Quantitative verification
Chapter 2: Timed automata

Jan Křetínský

Technical University of Munich

Winter 2017/18
Mathematically rigorous techniques and tools for

- specification
- design
- verification

of software and hardware systems.

**Definition 1**
Formal verification is the act of proving or disproving the correctness of a system with respect to a certain formal specification or property.
Formal Verification Techniques

- **manual** – human tries to produce a proof of correctness
- **semi-automatic** – theorem proving
- **automatic** – algorithm takes a model and a property; decides whether the model satisfies the property

We focus on automatic techniques.
Application Domains of FV

- generally safety-critical systems: a system whose failure can cause death, injury, or big financial loses (e.g., aircraft, nuclear station)
- particularly embedded systems
  - often safety critical
  - reasonably small and thus amenable to formal verification
Model Checking

- automatic verification technique
- user produces:
  - a model of a system
  - a logical formula which describes the desired properties
- model checking algorithm:
  - checks if the model satisfies the formula
  - if the property is not satisfied, a counterexample is produced
State Space

- model checking algorithms are based on state space exploration, i.e., "brute force"
- state space describes all possible behaviors of the model
- state space  \( \approx \) graph:
  - nodes = states of the system
  - edges = transitions of the system
- in order to construct state space, the model must be closed, i.e., we need to model environment of the system
Example: Model and State Space
Example: Peterson’s Algorithm

- flag[0], flag[1] (initialized to false) – meaning want to access CS
- turn (initialized to 1) – used to resolve conflict

Process 0:
while (true) {
  <noncritical section>;
  flag[0] := true;
  turn := 1;
  while flag[1] and turn = 1 do {};
  <critical section>;
  flag[0] := false;
}

Process 1:
while (true) {
  <noncritical section>;
  flag[1] := true;
  turn := 2;
  while flag[0] and turn = 2 do {};
  <critical section>;
  flag[1] := false;
}
Example: Peterson’s Algorithm

desired property: always, at most one process in CS
desired property: always, at most one process in CS: $G(\neg ([3][3]))$
Model Checking: Steps

1. **modeling**: system $\rightarrow$ model
2. **specification**: natural language $\rightarrow$ property
3. **verification**: algorithm for checking whether a model satisfies the property

For real-time systems:
- modeling formalism: timed automata
- specification formalism: reachability, (timed logics)
Fischer’s Protocol

- real-time mutual exclusion protocol – correctness depends on timing assumptions
- simple, just 1 shared variable, arbitrary number of processes
- assumptions: known upper bound $D$ for the time between successive steps of the execution of a process while it attempts to access its critical section
- each process has its own timer (for delaying)
Fischer’s protocol

- id
  shared variable, initialized by -1
- each process has its own timer (for delaying)
- for correctness it is necessary that $K > D$

Process i:
while (true) {
  <noncritical section>;
  while id != -1 do {};
  id := i;
  delay K;
  if (id = i) {
    <critical section>;
    id := -1;
  }
}
Modeling Fischer’s Protocol

- How to model clocks?
- How to model waiting (delay)?
Two possible models of time:

- discrete time domain
- continuous time domain
Discrete Time Domain

- clocks tick at regular interval
- at each tick, something may happen
- between ticks – the system only waits

![Discrete Time Domain Diagram]

- choose a fixed sample period $\varepsilon$
- all events happen at multiples of $\varepsilon$
- simple extension of classical model (time = new integer variable)
- main disadvantage – how to choose $\varepsilon$?
  - big $\varepsilon \Rightarrow$ too coarse model
  - small $\varepsilon \Rightarrow$ time fragmentation, too big state space
- usage: particularly synchronous systems (hardware circuits)
Continuous Time Domain

- time \(\approx\) real number
- delays may be arbitrarily small
- more faithful model, suitable for asynchronous systems
- model checking \(\approx\) traversal of state space
  - **Problem:** uncountable state space \(\Rightarrow\) cannot be directly handled by "brute force"
Timed Automata

- extension of finite state machines with clocks
- continuous real time semantics
- limited list of operations over clocks $\Rightarrow$ automatic verification feasible
- allowed operations:
  - comparison of a clock with a constant
  - reset of a clock
  - uniform flow of time (all clocks have the same rate)
What is a Timed Automaton?

- an automaton with locations (states) and edges
- the automaton spends time only in locations, not in edges
What is a Timed Automaton?

