Timed Automata and Model-Checking

Serge Haddad
LAMSADE-CNRS UMR 7024
Université Paris-Dauphine

Organisation

• Timed automata
• Reachability in timed automata
• TCTL model-checking for timed automata
• Undecidability results
• MITL model-checking for timed automata
Motivation

Why time in models?
- In telecommunication protocols, time-out are necessary to face faulty environments.
- In critical systems, time is necessary to schedule, suspend or abort tasks.
- In information systems, time is necessary to evaluate the accuracy of a data.

Why time in property languages?
- Bounded time response which expresses temporal relation between events.
- Termination in bounded time which corresponds to absolute time of the execution.
Timed automata: syntax

A timed automaton (TA) is:
- a finite automaton
- enlarged with a set of clocks \( X \) which evolve synchronously and continuously.

States (locations) of an automaton include invariants which restrict the way time may elapse in a location and whose syntax is:
- \( \land_{x \in X'} x \sim c \) where \( \sim \in \{<, \leq\} \) and \( c \) is some constant

Transitions of an automaton include clock resets and guards which restrict the temporal occurrence of a transition and whose syntax is:
- \( \land_{x \in X'} x \sim c \) where \( \sim \in \{>, <, \leq, \geq\} \) and \( c \) is some constant
Timed automata: an example
Timed automata: semantics

A configuration of timed automaton is given by:

- a location \( l \),
- a value \( v(x) \) per clock satisfying the location invariant \( (v|\text{=Inv}(l)) \).

A configuration may evolve by a delay \( d \) such that the invariant is still satisfied: \( (v+d|\text{=Inv}(l)) \) with \( (v+d)(x)=v(x)+d \)

A configuration may change by a transition \( (l,g,a,R,l') \) to a configuration \( (l',v') \) iff:

- \( v|\text{=}g \)
- if \( x \in R \) then \( v'(x)=0 \) else \( v'(x)=v(x) \)
- \( v'|\text{=}\text{Inv}(l') \)
Example continued

What happens next?
Network of timed automata

Modular design requires "composition" of timed automata. A network of timed automata (NTA) is:

- a set of timed automata,
- a (partial) synchronization function from "vectors" of local actions to global actions.

A network of timed automata may always be transformed into a timed automaton equivalent w.r.t. strong timed bisimulation and reachability.

But this translation yields an exponential blow up.
Network of timed automata: the train crossing example

The trains $i=1, 2, \ldots$

- **Far$_i$**: $10 < x_i < 20; \text{exit!}; \emptyset$
- **Bef$_i$**: $20 < x_i < 30; a; x_i := 0$
- **On$_i$**: $x_i < 20$

- **Far$_i$** transitions to **Bef$_i$** with the condition $x_i < 30$
- **Bef$_i$** transitions to **On$_i$** with the condition $x_i < 20$
Network of timed automata: the train crossing example

The gate

- **Open**
  - Transition: `true; down?; y:=0`
  - Guard: `y<10`

- **Low**
  - Transition: `y<10`
  - Guard: `a; Ø`

- **Raise**
  - Transition: `true; up?; y:=0`
  - Guard: `y<10`

- **Close**
  - Transition: `y<10`
  - Guard: `a; Ø`
Network of timed automata: the train crossing example

The controller

Goup

Godown

Idle

z ≤ 10; down!; ∅
z ≥ 20; up!; ∅
true; exit?; z := 0
true; app?; z := 0
true; exit?; ∅
true; app?; ∅
true; app?; z := 0
true; app?; ∅
true; exit?; ∅
Network of timed automata: the train crossing example

The synchronization function

<table>
<thead>
<tr>
<th>Train₁</th>
<th>Train₂</th>
<th>Gate</th>
<th>Controller</th>
</tr>
</thead>
<tbody>
<tr>
<td>app!</td>
<td>app!</td>
<td>app?</td>
<td>app</td>
</tr>
<tr>
<td>exit!</td>
<td>exit!</td>
<td>exit?</td>
<td>exit</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>down?</td>
<td>down!</td>
</tr>
<tr>
<td>a</td>
<td>a</td>
<td>up?</td>
<td>up!</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>up</td>
</tr>
</tbody>
</table>
Invariants in timed automata

