Formal techniques for software and hardware verification

Lecturers:

Vladimir Zakharov

Vladislav Podymov

e-mail:
valdus@yandex.ru

2020, fall semester

Lecture 9

Real-time systems

Timed automata (TA)

Infeasible runs of TA

Timed computational tree logic (TCTL)

Model checking problem for TCTL

Real-time systems

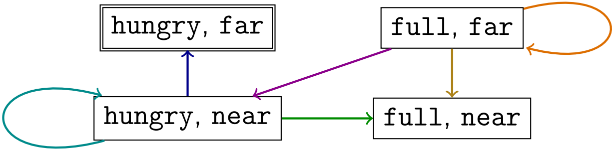
Imagine a system
consisting of a **bird** ($\mathfrak{B}$) and a mosquito **swarm** ($\mathfrak{S}$) so that :

- $\mathfrak{B}$ is (a) either `hungry` or `full`, and
  (b) either `far` from $\mathfrak{S}$, or `near` to it
- $\mathfrak{B}$ is born hungry, and far from $\mathfrak{S}$
- When $\mathfrak{B}$ is hungry, it flies close to $\mathfrak{S}$ and hunts a mosquito
- When $\mathfrak{B}$ eats a mosquito, it becomes full,
  and then eventually becomes hungry again
- When $\mathfrak{S}$ notices that $\mathfrak{B}$ is near, $\mathfrak{S}$ flies away from $\mathfrak{B}$

A "minimal" "natural" Kripke structure for the system
contains the following states:

`hungry, far`	`full, far`
`hungry, near`	`full, near`

Real-time systems



A lot of questions about system behavior immediately arise:

- Is $\mathfrak{S}$ always able to fly away from $\mathfrak{B}$?
- Is $\mathfrak{B}$ always able to eat a mosquito when it is near $\mathfrak{S}$?
- Is $\mathfrak{B}$ able to fly near $\mathfrak{S}$ being hungry indefinitely/infinitely?
- Is $\mathfrak{B}$ able to simultaneously fly close to $\mathfrak{S}$ **and** become hungry (... **and** eat a mosquito [... **and** fly away])?

The answers depend on certain abilities (speed, reaction time, ...) of $\mathfrak{B}$ and $\mathfrak{S}$

Real-time systems

Let us refine the system with some time/speed constraints:

- $\mathfrak{B}$ digests a mosquito for exactly 3 minutes
- $\mathfrak{B}$ gets close to $\mathfrak{S}$ from afar in 1 minute
- If $\mathfrak{B}$ is hungry and flying near $\mathfrak{S}$,
  then it catches a mosquito in 2 minutes
- If $\mathfrak{B}$ is flying near $\mathfrak{S}$,
  then $\mathfrak{S}$ notices $\mathfrak{B}$ after not less than 3 minutes,
  and then flies away immediately

The refied system might operate as follows:

Proximity: hungry ⇢ full ⇢ hungry ⇢ ⋯

Hunger: far ⇢ near ⇢ far ⇢ ⋯

Time: 0 1 3 4 6

Real-time systems

Let us refine the system with some time/speed constraints:

- $\mathfrak{B}$ digests a mosquito for exactly 3 minutes
- $\mathfrak{B}$ gets close to $\mathfrak{S}$ from afar in 1 minute
- If $\mathfrak{B}$ is hungry and flying near $\mathfrak{S}$,
  then it catches a mosquito in 2 minutes
- If $\mathfrak{B}$ is flying near $\mathfrak{S}$,
  then $\mathfrak{S}$ notices $\mathfrak{B}$ after not less than 3 minutes,
  and then flies away immediately

The refied system might operate as follows:

Proximity: hungry $\dashrightarrow$ full $\dashrightarrow$ hungry $\dashrightarrow \cdots$

Hunger: far $\dashrightarrow$ near $\dashrightarrow$ far $\dashrightarrow \cdots$

Time $\longrightarrow$: 0, $\frac{\pi}{6}$, $\sqrt{2}$, 4.8, $3+\sqrt{2}$

Real-time systems

Let us refine the system with some time/speed constraints:

