May 15, 2012

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

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

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 $\models \quad \subseteq \quad V_{\mathcal{C}} \times CC(\mathcal{C})$ 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?

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

Timed automata

A timed automaton is a special hybrid system:

- 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 explicite 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

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

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

Operational semantics

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

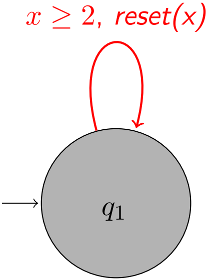
$$\frac{t > 0 \quad \nu' = \nu + t \quad \nu' \models Inv(l)}{(l,\nu) \xrightarrow{t} (l,\nu')} \quad \texttt{Rule}_{\texttt{Time}}$$

- Execution step: $\rightarrow = \xrightarrow{a} \cup \xrightarrow{t}$
- Path: $\sigma_0 \rightarrow \sigma_1 \rightarrow \sigma_2 \ldots$
- 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

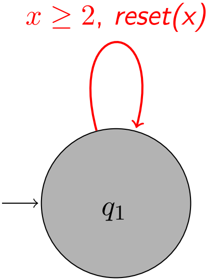
Examples:

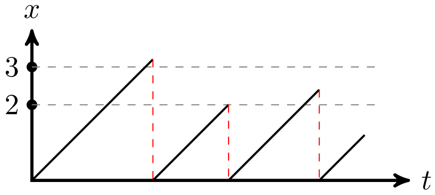
- Light switch
- Controller from the railroad crossing example
- Simplified railroad crossing
- Parallel composition for the simplified railroad crossing