- real valued clocks
- all clocks run at the **same speed**
- clock constraints **guard** the edges
What is a Timed Automaton?

- clocks can be reset when taking an edge
- only a reset to value 0 is allowed
What is a Timed Automaton?

- Location invariants forbid to stay in a state too long
- Invariants must be satisfied ⇒ force taking an edge

We also add labels to edges to allow definition of languages, behavioral equivalences, etc.
Definition 2

Let $\mathcal{C}$ be a set of clocks. Then the set $\mathcal{B}(\mathcal{C})$ of clock constraints is defined by the following abstract syntax

$$g ::= x \bowtie k \mid g \land g$$

where $x \in \mathcal{C}$, $k \in \mathbb{N}$ and $\bowtie \in \{\leq, <, =, >, \geq\}$. 
Timed Automata

Let $C$ be a set of clocks and let $\Sigma$ be a finite set of actions.

**Definition 3**

A *timed automaton* is a 4-tuple: $A = (L, \ell_0, E, I)$

- $L$ is a finite set of *locations*
- $\ell_0 \in L$ is an *initial location*
- $E \subseteq L \times \mathcal{B}(C) \times \Sigma \times 2^C \times L$ is a finite set of *edges*
- $I : L \to \mathcal{B}(C)$ assigns *invariants* to locations

An edge is $(s, c, a, S, t)$, where $s$ is the source location, $c$ is the clock constraint, $a$ is the action, $S$ is the set of clocks to be reset, and $t$ is the target location.

We omit the actions from edges if either $\Sigma$ is a singleton set or the actions are not relevant (e.g. for reachability).
Semantics: Main Idea

- semantics is a transition system (states & transitions)
- states given by:
  - location (local state of the automaton)
  - clock valuation
- transitions:
  - delay – only clock valuation changes
  - action – change of location
Clock Valuations

- A **clock valuation** is a function $\nu : C \rightarrow \mathbb{R}^+$.
- Given a set of clocks $Y \subseteq C$, denote by $\nu[Y := 0]$ the valuation obtained from $\nu$ by resetting clocks from $Y$:
  
  $$\nu[Y := 0](x) = \begin{cases} 0 & x \in Y \\ x & \text{otherwise.} \end{cases}$$

- $\nu + d \approx$ flow of time (by $d$ units):
  
  $$(\nu + d)(x) = \nu(x) + d$$

- $\nu \models g$ means that the valuation $\nu$ satisfies the constraint $g$
  
  - $\nu \models x \gg k$ iff $\nu(x) \gg k$
  - $\nu \models g_1 \land g_2$ iff $\nu \models g_1$ and $\nu \models g_2$
Examples

let \( \nu = (x \rightarrow 3, y \rightarrow 2.4, z \rightarrow 0.5) \)

- what is \( \nu[y := 0] \) (usually written as \( \nu[y := 0] \)) ?
- what is \( \nu + 1.2 \) ?
- does \( \nu \models y < 3 \) ?
- does \( \nu \models x < 4 \land z \geq 1 \) ?
Definition 4
The semantics of a timed automaton $A$ is a (labeled) transition system $S_A = (S, s_0, \rightarrow)$

- $S = L \times (C \rightarrow \mathbb{R}^+)$
- $s_0 = (\ell_0, \nu_0)$ where $\nu_0(x) = 0$ for all $x \in C$
- transitions are defined by

delay: $(\ell, \nu) \xrightarrow{\delta} (\ell, \nu + \delta)$ for all $\delta \in \mathbb{R}^+$ such that
- $\nu \models I(\ell)$
- $\nu + \delta' \models I(\ell)$ for all $0 \leq \delta' \leq \delta$

action: $(\ell, \nu) \xrightarrow{a} (\ell', \nu')$ iff $(\ell, g, a, Y, \ell') \in E$ where
- $\nu \models g$
- $\nu' = \nu[Y := 0]$
- $\nu' \models I(\ell')$

We write $(\ell, \nu) \rightarrow (\ell', \nu')$ iff $(\ell, \nu) \xrightarrow{h} (\ell', \nu')$ where $h \in \Sigma \cup \mathbb{R}^{\geq 0}$
Example

- What is a clock valuation?
- What is a state?
Example

- What is a clock valuation?
- What is a state?