- $\mathfrak{B}$ digests a mosquito for exactly 3 minutes
- $\mathfrak{B}$ gets close to $\mathfrak{S}$ from afar in 1 minute
- If $\mathfrak{B}$ is hungry and flying near $\mathfrak{S}$,
  then it catches a mosquito in 2 minutes
- If $\mathfrak{B}$ is flying near $\mathfrak{S}$,
  then $\mathfrak{S}$ notices $\mathfrak{B}$ after not less than 1 minute,
  and then flies away immediately

The refied system might operate as follows:

Proximity hungry $\dashrightarrow$ full $\dashrightarrow$ hungry $\dashrightarrow \cdots$

Hunger far $\dashrightarrow$ near $\dashrightarrow$ far $\dashrightarrow \cdots$

Time 0 $\frac{\pi}{6}$ $\sqrt{2}$ 4.8 $3+\sqrt{2}$ $\longrightarrow$

Real-time systems

Let us refine the system with some time/speed constraints:

- $\mathfrak{B}$ digests a mosquito for exactly 3 minutes
- $\mathfrak{B}$ gets close to $\mathfrak{S}$ from afar in 1 minute
- If $\mathfrak{B}$ is hungry and flying near $\mathfrak{S}$,
  then it catches a mosquito in 2 minutes
- If $\mathfrak{B}$ is flying near $\mathfrak{S}$,
  then $\mathfrak{S}$ notices $\mathfrak{B}$ after not less than 1 minute,
  and then flies away immediately

The refied system might operate as follows:

Proximity	hungry ⇢					…
Hunger	far ⇢	near ⇢	far ⇢	near ⇢	far ⇢	…
Time →	0	1	2	3	4	

Real-time systems

A system is said to be real-time (RTS) if
it contains real-time constraints (deadlines) for its components, and
its execution depends on whether the deadlines are met or not

For some RTSs, missed deadlines lead to unwanted consequences,
but are still acceptable:

- If I wake up too early,
  I will be drowsy, but alive
- If an important mail is late, life still goes on
- If a video frame is played late,
  the video lags, but will probably fix itself

Such RTSs are called soft

Real-time systems

A system is said to be real-time (RTS) if
it contains real-time constraints (deadlines) for its components, and
its execution depends on whether the deadlines are met or not

For some RTSs the deadlines are critical and must be held at all costs:

- If a car brakes too late, it may cost a (priceless) life
- If your parachute won't open in time, you will most likely die
- If any processing stage duration for any CPU instruction does not fit in a cycle, the whole CPU is completely useless

Such RTSs are called hard

Only **hard** RTSs are considered in this course

Real-time systems

An *execution state* of an RTS should contain a real time:

$\boxed{\text{hungry,near,2'53''}} \to$?

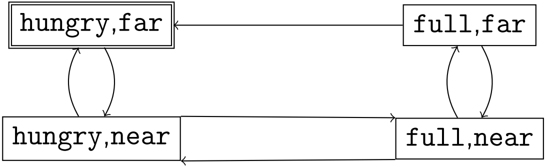
Kripke structures are not well suited to description and formal verification of systems with such timed states:
a Kripke structure is finite, and its paths are countable;
state space and runs of an RTS are usually uncountable

To overcome this problem, it is sufficient to:

1. Propose a finite description of uncountable RTSs
   and a countable description of their uncountable pathes
2. Adapt discrete temporal logic notions to real time
3. Reduce the obtained RTS verification problem
   to a known discrete verification problem (*in the next lecture*)

Timed automata by example

Back to the hungry bird:



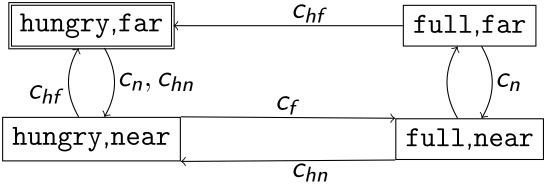
Let us try to refine the bird-swarm ($\mathfrak{B}$-$\mathfrak{S}$) model with real-time details starting with the automaton (Kripke structure) shown above