Example: Timed Automaton

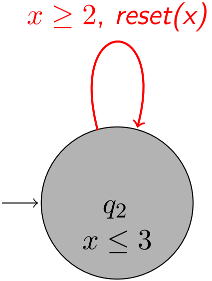


Example: Timed Automaton

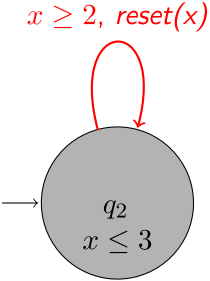


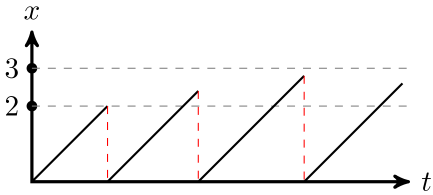


Example: Timed Automaton

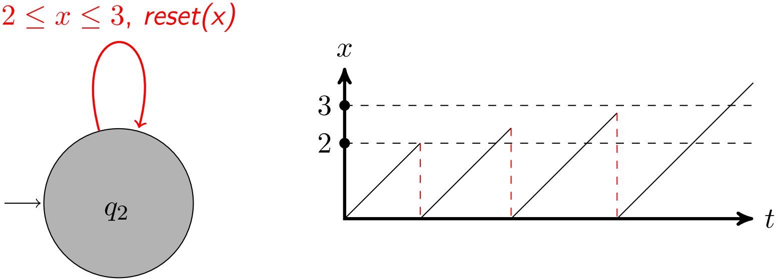


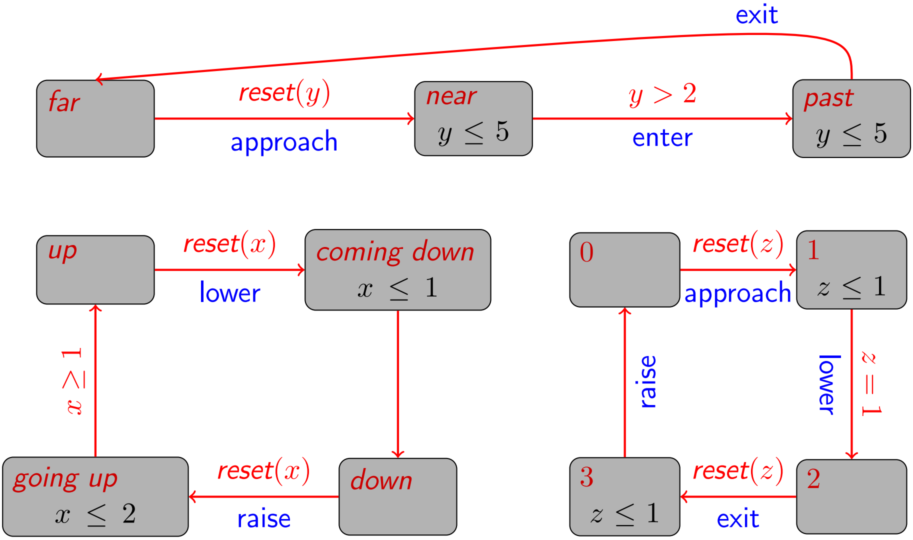
Example: Timed Automaton



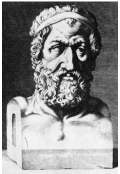


Example: Timed Automaton



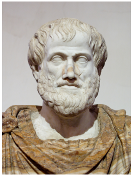

far
reset(y)
approach
near
y ≤ 5
y > 2
enter
past
y ≤ 5
exit
up
reset(x)
lower
coming down
x ≤ 1
x ≥ 1
going up
x ≤ 2
reset(x)
raise
down
0
reset(z)
approach
1
z ≤ 1
raise
lower
z = 1
3
z ≤ 1
reset(z)
exit
2

Time divergence, timelock, and zenoness



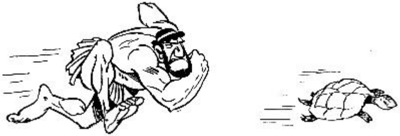
Zeno of Elea

(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.

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).

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.

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

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

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.

TCTL

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

  State formulae

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

  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 ∞).

TCTL syntax

- Syntactic sugar:

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

- Note: no next-time operator

TCTL semantics

Definition (TCTL semantics)

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

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

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 (cont.)

Definition (TCTL semantics)

For a time-divergent path $\pi = \sigma_0 \xrightarrow{\alpha_1} \sigma_1 \xrightarrow{\alpha_2} \ldots$ 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

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

- $\forall j \leq i.\ \sigma_j + d' \models \psi_1 \vee \psi_2$ 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:

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

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

TCTL 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!

TCTL model checking

Input: timed automaton $\mathcal{T}$, TCTL formula ψ
Output: the answer to the question if $\mathcal{T} \models \psi$

1. Eliminate the timing parameters from ψ, resulting in $\hat{\psi}$;
2. Make a finite abstraction of the state space
3. Construct abstract transition system RTS with $\mathcal{T} \models \psi$ iff $RTS \models \hat{\psi}$.
4. Apply CTL model checking to check whether $RTS \models \hat{\psi}$;
5. Return the model checking result.

TCTL model checking

Input: timed automaton $\mathcal{T}$, TCTL formula ψ
Output: the answer to the question if $\mathcal{T} \models \psi$

1. Eliminate the timing parameters from ψ, resulting in $\hat{\psi}$;
2. Make a finite abstraction of the state space
3. Construct abstract transition system RTS with $\mathcal{T} \models \psi$ iff $RTS \models \hat{\psi}$.
4. Apply CTL model checking to check whether $RTS \models \hat{\psi}$;
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 $\mathcal{T}' = \mathcal{T} \oplus z$ result from $\mathcal{T}$ by adding a fresh clock 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 $\mathcal{T}' = \mathcal{T} \oplus z$ result from $\mathcal{T}$ by adding a fresh clock 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 $\mathcal{T}' = \mathcal{T} \oplus z$ result from $\mathcal{T}$ by adding a fresh clock which never gets reset.

