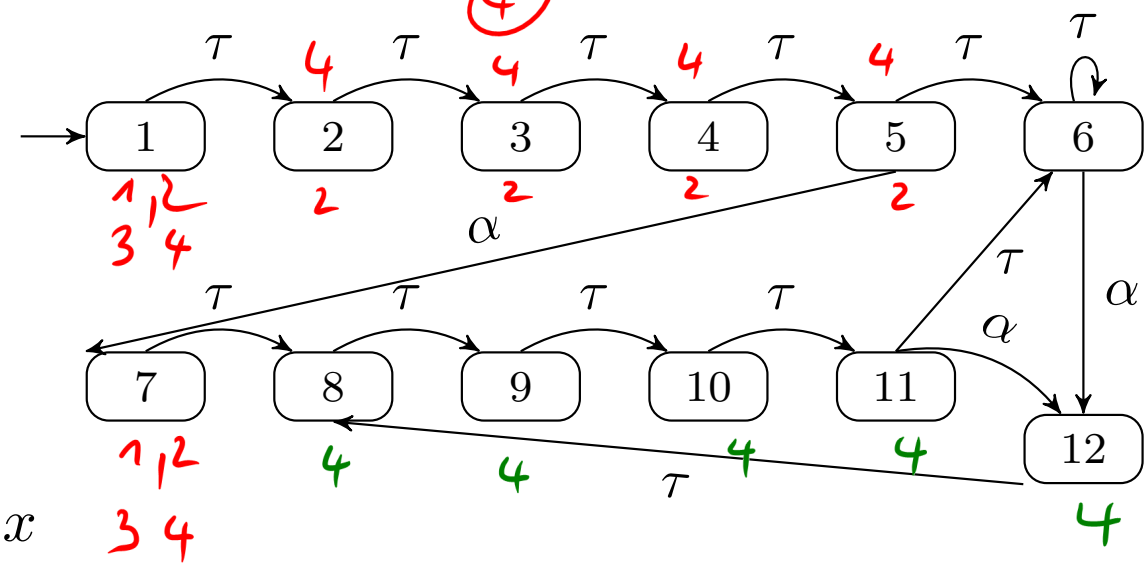


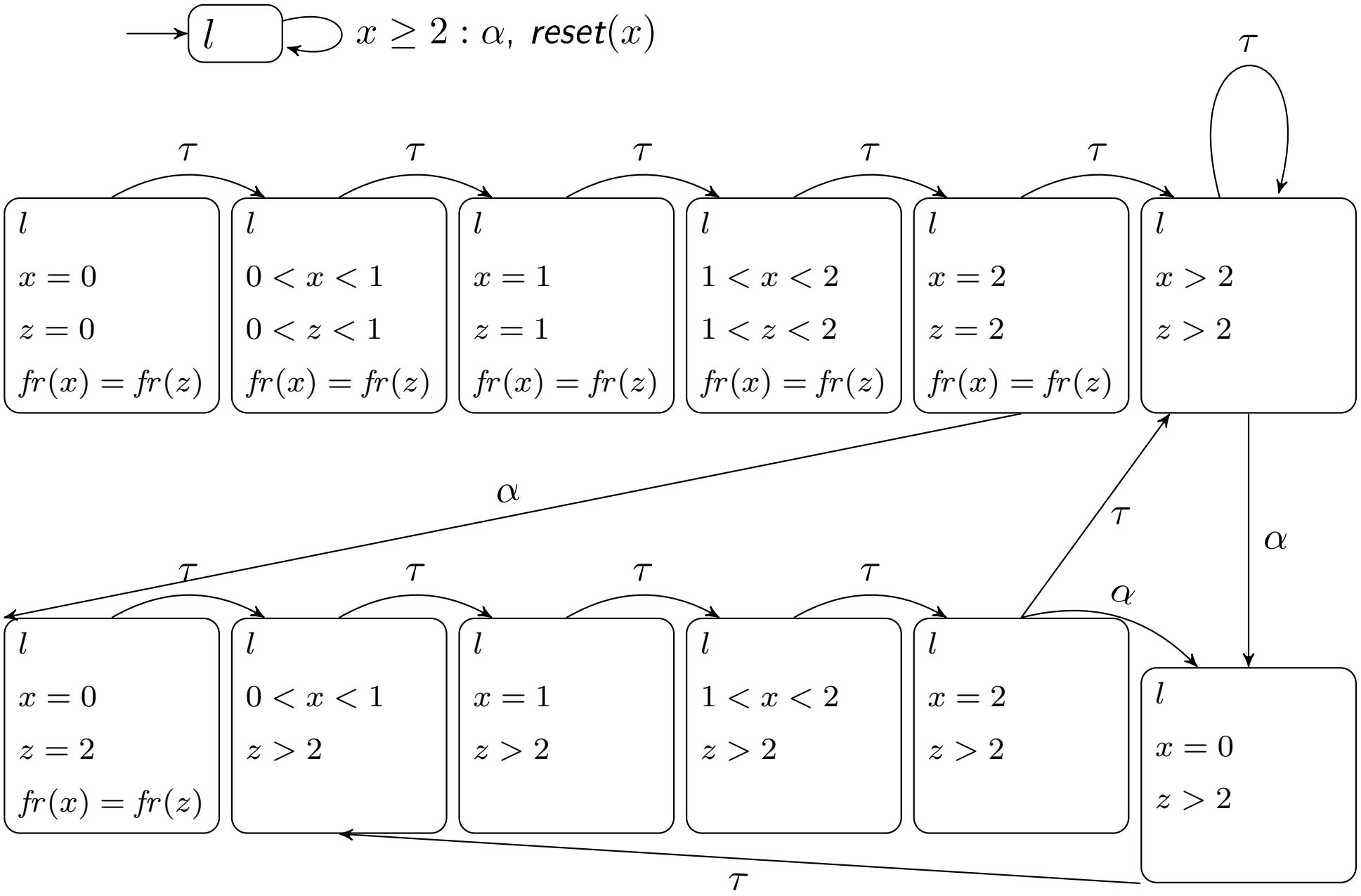
$\mathbf{E}\mathcal{F}^{(0,2]}\ (x = 0)$



$$\mathbf{E}\mathcal{F}^{(0,2]}(x=0)$$

$$AG\,EF(x=0 \wedge 0 < z \leq 2)$$




l
$x \geq 2 : \alpha,\ \mathit{reset}(x)$
τ
l $x = 0$ $z = 0$ $fr(x) = fr(z)$
l $0 < x < 1$ $0 < z < 1$ $fr(x) = fr(z)$
l $x = 1$ $z = 1$ $fr(x) = fr(z)$
l $1 < x < 2$ $1 < z < 2$ $fr(x) = fr(z)$
l $x = 2$ $z = 2$ $fr(x) = fr(z)$
l $x > 2$ $z > 2$
α
τ
α
τ
α
l $x = 0$ $z = 2$ $fr(x) = fr(z)$
l $0 < x < 1$ $z > 2$
l $x = 1$ $z > 2$
l $1 < x < 2$ $z > 2$
l $x = 2$ $z > 2$
l $x = 0$ $z > 2$
τ

Definition

Let $\mathcal{T} = (Loc, \mathcal{C}, Lab, Edge, Inv, Init)$ be a non-zeno timelock-free timed automaton with an atomic proposition set AP and a labeling function L, and let $\hat{\psi}$ be a time-unbounded TCTL formula over $\mathcal{C}$ and AP.
The region transition system of $\mathcal{T}$ for $\hat{\psi}$ is a labeled state transition system $RTS(\mathcal{T}, \psi) = (\Sigma', Lab, Edge', Init')$ with atomic propositions AP' and a labeling function L' such that