Imagine that when the automaton executes,
a collection of **stopwatches** (clocks) executes alongside:

- Clocks are constantly ticking at the same pace, starting with 0
- Any clock can be reset (set to 0) when a transition is executed to start tracking time passed since the reset

Timed automata by example

Back to the hungry bird:

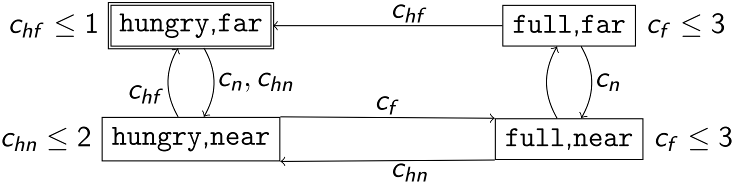


Each transition is marked with a subset of clocks to be *reset* when the transition is executed

A clock …	**tracks how much time passed since …**
c_f	$\mathfrak{B}$ became full
c_n	$\mathfrak{B}$ flew by $\mathfrak{S}$ from afar
c_{hf}	$\mathfrak{B}$ found itself hungry far from $\mathfrak{S}$
c_{hn}	$\mathfrak{B}$ found itself hungry close to $\mathfrak{S}$

Timed automata by example

Back to the hungry bird:



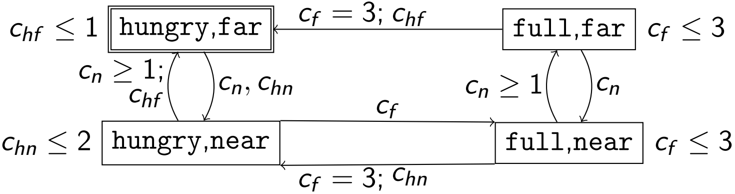
Each state is marked with clock constraints (called invariants) which must be satisfied while the automaton remains in the state

The constraints above mean that:

- $\mathfrak{B}$ cannot be full for more than 3 time units (***minutes***)
- $\mathfrak{B}$ cannot hunger afar from $\mathfrak{S}$ for more than 1 minute
- $\mathfrak{B}$ cannot hunger close to $\mathfrak{S}$ for more than 2 minutes

Timed automata by example

Back to the hungry bird:



Transitions are marked with clock constraints (called guards) which must be satisfied when the transition is being executed

The constraints above mean that:

- 3 minutes since the last meal is the only time when $\mathfrak{B}$ is able to become hungry
- To be able to fly away from $\mathfrak{B}$, $\mathfrak{S}$ must wait for at least 1 minute after $\mathfrak{B}$ flew by

When an automaton is complemented with clocks and marked with resets, invariants, and guards, it becomes a timed automaton

Timed automata

$\mathbb{N}_0$ is the set of all nonnegative integers

$\mathcal{C}$ hereafter denotes a finite set of **clocks**

Atomic clock constraints (over $\mathcal{C}$) are the following expressions:

$$\textbf{true}, \quad x < k, \quad x \leq k, \quad x - y < k, \quad x - y \leq k,$$

where $x, y \in \mathcal{C}$ and $k \in \mathbb{N}_0$

$ACC(\mathcal{C})$ is the set of all atomic clock constraints over $\mathcal{C}$

Clock constraints (over $\mathcal{C}$) are defined by the following BNF:

$$g \ ::= \ (acc) \ \mid \ (g \,\&\, g) \ \mid \ (\neg g),$$

where g is a clock constraint, and $acc \in ACC(\mathcal{C})$

Parentheses are omitted according to usual operator precedence

$CC(\mathcal{C})$ is the set of all clock constraints over $\mathcal{C}$

Timed automata

$\mathbb{R}_{\geq 0}$ is the set of all nonnegative real numbers

A clock valuation over $\mathcal{C}$ is a function

$$\nu : \mathcal{C} \to \mathbb{R}_{\geq 0}$$

A clock constraint cc is satisfied by a clock valuation ν ($\nu \models cc$) in the following cases: $(\prec \in \{<, \leq\})$