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

1. $$\begin{array}{llll} \sigma & \models_{TCTL} & \mathbf{E}(\psi_1 \; \mathcal{U}^J \; \psi_2) & \textit{iff} \\ \textit{reset}(z) \textit{ in } \sigma & \models_{TCTL} & \mathbf{E}\left((\psi_1 \vee \psi_2) \; \mathcal{U} \; ((z \in J) \wedge \psi_2)\right). & \end{array}$$

1. Eliminating timing parameters

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

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

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

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

1. Eliminating timing parameters

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

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

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

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

3. $\sigma \models_{TCTL} \mathbf{E}\mathcal{F}^{\leq 2}\psi_1$ *iff* *reset*(z) *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 $\mathcal{T}' = \mathcal{T} \oplus z$ result from $\mathcal{T}$ by adding a fresh clock which never gets reset.

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

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

2. $$\begin{array}{llllll} \sigma & \models_{TCTL} & \mathbf{A}(\psi_1 & \mathcal{U}^J & \psi_2) \textit{ iff} \\ \textit{reset}(z) \textit{ in } \sigma & \models_{TCTL} & \mathbf{A}\left((\psi_1 \vee \psi_2)\right. & \mathcal{U} & \left.((z \in J) \wedge \psi_2)\right). \end{array}$$

3. $\sigma \models_{TCTL} \mathbf{E}\mathcal{F}^{\leq 2}\psi_1$ *iff* *reset*(z) *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* *reset*(z) *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 if $\mathcal{T} \models \psi$

1. Eliminate the timing parameters from ψ, resulting in $\hat{\psi}$;
2. Make a finite abstraction of the state space
3. Construct abstract transition system RTS with $\mathcal{T} \models \psi$ iff $RTS \models \hat{\psi}$.
4. Apply CTL model checking to check whether $RTS \models \hat{\psi}$;
5. Return the model checking result.

Keywords:

Finite abstraction

Equivalence relation, equivalence classes

Bisimulation

And what does it mean in our context?

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:

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

Since the set of such (sub)formulae is finite, we strive for a finite number of equivalence classes.

Bisimulation for two LSTSs

Definition

Let $LSTS_1 = (\Sigma_1, Lab_1, Edge_1, Init_1), LSTS_2 = (\Sigma_2, Lab_2, Edge_2, Init_2)$ be two state transition systems, AP a set of atomic propositions, and $L_1 : \Sigma_1 \to 2^{AP}$ and $L_2 : \Sigma_2 \to 2^{AP}$ labeling functions over AP.
A *bisimulation* for $(LSTS_1, LSTS_2)$ is an equivalence relation $\approx \subseteq \Sigma_1 \times \Sigma_2$ such that for all $\sigma_1 \approx \sigma_2$

1. $L(\sigma_1) = L(\sigma_2)$
2. for all $\sigma_1' \in \Sigma_1$ with $\sigma_1 \xrightarrow{a} \sigma_1'$ there exists $\sigma_2' \in \Sigma_2$ such that $\sigma_2 \xrightarrow{a} \sigma_2'$ and $\sigma_1' \approx \sigma_2'$.

Bisimulation for a single LSTS

Definition

Let $LSTS = (\Sigma, Lab, Edge, Init)$ be a state transition system, AP a set of atomic propositions, and $L : \Sigma \to 2^{AP}$ a labeling function over AP.
A *bisimulation* for $LSTS$ 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 $\sigma_1' \approx \sigma_2'$.

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 $L : \Sigma \to 2^{AP}$.

A *time abstract bisimulation* on $\mathcal{T}$ 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 $\sigma_1' \approx \sigma_2'$.

Bisimulation

Lemma

Assume a timed automaton $\mathcal{T}$ with state space Σ, and a bisimulation $\approx \subseteq \Sigma \times \Sigma$ on $\mathcal{T}$.
Then for all $\sigma, \sigma' \in \Sigma$ with $\sigma \approx \sigma'$ we have that for each path

$$\pi : \sigma \stackrel{\alpha_1}{\to} \sigma_1 \stackrel{\alpha_2}{\to} \sigma_2 \stackrel{\alpha_3}{\to} \ldots$$

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

$$\pi' : \sigma' \stackrel{\alpha'_1}{\to} \sigma'_1 \stackrel{\alpha'_2}{\to} \sigma'_2 \stackrel{\alpha'_3}{\to} \ldots$$

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

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

Now, back to timed automata. How could such a bisimulation look like?