- clock valuation: assignment of a real value to $x$
- initial state $(\text{off}, 0)$; another state e.g. $(\text{light}, 1.4)$
the semantics is infinite state (even uncountable)
the semantics is even infinitely branching

Investigated areas:

- languages – emptiness, universality, language inclusion (undecidable),...
- equivalence checking – bisimulation of timed automata (timed and untimed), simulation, ...
- verification – reachability, (timed) temporal logics, ...

Reachability Problem

A run is a maximal sequence (i.e. the one that cannot be prolonged) of the form $(\ell_0, \nu_0) \rightarrow (\ell_1, \nu_1) \rightarrow \cdots$

**Definition 5**

**Input:** a timed automaton $A$, a location $\ell$ of the automaton  
**Question:** Does there exist a run of $A$ which reaches $\ell$?

This problem formalizes the verification of safety problems – is an erroneous state reachable?
Reachability: Attempt 1

- discretization (sampled semantics)
- allow time step (delay) 1
- clock above maximal constant $\Rightarrow$ value does not increase
- finite state space
Reachability: Attempt 1

- discretization (sampled semantics)
- allow time step (delay) 1
- clock above maximal constant \(\Rightarrow\) value does not increase
- finite state space
- not equivalent \(\Rightarrow\) find a counterexample
Reachability: Attempt 2

- what about time step 0.5?
Reachability: Attempt 2

- what about time step 0.5?
Reachability: Attempt 2

- what about time step 0.5?

- what about time step 0.25?
- what about time step $2^{-n}$?
for each automaton there exists $\varepsilon$ such that sampled semantics with time step $\varepsilon$ and dense semantics are equivalent w.r.t. reachability

no fixed $\varepsilon$ is sufficient for all timed automata

for more complex verification problems sampled and dense semantics are not equivalent
Another approach?

- Idea: is it necessary to distinguish the following valuations? \((0.589, 1.234)\) and \((0.587, 1.235)\)
- some clock valuations are equivalent as the automaton cannot distinguish between them w.r.t. reachable locations
- let us find such equivalence classes (so called regions)


**Theorem 6**

*The reachability problem is in \( \text{PSPACE} \).*

- note that even decidability is not straightforward – the semantics is infinite state
- **decidability** proved by region construction (to be discussed)
- completeness proved by general reduction from linearly bounded Turing machines (not discussed)
Main idea:

- define equivalence \( \equiv \) on valuations so that if \( \nu \equiv \mu \) then the automaton “cannot distinguish between \((\ell, \nu)\) and \((\ell, \mu)\)”

In particular, both configurations \((\ell, \nu)\) and \((\ell', \mu')\) can reach the same set of locations (Note that this equivalence is basically a bisimulation)

- work with regions, i.e., equivalence classes of valuations, instead of valuations
- finite number of regions

What conditions on \( \equiv \) do we need?
Main idea:

- define equivalence ≡ on valuations so that if \( \nu \equiv \mu \) then the automaton “cannot distinguish between \((\ell, \nu)\) and \((\ell, \mu)\)”
- define ≡ so that \( \nu \equiv \mu \) implies that for every \( \ell \)
  - if \((\ell, \nu) \rightarrow (\ell', \nu')\) then \((\ell, \mu) \rightarrow (\ell', \mu')\) so that \(\nu' \equiv \mu'\)
  - if \((\ell, \mu) \rightarrow (\ell', \mu')\) then \((\ell, \nu) \rightarrow (\ell', \nu')\) so that \(\nu' \equiv \mu'\)

In particular, both configurations \((\ell, \nu)\) and \((\ell', \mu')\) can reach the same set of locations

(Note that this equivalence is basically a bisimulation)
Main idea:

- define equivalence $\equiv$ on valuations so that if $\nu \equiv \mu$ then the automaton “cannot distinguish between $(\ell, \nu)$ and $(\ell, \mu)$”
- define $\equiv$ so that $\nu \equiv \mu$ implies that for every $\ell$
  - if $(\ell, \nu) \rightarrow (\ell', \nu')$ then $(\ell, \mu) \rightarrow (\ell', \mu')$ so that $\nu' \equiv \mu'$
  - if $(\ell, \mu) \rightarrow (\ell', \mu')$ then $(\ell, \nu) \rightarrow (\ell', \nu')$ so that $\nu' \equiv \mu'$