- $\nu \models \textbf{true}$
- $\nu \models x \prec k \quad \Leftrightarrow \quad \nu(x) \prec k$
- $\nu \models x - y \prec k \quad \Leftrightarrow \quad \nu(x) - \nu(y) \prec k$
- $\nu \models cc_1 \,\&\, cc_2 \quad \Leftrightarrow \quad \nu \models cc_1$ and $\nu \models cc_2$
- $\nu \models \neg cc_1 \quad \Leftrightarrow \quad \nu \not\models cc_1$

A timed constraint is called an invariant iff it does not contain differences ("$x - y$") and negations ("$\neg$")

$IC(\mathcal{C})$ is the set of all invariants over $\mathcal{C}$

Timed automata

Other relations and operations which can be used in (**non-invariant**) clock constraints:

$$
\begin{array}{rcl}
\textbf{false} & \equiv & \neg\textbf{true} \\
g_1 \vee g_2 & \equiv & \neg(g_1 \,\&\, g_2) \\
g_1 \rightarrow g_2 & \equiv & \neg g_1 \vee g_2 \\
x \geq k & \equiv & \neg(x < k) \\
x - y \geq k & \equiv & \neg(x - y < k) \\
x > k & \equiv & \neg(x \leq k) \\
x - y > k & \equiv & \neg(x - y \leq k) \\
x = k & \equiv & (x \leq k) \,\&\, (x \geq k) \\
x - y = k & \equiv & (x - y \leq k) \,\&\, (x - y \geq k) \\
x \neq k & \equiv & \neg(x = k) \\
x - y \neq k & \equiv & \neg(x - y = k)
\end{array}
$$

Timed automata

A timed automaton (TA) over a set of *atomic propositions* AP is a tuple $(L, \ell_0, \xi, \mathcal{C}, I, T)$, where:

- L is a finite set of states
- ℓ_0 is an initial state, $\ell_0 \in L$
- $\xi : L \to 2^{AP}$ is a labeling function which has the same meaning as for *Kripke structures*
- $\mathcal{C}$ is a finite set of clocks
- $I : L \to IC(\mathcal{C})$ is an invariant mapping
- $T \subseteq L \times TC(\mathcal{C}) \times 2^{\mathcal{C}} \times L$ is a transition relation
  - (l_1, g, X, l_2) is a transition from l_1 to l_2 guarded by g which resets clocks of the set X
  - graph-related representation: $l_1 \xrightarrow{g,X} l_2$

Timed automata

Note that right-hand side of atomic clock constraints ($x \prec n$; $x - y \prec n$) is a **nonnegative integer**

In particular, "$x < \sqrt{2}$" and "$x < \frac{2}{3}$" are **not** atomic clock constraints

The usual implicit assumption states that non-integer numbers in right-hand sides are excessive:

- Any real number can be approximated with a rational one with any given accuracy
- Denominators of any finite set of rational numbers can be made equal
- To eliminate a common denominator n of all rational numbers of TA, it is sufficient to divide a time unit duration by n

Timed automata

An execution configuration of a TA $A = (L, \ell_0, \xi, \mathcal{C}, I, T)$ is a pair (ℓ, ν), where $\ell \in L$, and $\nu : \mathcal{C} \to \mathbb{R}_{\geq 0}$

For clarity, sometimes TA clocks are assumed to be linearly ordered: $\mathcal{C} = (x_1, \ldots, x_m)$ — and a clock valuation ν is denoted by a tuple $(\nu(x_1), \ldots, \nu(x_m))$

An initial configuration of A is $(\ell_0, 0, 0, \ldots, 0)$

For brief definition of a TA execution step, the following denotations are needed: (*ν is a clock valuation; $d \in \mathbb{R}_{\geq 0}$; $X \subseteq \mathcal{C}$*)