Since, in general,

- the atomic propositions assigned to and
- the paths starting at

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.
Notation:

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

For clock constraints $x < c$ with $c \in \mathbb{N}$ we have:

$$\nu \models x < c \;\Leftrightarrow\; \nu(x) < c \;\Leftrightarrow\; \lfloor \nu(x) \rfloor < c.$$

For clock constraints $x \leq c$ with $c \in \mathbb{N}$ we have:

$$\nu \models x \leq c \;\Leftrightarrow\; \nu(x) \leq c \;\Leftrightarrow\; \lfloor \nu(x) \rfloor < c \vee (\lfloor \nu(x) \rfloor = c \wedge \mathit{frac}(\nu(x)) = 0)\,.$$

I.e., only states (l, ν) and (l, ν') satisfying

$$\lfloor \nu(x) \rfloor = \lfloor \nu'(x) \rfloor \textit{ and } \mathit{frac}(\nu(x)) = 0 \textit{ iff } \mathit{frac}(\nu'(x)) = 0$$

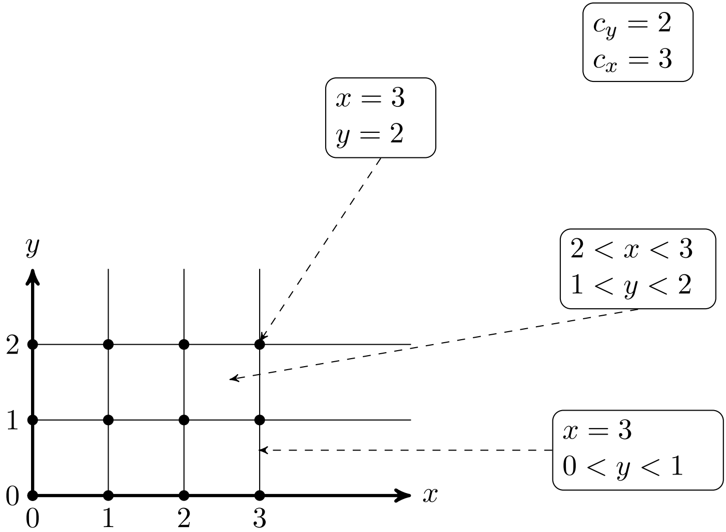
for all $x \in \mathcal{C}$ should be equivalent.

Problem: It would generate infinitely many equivalence classes!

Let c_x be the largest constant which a clock x is compared to in $\mathcal{T}$ 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

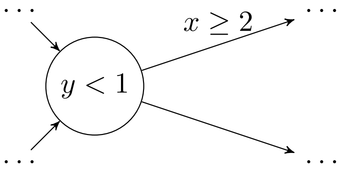
$$\begin{array}{l} (\nu(x) > c_x \wedge \nu'(x) > c_x) \quad \vee \\ \big(\lfloor \nu(x) \rfloor = \lfloor \nu'(x) \rfloor \ \wedge \ \mathit{frac}(\nu(x)) = 0 \ \mathit{iff} \ \mathit{frac}(\nu'(x)) = 0\big) \end{array}$$

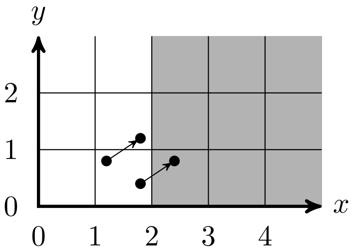
for all $x \in \mathcal{C}$ should be equivalent.