In particular, both configurations $(\ell, \nu)$ and $(\ell', \mu')$ can reach the same set of locations
(Note that this equivalence is basically a bisimulation)

- work with regions, i.e., equivalence classes of valuations, instead of valuations
- finite number of regions

What conditions on $\equiv$ do we need?
Let $d \in \mathbb{R}^{\geq 0}$. Define

- $\lfloor d \rfloor$ to be the integer part of $d$
- $fr(d)$ to be the fractional part of $d$

Thus $d = \lfloor d \rfloor + fr(d)$

Example: $\lfloor 42.37 \rfloor = 42$, $fr(42.37) = 0.37$
Equivalence on Clock Valuation: Condition 1

Let $c_x$ be the largest constant compared to a clock $x$ ("max bound")

Two valuations $\nu$ and $\mu$ are equivalent, $\nu \cong \mu$ iff the following conditions are satisfied:

C1 Clock $x$ is in both valuations $\nu$ and $\mu$ above its max bound, or it has the same integer part in both of them:

$$\nu(x) \geq c_x \land \mu(x) \geq c_x \quad \text{or} \quad \lfloor \nu(x) \rfloor = \lfloor \mu(x) \rfloor$$
Equivalence on Clock Valuation: Condition 1

Let $c_x$ be the largest constant compared to a clock $x$ ("max bound")

Two valuations $\nu$ and $\mu$ are equivalent, $\nu \equiv \mu$ iff the following conditions are satisfied:

**C1** Clock $x$ is in both valuations $\nu$ and $\mu$ above its max bound, or it has the same integer part in both of them:

$$\nu(x) \geq c_x \land \mu(x) \geq c_x \quad \text{or} \quad \lfloor \nu(x) \rfloor = \lfloor \mu(x) \rfloor$$

**C2** If the value of clock is below its max bound, then either it has zero fractional part in both $\nu$ and $\mu$ or in none of them:

$$\nu(x) \leq c_x \quad \Rightarrow \quad (fr(\nu(x)) = 0 \iff fr(\mu(x)) = 0)$$
Equivalence on Clock Valuation: Condition 1

Let $c_x$ be the largest constant compared to a clock $x$ ("max bound")

Two valuations $\nu$ and $\mu$ are equivalent, $\nu \cong \mu$ iff the following conditions are satisfied:

C1 Clock $x$ is in both valuations $\nu$ and $\mu$ above its max bound, or it has the same integer part in both of them:

$$\nu(x) \geq c_x \land \mu(x) \geq c_x \quad \text{or} \quad \lfloor \nu(x) \rfloor = \lfloor \mu(x) \rfloor$$

C2 If the value of clock is below its max bound, then either it has zero fractional part in both $\nu$ and $\mu$ or in none of them:

$$\nu(x) \leq c_x \quad \Rightarrow \quad (fr(\nu(x)) = 0 \iff fr(\mu(x)) = 0)$$

C3 For two clocks that are below their max bound, ordering of fractional parts must be the same in both $\nu$ and $\mu$:

$$\nu(x) \leq c_x \land \mu(x) \leq c_y \quad \Rightarrow \quad (fr(\nu(x)) \leq fr(\nu(y)) \iff fr(\mu(x)) \leq fr(\mu(y)))$$
Identify $c_x$ and $c_y$

suppose $c_x = 4$, $c_y = 5$, $c_z = 1$

- let $(x, y, z)$ denote valuations, decide:
  1. $(0, 0.14, 0.3) \equiv (0.05, 0.1, 0.32)$ ?
  2. $(1.9, 4.2, 0.4) \equiv (2.8, 4.3, 0.7)$ ?
  3. $(0.05, 0.1, 0.3) \equiv (0.2, 0.1, 0.4)$ ?
  4. $(0.03, 1.1, 0.3) \equiv (0.05, 1.2, 0.3)$ ?
Regions

Definition 7
Classes of equivalence \( \equiv \) are called regions, denoted by \([\nu]\).

Example:
- Suppose TA with two clocks, \(c_x = 3, c_y = 2\)
- Draw all regions (since we have just 2 clocks, we can draw them in plane)
Definition 7
Classes of equivalence \( \equiv \) are called regions, denoted by \([\nu]\).