- $\Sigma' = \{(l, [\nu]) \mid (l, \nu) \in \Sigma \wedge \nu \in Inv(l)\}$
- $Init' = \{(l, [\nu]) \in \Sigma' \mid (l, \nu) \in Init\}$
- $AP' = AP \cup ACC(\mathcal{T}) \cup ACC(\hat{\psi})$
- $L'((l, [\nu])) = L(l) \cup \{g \in AP' \backslash AP \mid \nu \models g\}$

and

Definition

$$\frac{(l,\nu) \xrightarrow{a} (l',\nu')}{(l,[\nu]) \xrightarrow{a} (l',[\nu'])} \quad \texttt{Rule}_{\texttt{Discrete}}$$

$$\frac{succ(r) \models Inv(l)}{(l,r) \xrightarrow{t} (l, succ(r))} \quad \texttt{Rule}_{\texttt{Time}}$$

3. The abstract transition system

Lemma

For non-zeno timelock-free $\mathcal{T}$ and $\pi = s_0 \to s_1 \to \ldots$ an initial, infinite path of $\mathcal{T}$:

- *if π is time-convergent, then there is an index j and a state region (l, r) such that $s_i \in (l, r)$ for all $i \geq j$.*
- *if there is a state region (l, r) with $r \neq r_\infty$ and an index j such that $s_i \in (l, r)$ for all $i \geq j$ then π is time-convergent.*

Lemma

For a non-zeno timelock-free timed automaton $\mathcal{T}$ and a TCLT formula ψ:

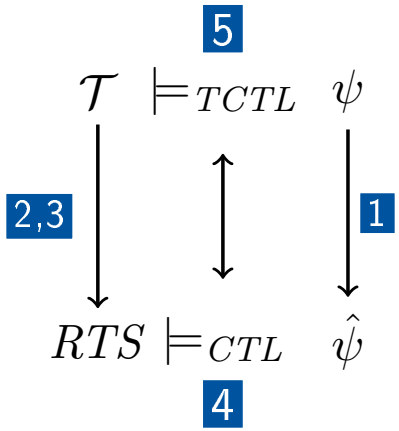
$$\mathcal{T} \models_{TCTL} \psi \quad \textit{iff} \quad RTS(\mathcal{T}, \hat{\psi}) \models_{CTL} \hat{\psi}$$

TCTL model checking

Input: timed automaton $\mathcal{T}$, TCTL formula ψ

Output: the answer to the question whether $\mathcal{T} \models \psi$

1. Eliminate the timing parameters from the TCTL formula ψ, resulting in a CTL formula $\hat{\psi}$ (with clock constraints as atomic propositions).
2. Make a finite abstraction of the state space of $\mathcal{T}$.
3. Construct abstract state transition system RTS (with the states labeled by clock constraints as atomic propositions) such that $\mathcal{T} \models_{TCTL} \psi$ iff $RTS \models_{CTL} \hat{\psi}$.
4. Apply CTL model checking to check whether $RTS \models_{CTL} \hat{\psi}$.
5. Return the model checking result.



TCTL model checking

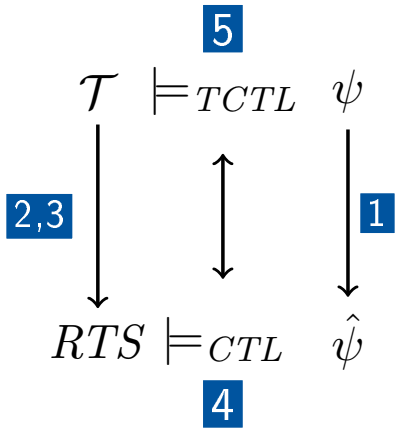
The procedure is quite similar to CTL model checking for finite automata.

One difference:

- Handling nested time bounds in TCTL formulae

TCTL model checking

Input: timed automaton $\mathcal{T}$, TCTL formula ψ

Output: the answer to the question whether $\mathcal{T} \models \psi$

1. Eliminate the timing parameters from the TCTL formula ψ, resulting in a CTL formula $\hat{\psi}$ (with clock constraints as atomic propositions).
2. Make a finite abstraction of the state space of $\mathcal{T}$.
3. Construct abstract state transition system RTS (with the states labeled by clock constraints as atomic propositions) such that $\mathcal{T} \models_{TCTL} \psi$ iff $RTS \models_{CTL} \hat{\psi}$.
4. Apply CTL model checking to check whether $RTS \models_{CTL} \hat{\psi}$.
5. Return the model checking result.