$c_y = 2$
$c_x = 3$
$x = 3$
$y = 2$
$2 < x < 3$
$1 < y < 2$
$x = 3$
$0 < y < 1$
y
x
0
1
2
0
1
2
3

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

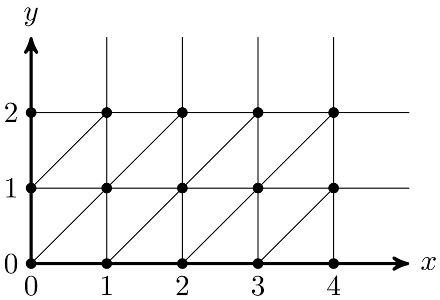
$$\begin{array}{l} (\nu(x),\nu'(x) > c_x \wedge \nu(y),\nu'(y) > c_x) \quad \vee \\ \quad (\quad \mathit{frac}(\nu(x)) < \mathit{frac}(\nu(y)) \quad \mathit{iff} \quad \mathit{frac}(\nu'(x)) < \mathit{frac}(\nu'(y)) \quad \wedge \\ \qquad \mathit{frac}(\nu(x)) = \mathit{frac}(\nu(y)) \quad \mathit{iff} \quad \mathit{frac}(\nu'(x)) = \mathit{frac}(\nu'(y)) \quad \wedge \\ \qquad \mathit{frac}(\nu(x)) > \mathit{frac}(\nu(y)) \quad \mathit{iff} \quad \mathit{frac}(\nu'(x)) > \mathit{frac}(\nu'(y))) \end{array}$$

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

$$\begin{array}{l} (\nu(x),\nu'(x) > c_x \wedge \nu(y),\nu'(y) > c_y) \quad \vee \\ \quad (\mathit{frac}(\nu(x)) \leq \mathit{frac}(\nu(y)) \quad \mathit{iff} \quad \mathit{frac}(\nu'(x)) \leq \mathit{frac}(\nu'(y))) \end{array}$$

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

2. Finite state space abstraction



$$\boxed{\begin{aligned} c_y &= 2 \\ c_x &= 4 \end{aligned}}$$

finite index

Definition

For a timed automaton $\mathcal{T}$ and a TCTL formula ψ, both over a clock set $\mathcal{C}$, we define the clock equivalence relation $\cong \subseteq \Sigma \times \Sigma$ 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

$$\lfloor \nu(x) \rfloor = \lfloor \nu'(x) \rfloor \;\wedge\; (\mathit{frac}(\nu(x)) = 0 \quad \mathit{iff} \quad \mathit{frac}(\nu'(x)) = 0)$$

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

$$\mathit{frac}(\nu(x)) \leq \mathit{frac}(\nu(y)) \quad \mathit{iff} \quad \mathit{frac}(\nu'(x)) \leq \mathit{frac}(\nu'(y)).$$

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 $[\sigma] = \{(l,\nu') \in \Sigma \mid \nu \cong \nu'\}$.

2. Finite state space abstraction

Lemma

Clock equivalence is a bisimulation over $AP' = AP \cup ACC(\mathcal{T}) \cup ACC(\psi)$.

TCTL model checking

Input: timed automaton $\mathcal{T}$, TCTL formula ψ
Output: the answer to the question if $\mathcal{T} \models \psi$

1. Eliminate the timing parameters from ψ, resulting in $\hat{\psi}$;
2. Make a finite abtraction of the state space
3. Construct abstract transition system RTS with $\mathcal{T} \models \psi$ iff $RTS \models \hat{\psi}$.
4. Apply CTL model checking to check whether $RTS \models \hat{\psi}$;
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