- $\nu + d$ is the clock valuation defined by the identity
$$(\nu + d)(x) = \nu(x) + d$$
- $\nu[X]$ is the following clock valuation:
  - $\nu[X](x) = 0$, if $x \in X$
  - $\nu[X](y) = \nu(y)$, if $y \notin X$

Timed automata

A few more denotations: ($\sigma = (\ell, \nu)$ *is a configuration;* $d \in \mathbb{R}_{\geq 0}$*;* $X \subseteq \mathcal{C}$*;* ℓ' *is a TA state*)

- $\sigma + d = (\ell, \nu + d)$
- $\sigma[X] = (\ell, \nu[X])$
- $\sigma[\ell/\ell'] = (\ell', \nu)$

Execution steps of a TA belong to one of two classes:

1. Transition steps ($\sigma \hookrightarrow \sigma'$)
2. Delay steps ($\sigma \mapsto \sigma'$)

An execution step relation $\rightarrow$ of a TA A is a union of $\mapsto$ and $\hookrightarrow$

Hereafter, $\mathbb{R}_{>0}$ is the set of all positive real numbers

Timed automata

Let $A = (L, \ell_0, \xi, \mathcal{C}, I, T)$ be a TA, and $c = (\ell, \nu)$ a configuration

Delay step

$\sigma \overset{d}{\mapsto} \sigma'$ iff: $\quad (d \in \mathbb{R}_{>0})$

- $\sigma' = \sigma + d$
- $\nu + d \models I(\ell)$

$\sigma \mapsto \sigma'$ iff there exists d, $d \in \mathbb{R}_{>0}$, such that $\sigma \overset{d}{\mapsto} \sigma'$

Transition step

$\sigma \xhookrightarrow{\ell \xrightarrow{g,X} \ell'} \sigma'$ iff: $\quad (\ell \xrightarrow{g,X} \ell' \in T)$

- $\sigma' = \sigma[X][\ell/\ell']$
- $\nu \models g$
- $\nu[X] \models I(\ell')$

$c \hookrightarrow c'$ iff A contains a transition t such that $c \overset{t}{\hookrightarrow} c'$

Timed automata

A trace of a TA from a configuration σ (in short, a σ-trace) is a sequence of configurations of the form

$$\sigma \to \sigma_1 \to \sigma_2 \to \dots,$$

A partial run of a TA is a trace from its initial configuration

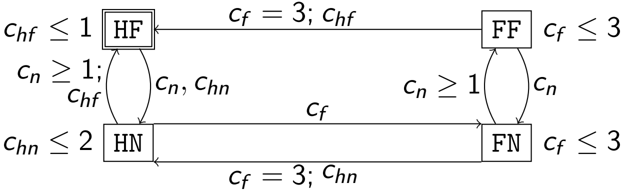
A configuration σ is a deadlock iff
there **does not** exist a configuration σ' such that $\sigma \to \sigma'$

A trace of a TA is called complete iff it is infinite or its last state is a deadlock

A run of a TA is a complete partial run

Note that these notions slightly differ
from what was introduced in the first lectures for Kripke structures

Timed automata



Example: a partial run of the TA above

(*the clocks in order:* c_f, c_n, c_{hf}, c_{hn})

$$
\begin{array}{lll}
 & (HF, 0, 0, 0, 0) & \mapsto (HF, \frac{\pi}{6}, \frac{\pi}{6}, \frac{\pi}{6}, \frac{\pi}{6}) \\
\hookrightarrow & (HN, \frac{\pi}{6}, 0, \frac{\pi}{6}, 0) & \mapsto (HN, \frac{\pi+6}{6}, 1, \frac{\pi+6}{6}, 1) \\
\mapsto & (HN, \frac{\pi+12}{6}, 2, \frac{\pi+12}{6}, 2) & \hookrightarrow (HF, \frac{\pi+12}{6}, 2, 0, 2) \\
\hookrightarrow & (HN, \frac{\pi+12}{6}, 0, 0, 0) & \hookrightarrow (FN, 0, 0, 0, 0) \\
\mapsto & (FN, \sqrt{2}, \sqrt{2}, \sqrt{2}, \sqrt{2}) & \hookrightarrow (FF, \sqrt{2}, \sqrt{2}, \sqrt{2}, \sqrt{2}) \\
\hookrightarrow & (FN, \sqrt{2}, 0, \sqrt{2}, \sqrt{2}) & \mapsto (FN, 3, 3-\sqrt{2}, 3, 3) \\
\hookrightarrow & (HN, 3, 3-\sqrt{2}, 3, 0) & \hookrightarrow (HF, 3, 3-\sqrt{2}, 0, 0)
\end{array}
$$