Example:

- Suppose TA with two clocks, \( c_x = 3, c_y = 2 \)
- Draw all regions (since we have just 2 clocks, we can draw them in plane)
Lemma 8
\( \nu \cong \mu \) implies that for every \( \ell \)

- if \( (\ell, \nu) \rightarrow (\ell', \nu') \) then \( (\ell, \mu) \rightarrow (\ell', \mu') \) so that \( \nu' \cong \mu' \)
- if \( (\ell, \mu) \rightarrow (\ell', \mu') \) then \( (\ell, \nu) \rightarrow (\ell', \nu') \) so that \( \nu' \cong \mu' \)

Lemma 9
The number of regions is at most \(|C|! \cdot 2^{|C|} \cdot \prod_{x \in C}(2c_x + 2)\).
A region graph is a (labeled) transition system where
- states are pairs of the form \((\ell, [\nu])\) where \(\ell\) is a location and \(\nu\) is a valuation
- transitions are defined by

\[
(\ell, [\nu]) \rightarrow (\ell', [\nu']) \quad \text{iff} \quad (\ell, \nu) \rightarrow (\ell', \nu')
\]

- region graph is equivalent to the semantics of \(A\) w.r.t. reachability, i.e., a location \(\ell\) is reachable in the region graph iff it is reachable in the semantics of \(A\)
- moreover, region graph is finite and can be effectively constructed \(\Rightarrow\) region graph can be used to solve the reachability problem
Operations on Regions

To construct the region graph, we need the following operations:

- let time pass – go to adjacent region at top right
- intersect with a clock constraint (note that clock constraints define supersets of regions)
  - if region is in the constraint: no change
  - otherwise: empty
- reset a clock – go to a corresponding region
Example: Timed Automaton
Example: Region Graph

Here transitions that do not change location have been compressed (i.e. each transition in the above graph consists of an arbitrary number of delay transitions succeeded by one action transition)
Regions – impractical, too many regions constructed explicitly – no on-the-fly approach

Definition 10
Denote by $\mathcal{B}^+(C)$ the set of extended clock constraints defined by

$$\psi ::= x \preceq k \mid x - y \preceq k \mid \phi \land \phi$$

where $x, y \in C$, $k \in \mathbb{N}$ and $\preceq \in \{\leq, <, =, >, \geq\}$.

Definition 11
A zone is a set of clock valuations described by an extended clock constraint $g_z \in \mathcal{B}^+(C)$:

$$Z = \{\nu \mid \nu \models g_z\}$$

A symbolic state is a pair $(\ell, Z)$ where $\ell$ is a location and $Z$ a zone.
Zone Operations & Symbolic Transitions

- $Z^\uparrow = \{ \nu + \delta \mid \nu \in Z \land \delta \in \mathbb{R}_{\geq 0} \}$
- $Z[Y := 0] = \{ \nu[Y := 0] \mid \nu \in Z \}$

**Lemma 12**

*If $Z$ is a zone, then both $Z^\uparrow$ and $Z[Y := 0]$ are zones.*

Symbolic transition relation $\rightsquigarrow$ over symbolic states:

- $(\ell, Z) \rightsquigarrow (\ell, Z^\uparrow \land I(\ell))$
- $(\ell, Z) \rightsquigarrow (\ell', (Z \land g)[Y := 0] \land I(\ell'))$ if $(\ell, g, a, Y, \ell') \in E$
Zones – Reachability

Theorem 13

- If $(\ell, Z) \leadsto (\ell', Z')$ and $\nu' \in Z'$, then $(\ell, \nu) \rightarrow (\ell', \nu')$ for some $\nu \in Z$
- If $(\ell, \nu) \rightarrow (\ell', \nu')$ with $\nu \in Z$, then $(\ell, Z) \leadsto (\ell', Z')$ with $\nu' \in Z'$

It follows that

- whenever $(\ell', Z')$ is reachable from $(\ell_0, \{\nu_0\})$, then all states of the form $(\ell', \nu')$ with $\nu' \in Z'$ are reachable from $(\ell_0, \nu_0)$,
- whenever $(\ell', \nu')$ with $\nu' \in Z'$ is reachable from $(\ell_0, \nu_0)$, then $(\ell', Z')$ is reachable from $(\ell_0, \{\nu_0\})$. 
Example: Zones

y := 0

a

y ≤ 2

y := 0

x := 0

b

x ≤ 2

c

y ≤ 2 ∧ x ≥ 4

y ≤ x ≤ y + 4

y = 0 ∧ x ≤ 2

y = 0 ∧ x ≤ 4
Let $C_0 = C \cup \{0\}$ where $0$ is the clock with constant value $0$.