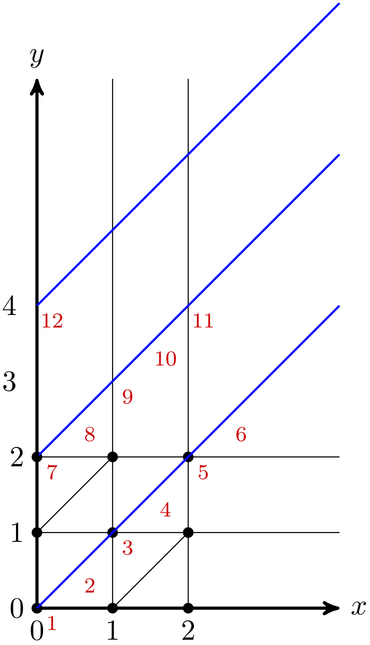
Definition

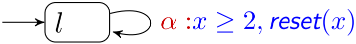
The clock region $r_\infty = \{\nu \in V \mid \forall x \in \mathcal{C}.\ \nu(x) > c_x\}$ is called unbounded. Let r, r' be two clock regions. The region r' is the successor clock region of r, denoted by $r' = succ(r)$, if either

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

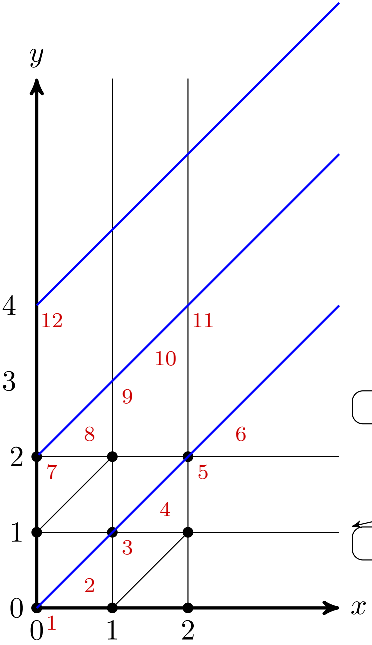
$$\exists d \in \mathbb{R}_{>0}.\ (\nu + d \in r' \ \wedge\ \forall 0 \leq d' \leq d.\ \nu + d' \in r \cup r').$$

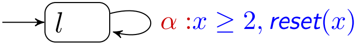
The successor state region is defined as $succ((l, r)) = (l, succ(r))$.