Infeasible runs of timed automata

A duration $delay(\sigma \to \sigma')$ of an execution step $\sigma \to \sigma'$ is a number

- d, if $\sigma \overset{d}{\mapsto} \sigma'$
- 0, if $\sigma \hookrightarrow \sigma'$

A duration of a trace $\sigma_1 \to \sigma_2 \to \sigma_3 \to \dots$ is

- the sum $\sum_{i=1}^{k} delay(\sigma_i \to \sigma_{i+1})$, if the length of the trace is finite and equals $(k+1)$
- the limit of the series $\sum_{i=1}^{\infty} delay(\sigma_i \to \sigma_{i+1})$, if the trace is infinite

Infeasible runs of timed automata

A trace is called convergent iff its duration is finite, and divergent otherwise

A zeno trace is a convergent trace which contains infinitely many transition steps

Examples

Divergent runs:

$(\ell, 0) \mapsto (\ell, 1) \mapsto (\ell, 2) \mapsto \ldots \mapsto (\ell, n) \mapsto \ldots$

$(\ell_1, 0) \mapsto (\ell_1, 1) \hookrightarrow (\ell_2, 0) \mapsto (\ell_2, 1) \hookrightarrow \ldots \mapsto (\ell_n, 1) \hookrightarrow (\ell_n, 0) \mapsto \ldots$

Convergent nonzeno runs:

$(\ell_1, 0) \mapsto (\ell_1, 1) \hookrightarrow (\ell_2, 0) \mapsto (\ell_2, 2)$ — deadlock

$(\ell, 0) \mapsto (\ell, \frac{1}{2}) \mapsto (\ell, \frac{2}{3}) \mapsto \ldots \mapsto (\ell, \frac{n-1}{n}) \mapsto \ldots$

Zeno runs:

$(\ell_1, 0) \mapsto (\ell_1, \frac{1}{2}) \hookrightarrow (\ell_2, 0) \mapsto (\ell_2, \frac{1}{4}) \hookrightarrow \ldots \hookrightarrow (\ell_n, 0) \mapsto (\ell_n, \frac{1}{2^n}) \hookrightarrow \ldots$

Infeasible runs of timed automata

A “real” RTS executes in potentially unbounded increasing time, which means that all **convergent** runs should be considered infeasible: not corresponding to any “realistic” runs of the RTS

Unfortunately, for **any** TA there exists at least one convergent run, and for any nontrivial TA — infinitely many convergent runs

Some of the convergent runs are excluded by semantics of a specification language, but some should be excluded **before** such language is picked

A TA is sound iff the following conditions are met:

- All its runs are nonzeno
- Any its partial run can be extended to a divergent run

Timed computational tree logic (TCTL): syntax

Timed computation tree logic (Timed CTL; TCTL) is a real-time analogue of *CTL*

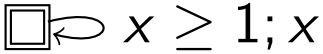
A *minimal* syntax of tctl-formulas over sets of *atomic propositions* AP and *clocks* $\mathcal{C}$ is defined by the following BNF:
$\varphi ::= p \mid (acc) \mid (\varphi \,\&\, \varphi) \mid (\neg\varphi) \mid (\mathbf{E}(\varphi\mathbf{U}\varphi)) \mid (\mathbf{A}(\varphi\mathbf{U}\varphi))$,
where φ is a tctl-formula, $p \in AP$, and $acc \in ACC(\mathcal{C})$