Each zone can be described using a conjunction of constraints of the form

$$x - y \leq k \quad x - y < k$$

where $x, y \in C_0$ and $k \in \mathbb{N}$.

- When $x - y \leq k$ and $x - y < k$, take only $x - y < k$,
- when $x - y \leq k$ and $x - y \leq k'$, take only $x - y \leq \min\{k, k'\}$

$\Rightarrow$ There are $|C_0||C_0|$ such constraints.

Store the constraints into a *difference bound matrix*.
Difference Bound Matrix

\[ x < 20 \land y \leq 20 \land y - x \leq 10 \land x - y \leq -10 \land z > 5 \]

matrix representation can be used to perform necessary operation: passing of time, resetting clock, intersection with constraint, ...
Network of TA

- Interleaving semantics
- Handshake communication – synchronization on c! and c? pairs
Networks of TA

Let $Chan$ be a finite set of communication channels

Assume $\Sigma = \{c! \mid c \in Chan\} \cup \{c? \mid c \in Chan\} \cup N$ where $N$ contains a special action $\tau$ (an internal action)

**Definition 14**
Consider $n$ timed automata $A_i = (L_i, \ell^i_0, E_i, I_i)$. The parallel composition $\mathcal{A} = A_1 | \cdots | A_n$ is a network of timed automata.

A location vector: $\vec{\ell} = (\ell_1, \ldots, \ell_n)$

Invariants are composed into common invariants over location vectors: $I(\vec{\ell}) = I_1(\ell_1) \land \cdots \land I_n(\ell_n)$
Semantics is defined by a transition system \((S, s_0, \rightarrow)\) where

- \(S = (L_1 \times \cdots \times L_n) \times (C \rightarrow \mathbb{R}^{\geq 0})\)
  i.e. states are of the form \((\vec{\ell}, \nu)\)
- \(s_0 = (\vec{\ell}_0, \nu_0)\) where \(\vec{\ell}_0 = (\ell_1^0, \ldots, \ell_n^0)\) and \(\nu_0(x) = 0\) for \(x \in C\)
- transitions:
  - \((\vec{\ell}, \nu) \rightarrow (\vec{\ell}, \nu + \delta)\) if \(\nu + \delta' \models I(\vec{\ell})\) for each \(\delta' \in [0, \delta]\)
  - \(((\ell_1, \ldots, \ell_i, \ldots, \ell_n), \nu) \rightarrow ((\ell_1, \ldots, \ell_i', \ldots, \ell_n), \nu')\) if there exists 
    \((\ell_i, g, a, Y, \ell_i') \in E_i\) such that
    - \(\nu \models g_i\)
    - \(\nu' = \nu[Y := 0]\) and \(\nu' \models I(\ell_1, \ldots, \ell_i', \ldots, \ell_n)\)
  - \(((\ell_1, \ldots, \ell_i, \ldots, \ell_j, \ldots, \ell_n), \nu) \rightarrow ((\ell_1, \ldots, \ell_i', \ldots, \ell_j', \ldots, \ell_n), \nu')\) if there exist 
    \((\ell_i, g_i, c?, Y_i, \ell_i') \in E_i\) and 
    \((\ell_j, g_j, c!, Y_j, \ell_j') \in E_j\) such that
    - \(\nu \models g_i \land g_j\)
    - \(\nu' = \nu[Y_i \cup Y_j := 0]\) and \(\nu' \models I(\ell_1, \ldots, \ell_i', \ldots, \ell_j', \ldots, \ell_n)\)
UPPAAL is a toolbox for modeling, simulation and verification of real-time systems

- Uppsala University + Aalborg University = UPPAAL

- Modeling language: networks of timed automata (+ additional features)
- widely used for teaching
- several industrial case studies
- www.uppaal.org
Functionality of UPPAAL

- **modeling** – graphical tool for specification of timed automata, templates
- **simulation** – simulation of the model (manual, random)
- **verification** – verification of simple properties (restricted subset of Computation Tree Logic), counterexample can be simulated

Java user interface and C++ verification engine
Extensions of Timed Automata (UPAAL)

- **Bounded integer variables** – declared as
  \[
  \text{int}[\text{min},\text{max}] \text{ name}
  \]
  where \text{min} and \text{max} are the lower and upper bound, respectively. Violating a bound leads to an invalid state that is discarded at run-time.