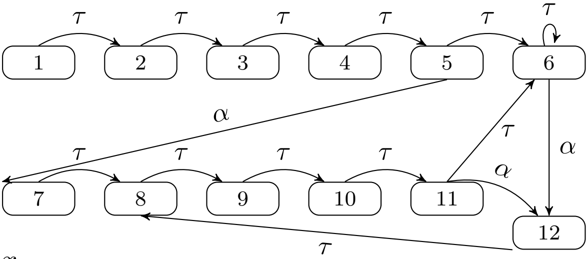


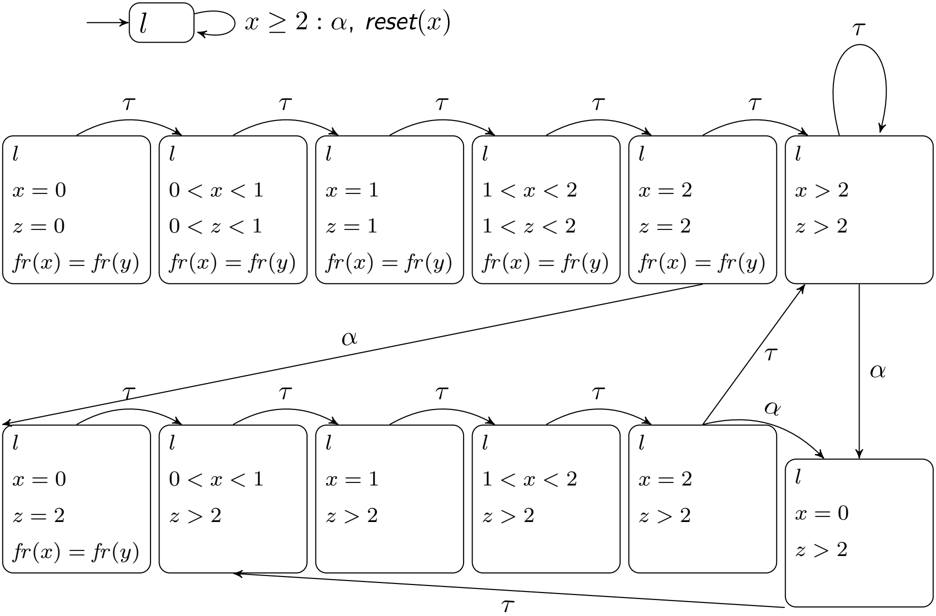
$\mathbf{E}\mathcal{F}^{(0,2]}\ x = 0$





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




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

3. The abstract transition system

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 an unbounded TCTL formula over $\mathcal{C}$ and AP.
The region transition system of $\mathcal{T}$ for $\hat{\psi}$ is a labelled state transition system $\mathcal{RTS}(\mathcal{T}, \psi) = (\Sigma', Lab', Edge', Init')$ with atomic propositions AP' and a labeling function L' such that

- Σ' the finite set of all state regions
- $Init' = \{[\sigma] \mid \sigma \in Init\}$
- $AP' = AP \cup ACC(\mathcal{T}) \cup ACC(\hat{\psi})$
- $L'((l, r)) = L(l) \cup \{g \in AP' \backslash AP \mid r \models g\}$

and

3. The abstract transition system

Definition

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

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

Lemma

For non-zeno $\mathcal{T}$ and $\pi = s_0 \rightarrow s_1 \rightarrow \ldots$ 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 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 if $\mathcal{T} \models \psi$

1. Eliminate the timing parameters from ψ, resulting in $\hat{\psi}$;
2. Make a finite abstraction of the state space
3. Construct abstract transition system RTS with $\mathcal{T} \models \psi$ iff $RTS \models \hat{\psi}$.
4. Apply CTL model checking to check whether $RTS \models \hat{\psi}$;
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 if $\mathcal{T} \models \psi$

1. Eliminate the timing parameters from ψ, resulting in $\hat{\psi}$;
2. Make a finite abstraction of the state space
3. Construct abstract transition system RTS
   $\mathcal{T} \models \psi$ iff $RTS \models \hat{\psi}$
4. Apply CTL model checking to check whether $RTS \models \hat{\psi}$;
5. Return the model checking result.

CTL model checking

Given a state transition system and a CTL formula ψ, CTL model checking labels the states recursively with the sub-formulae of ψ inside-out.

- The labeling with atomic propositions $a \in AP$ is given by a labeling function.
- Given the labelings for ψ_1 and ψ_2, we label a state with $\psi_1 \wedge \psi_2$ iff the state is labeled with both ψ_1 and ψ_2.
- Given the labeling for ψ, we label a state with $\neg\psi$ iff the state is not labeled with ψ.

CTL model checking

- Given the labeling for ψ, we label a state with $\mathbf{E}\mathcal{X}\psi$ iff there is a successor state labeled with ψ.
- Given the labeling for ψ_1 and ψ_2, we
  - label all with ψ_2 labeled states additionally with $\mathbf{E}\psi_1 \ \mathcal{U} \ \psi_2$, and
  - label all states that have the label ψ_1 and have a successor state with the label $\mathbf{E}\psi_1 \ \mathcal{U} \ \psi_2$ also with $\mathbf{E}\psi_1 \ \mathcal{U} \ \psi_2$ iteratively until a fixed point is reached.
- Given the labeling for ψ, we label a state with $\mathbf{A}\mathcal{X}\psi$ iff all successor states are labeled with ψ.
- Given the labeling for ψ_1 and ψ_2, we
  - label all with ψ_2 labeled states additionally with $\mathbf{A}\psi_1 \ \mathcal{U} \ \psi_2$, and
  - label all states that have the label ψ_1 and all of their successor states have the label $\mathbf{A}\psi_1 \ \mathcal{U} \ \psi_2$ also with $\mathbf{A}\psi_1 \ \mathcal{U} \ \psi_2$ iteratively until a fixed point is reached.