Informally, the letters **E**, **A**, and **U**
have a meaning similar to the same letters in CTL,
but adapted to real time execution specifics:

- $\mathbf{E}\Phi$: there exists a **divergent** run for which Φ is true
- $\mathbf{A}\Phi$: for any **divergent** run Φ is true
- $\varphi\mathbf{U}\psi$: in **real-time** future ψ eventually becomes true, and until that time φ is true

TCTL: implicit trace configurations

Consider the following *sound* TA A with clocks x, y, and tctl-formula φ:



$$\mathbf{A}(\mathbf{true}\mathbf{U}(y = 1))$$

According to the informal meaning, φ should be true for an RTS modelled by A:
"**observing any divergent run of the RTS, we eventually (*after exactly* 1 *second*) see that** $y = 1$"

The latter means that the following divergent run of the RTS should satisfy the formula "$(\mathbf{true}\mathbf{U}(y = 1))$":

$$(\ell, 0, 0) \mapsto (\ell, 2, 2) \to \ldots$$

According to the run, between the first two explicitly stated configurations the TA **implicitly** visits every configuration of the form (ℓ, d, d), where $0 < d < 2$ (continuously waiting for 2 minutes)

TCTL semantics should take into account such implicitly visited configurations

TCTL: implicit trace configurations

Consider a trace τ (of some TA):

$$\sigma_0 \to \sigma_1 \to \sigma_2 \to \dots$$

A configuration σ is generated at i-th step of τ ($i \geq 1$) iff

- $\sigma = \sigma_i$, or
- $\sigma_{i-1} \overset{d}{\longmapsto} \sigma_i$, $\sigma_{i-1} \overset{d'}{\longmapsto} \sigma$ and $d' < d$

A configuration σ is generated by the trace τ iff either $\sigma = \sigma_0$, or σ is generated at any step of τ

TCTL: semantics

Consider a TA $A = (L, \ell_0, \xi, \mathcal{C}, I, T)$ over AP, its configuration $\sigma = (\ell, \nu)$, and a tctl-formula φ over AP and $\mathcal{C}$

The formula φ is sarisfied by a configuration σ of A ($A, \sigma \models \varphi$) in the following cases:

- $A, \sigma \models a$, where $a \in AP \quad \Leftrightarrow \quad a \in \xi(\ell)$
- $A, \sigma \models acc$, where $acc \in ACC(\mathcal{C}) \quad \Leftrightarrow \quad \nu \models acc$
- $A, \sigma \models \psi \,\&\, \chi \quad \Leftrightarrow \quad A, \sigma \models \psi$ and $A, \sigma \models \chi$
- $A, \sigma \models \neg\psi \quad \Leftrightarrow \quad A, \sigma \not\models \psi$
- $A, \sigma \models \mathbf{E}\Phi \quad \Leftrightarrow$ there exists a divergent σ-trace τ of A such that $A, \tau \models \Phi$
- $A, \sigma \models \mathbf{A}\Phi \quad \Leftrightarrow$ for any divergent σ-trace τ of A if holds $A, \tau \models \Phi$

TCTL: semantics

Consider a TA $A = (L, \ell_0, \xi, \mathcal{C}, I, T)$ over AP, its configuration $\sigma = (\ell, \nu)$, and a tctl-formula φ over AP and $\mathcal{C}$

The formula φ is sarisfied by a configuration σ of A ($A, \sigma \models \varphi$) in the following cases:

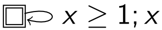
- $A, \tau \models \psi \mathbf{U} \chi$,
  where $\tau = (\sigma_0 \to \sigma_1 \to \dots)$ is a divergent trace $\Leftrightarrow$
  - $A, \sigma_0 \models \chi$, or
  - there exists a number k, $k \geq 1$, and a configuration σ generated at k-th step of τ such that
    - $A, \sigma \models \chi$, and
    - for any configuration δ generated by $\sigma_0 \to \sigma_1 \to \dots \to \sigma_k \to \sigma$ it holds $A, \delta \models \psi \vee \chi$