- **Arrays**

- **Broadcast channels** – One sender \( c! \) can synchronise with an arbitrary number of receivers \( c? \). Any available receiver must synchronize. Broadcast sending is never blocking.
Fischer’s Algorithm

With the following declarations (for 6 processes):

\[
\text{int}[0,6] \ id; \ \text{const} \ k \ 2; \ \text{clock} \ x
\]

and the following parameter (for 6 processes): \text{int}[1,6] \ pid
Extensions to TA in UPPAAL

- **Urgent locations** – time is not allowed to pass in the location, i.e., they are semantically equivalent to adding an extra clock $x$ that is reset on all incoming edges, and having an invariant $x \leq 0$ on the location.

- **Committed locations** – even more restrictive than urgent locations. A state of a network is committed if any of its locations is committed. A committed state cannot delay and the next transition must involve an outgoing edge from at least one of the committed locations.

... useful in modeling atomic actions.
The properties

UPPAAL tool uses a simple fragment of CTL as a specification language

Syntax:

\[ E \lozenge P \mid A \square P \mid E \square P \mid A \lozenge P \mid P \rightarrow P \]

\[ P := A.\ell \mid g_c \mid g_d \mid \neg P \mid P \lor P \]

where

- \( A.\ell \) – a location \( \ell \) of an automaton \( A \) (in a given network)
- \( g_c \) – a clock constraint
- \( g_d \) – a predicate over data variables (such as \( v \geq 1 \), or \( v = v' - 1 \))
Properties

- **E◊P** = it is possible to reach a state in which \( P \) is satisfied

  (written as E<>P)

- **A◊P =** \( P \) will inevitably become true, the automaton is guaranteed to eventually reach a state in which \( P \) is true.

  (written as A<>P)
Properties

- $A \square P = P$ holds always and everywhere in the future (written as $A[\square]P$)
- $E \square P = P$ is potentially always true; there is a run in which $P$ is true in all states (written as $E[\square]P$)
Properties

- $P \rightarrow Q = P$ leads to $Q$; if $P$ becomes true, $Q$ will inevitably become true later on;

$$P \rightarrow Q \equiv A \Box (P \text{ imply } A \Diamond Q)$$

(written as $P \rightarrow Q$)
Syntax of TCTL state-formulas over a set of atomic propositions $AP$ and a set of clocks $C$:

$$\Phi ::= \text{true} \mid a \mid g \mid \Phi \land \Phi \mid \neg \Phi \mid E\Phi U^J \Phi \mid A\Phi U^J \Phi$$

where $a \in AP$, $g$ is a clock constraint and $J$ is an interval in $\mathbb{R}^\geq 0$ with bounds in $\mathbb{N}$
A run is *divergent*, if its total execution time is infinite
recall that a run is a maximal path; it can be convergent if either it makes
finitely many transitions, or the length of delays converges to a finite number

Let $L$ be a function which to every location assigns a set of
atomic propositions. For a state $s = (\ell,\nu)$ we define a
satisfaction relation $\models$ by

- $s \models \text{true}$
- $s \models a$ iff $a \in L(\ell)$
- $s \models g$ iff $\nu \models g$
- $s \models \neg \Phi$ iff $s \not\models \Phi$
- $s \models \Phi_1 \land \Phi_2$ iff $(s \models \Phi_1)$ and $(s \models \Phi_2)$
- $s \models E\Phi_1 U^J\Phi_2$ iff $\omega \models \Phi_1 U^J\Phi_2$ for some divergent run $\omega$
- $s \models A\Phi_1 U^J\Phi_2$ iff $\omega \models \Phi_1 U^J\Phi_2$ for all divergent runs $\omega$
Let \( \omega = (\ell_0, v_0) \xrightarrow{h_1} (\ell_1, v_1) \xrightarrow{h_2} (\ell_2, v_2) \xrightarrow{h_3} \cdots \) be divergent run. Here each \( h_i \) is either a real number (delay), or an action of \( \Sigma \).