The formula φ is satisfied by the TA A ($A \models \varphi$) iff it is satisfied by the initial configuration of A

TCTL: “Until”

$A, \tau \models \psi \mathbf{U} \chi \Leftrightarrow$ ... for any configuration δ generated by τ it holds $A, \delta \models \psi \vee \chi$

Consider the following TA A with clocks x, y and formula φ:



$$\mathbf{A}((y \leq 1)\mathbf{U}(y > 1))$$

The following holds: $A, (\ell, 0, 0) \models \varphi$ —
and it complies with the informal meaning of φ:
“**there exists a divergent run of the RTS such that its duration will eventually exceed 1, and until that time the duration will be not greater than 1**”

Note the formula “$\psi \vee \chi$” in the definition of “$\mathbf{U}$” for TCTL: in the corresponding place of the CTL* definition “ψ” is written instead

This difference is

- insignificant: $\psi \mathbf{U} \chi \equiv (\psi \vee \chi)\mathbf{U}\chi$ in CTL*
- necessary: if “$\psi \vee \chi$” is replaced with “ψ”, then $A, (\ell, 0, 0) \not\models \varphi$, which would be inadequate

TCTL: other operations

Other *familiar* boolean connectives and temporal operations are usually included in TCTL:

$$\varphi \vee \psi \equiv \neg(\neg\varphi \,\&\, \neg\psi) \qquad \varphi \to \psi \equiv \neg\varphi \vee \psi$$

$$\mathbf{E}(\varphi\mathbf{R}\psi) \equiv \neg\mathbf{E}(\neg\varphi\mathbf{U}\neg\psi) \qquad \mathbf{A}(\varphi\mathbf{R}\psi) \equiv \neg\mathbf{A}(\neg\varphi\mathbf{U}\neg\psi)$$

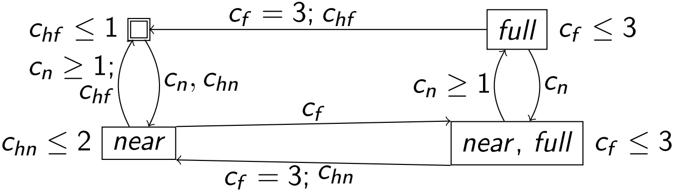
$$\mathbf{EF}\varphi \equiv \mathbf{E}(\mathbf{true}\mathbf{U}\varphi) \qquad \mathbf{AF}\varphi \equiv \mathbf{A}(\mathbf{true}\mathbf{U}\varphi)$$

$$\mathbf{AG}\varphi \equiv \neg\mathbf{EF}\neg\varphi \qquad \mathbf{EG}\varphi \equiv \neg\mathbf{AF}\neg\varphi$$

Informally, **F**, **G**, and **R** in TCTL differ from the same letters in CTL in the same way as the letter **U**:

- $\mathbf{F}\varphi$: in **real-time** future φ eventually becomes true
- $\mathbf{G}\varphi$: in **real-time** future φ is always true
- $\varphi\mathbf{R}\psi$: ... (*try it yourself*)

TCTL: other operations



A few more tctl-properties:

- No matter what happens, if $\mathfrak{B}$ is hungry, then it has a chance to eat

$$\mathbf{AG}(\neg full \to \mathbf{EF} full)$$

- $\mathfrak{B}$ cannot fly hungry and afar from $\mathfrak{S}$ for more than a minute

$$\neg\mathbf{EF}(\neg full \,\&\, \neg near \,\&\, c_{hf} > 1)$$

- In 2 minutes after $\mathfrak{B}$ finds itself hungry and near $\mathfrak{S}$ it is decided whether the hunt is successfull or not

$$\mathbf{AG}(c_{hn} = 0 \to \mathbf{AF}(c_{hn} \leq 2 \,\&(far \vee full)))$$

Model checking problem for TCTL

Given a sound timed automaton A
and a tctl-formula φ,
check whether the relation $A \models \varphi$ holds