Define
\[
\delta_i = \begin{cases} 
  h_i & \text{if } h_i \text{ is a delay in } \mathbb{R}^\geq 0 \\
  0 & \text{otherwise, i.e., } h_i \in \Sigma
\end{cases}
\]

Given \( t \in \mathbb{R}^\geq 0 \) we denote by \( \omega_t \) the state “visited” by \( \omega \) at time \( t \):

\[
\omega_t = (\ell_i, v_i + \delta)
\]
where

\( i \) is the maximal number s. t. \( \sum_{j=1}^i \delta_j \leq t \)

\( \delta = t - \sum_{j=1}^i \delta_j \)

\( \omega \models \Phi_1 U^J \Phi_2 \) if there is time \( t \in J \) such that

\( \omega_t \models \Phi_2 \)

\( \omega_{t'} \models \Phi_1 \lor \Phi_2 \) for all \( t' < t \)
TCTL – Very Briefly – Examples

- $E \textnormal{true} U^{[0,1]} a = \text{there exists a run which reaches a location satisfying } a \text{ during the first time unit}$

- $E b U^{[0,1]} a = \text{there exists a run which reaches a location satisfying } a \text{ at some time } t \in [0,1] \text{ and before that visits only states that satisfy either } b \text{ or } a$

Define $\Diamond^J \Phi \equiv \textnormal{true} U^J \Phi$ and $\Box^J \Phi = \neg \Diamond^J \neg \Phi$

- $A \Diamond^{[0,3]} a$

- $A \Diamond^{[1,2]} (E \Diamond^{[0,1]} a)$

- $A \Box^{[0,1]} \neg (E \Diamond^{[0,11]} a)$
TCTL Model Checking: Approach

**Algorithm 43** Basic recipe of TCTL model checking

*Input:* timed automaton $TA$ and TCTL formula $\Phi$ (both over $AP$ and $C$)

*Output:* $TA \models \Phi$

\[ \hat{\Phi} := \text{eliminate the timing parameters from } \Phi; \]

\[ \text{determine the equivalence classes under } \cong; \]

\[ \text{construct the region transition system } TS = RTS(TA); \]

\[ \text{apply the CTL model-checking algorithm to check } TS \models \hat{\Phi}; \]

\[ TA \models \Phi \text{ if and only if } TS \models \hat{\Phi}. \]

How to eliminate timing intervals $J$ other than $[0, \infty)$?

- introduce a new clock $z$, which is never reset
- check if $z \in J$
Algorithm 44 TCTL model checking (basic idea)

Input: non-zeno, timelock-free timed automaton TA and TCTL formula $\Phi$
Output: “yes” if $TA \models \Phi$, “no” otherwise.

$$R := RTS(TA \oplus z, \Phi);$$

(* with state space $S_{rts}$ and labeling $L_{rts}$ *)

for all $i \leq |\Phi|$ do
  for all $\Psi \in \text{Sub}(\Phi)$ with $|\Psi| = i$ do
    switch($\Psi$):
      true : $Sat_R(\Psi) := S_{rts};$
      $a$ : $Sat_R(\Psi) := \{ s \in S_{rts} \mid a \in L_{rts}(s) \};$
      $\Psi_1 \land \Psi_2$ : $Sat_R(\Psi) := \{ s \in S_{rts} \mid \{a_{\Psi_1}, a_{\Psi_2}\} \subseteq L_{rts}(s) \};$
      $\neg \Psi'$ : $Sat_R(\Psi) := \{ s \in S_{rts} \mid a_{\Psi'} \notin L_{rts}(s) \};$
      $\exists(\Psi_1 \cup^J \Psi_2)$ : $Sat_R(\Psi) := Sat_{CTL}(\exists((a_{\Psi_1} \lor a_{\Psi_2}) \cup ((z \in J) \land a_{\Psi_2})));$
      $\forall(\Psi_1 \cup^J \Psi_2)$ : $Sat_R(\Psi) := Sat_{CTL}(\forall((a_{\Psi_1} \lor a_{\Psi_2}) \cup ((z \in J) \land a_{\Psi_2})));$
    end switch

(* add $a_\Psi$ to the labeling of all state regions where $\Psi$ holds *)

forall $s \in S_{rts}$ with $s\{z := 0\} \in Sat_R(\Psi)$ do $L_{rts}(s) := L_{rts}(s) \cup \{a_\Psi\}$ od;

od
od
if $I_{rts} \subseteq Sat_R(\Phi)$ then return “yes” else return “no” fi