Invariants may be deleted in timed automata:

- by adding the conjunct $\text{Inv}(l)$ to every guard of an edge $e$ outgoing from $l$
- by adding the conjunct $\bigwedge_{x \in X' \setminus R(e)} x \sim c$ to every guard of an edge $e$ incoming to $l$ with $\text{Inv}(l) = \bigwedge_{x \in X'} x \sim c$

Nevertheless, this transformation

- is valid w.r.t. (restricted) reachability and linear temporal logics
- is invalid w.r.t. bisimulation and branching temporal logics
Reachability analysis: the key idea

The number of (reachable) configurations is infinite (and even uncountable). So one wants to partition configurations into regions such that:

1. Two configurations in a region allow the same transitions and the new configurations belong to the same region.

2. If a configuration in a region letting time elapse reaches a new region every other configuration may reach the same region by time elapsing.

3. There is a finite representation of a region such that the discrete and time successors of the region are computable.

4. The number of regions is finite.
A first partition
(two clocks and two locations)

Guards and invariants check integer values

Why this partition is not appropriate?
A second partition
(two clocks and two locations)

The exact value of a clock is irrelevant when it is beyond the maximal constant of the TA (here 2)

Why this partition is not appropriate?
A third partition
(two clocks and two locations)

Check that this partition is appropriate
The region graph: illustration

Build the region graph of the above TA
About reachability in the region graph

Warning: when a region is "reachable", it does not mean that every configuration of the region is reachable.

However it means that there is another reachable configuration of the region which differs only on the values of irrelevant clocks.

Hence, in order to check the reachability of a configuration, it is enough to increase the maximal constant.

- example: reachability of \((1.0, 1.7, 2.3)\) requires to choose at least 3 as maximal constant.
Regions and zones

In practice, the building of the region graph is relatively inefficient since the partition is too finer w.r.t. a particular TA.

An alternative way is to work with zones. A zone is defined by particular linear constraints:

- $\bigwedge_{x \in X} Z(0, x) \sim x \sim Z(x, 0)$
- $\bigwedge_{x, y \in X} x - y \sim Z(x, y)$ where $\sim \in \{<, \leq\}$ and $Z(x, y)$ is some constant including $\infty$ and $-\infty$.

Zones are a useful data structure:

- They admit a canonical representation computable in polynomial time (via a shortest path algorithm in graphs) including the emptiness test.
- The projection, relativisation, past and future of a zone and intersection of zones are zones which can be efficiently computed.
Operations on zones

Projection
\[ \text{Proj} (Z, x_1) (x_2, x_3, \ldots, x_n) = \exists x_1 \ Z' (x_1, x_2, \ldots, x_n) \]

Relativisation
\[ \text{Rel} (Z, x_1) (x_2, x_3, \ldots, x_n) = \exists x_1 \ Z (x_1, x_2 + x_1, \ldots, x_n + x_1) \]

Past (positive variables)
\[ \text{Past} (Z) (x_1, x_2, \ldots, x_n) = \exists t \geq 0 \ Z (x_1 + t, x_2 + t, \ldots, x_n + t) \]

Future (positive variables)
\[ \text{Future} (Z) (x_1, x_2, \ldots, x_n) = \exists t \geq 0 \ Z (x_1 - t, x_2 - t, \ldots, x_n - t) \]

Reset
\[ R (Z, x_1) (x_1, x_2, x_3, \ldots, x_n) = \exists x \ Z (x, x_2, \ldots, x_n) \land x_1 = 0 \]
Canonical representation of zones (1)

Draw the zone on the plane
Canonical representation of zones (2)

Canonize(Ord)  // Complexity ?
    for x ∈ X ∪ {0} do
        for y ∈ (X ∪ {0})\{x} do
            for z ∈ (X ∪ {0})\{x} do
                otemp:=min(Ord(y,z),Ord(y,x)+Ord(x,z));
                if y≠z then
                    Ord(y,z):=otemp;
                else if otemp<(≤,0) then
                    return("empty zone");

(∼,c)+(∼',c')=(∼",c+c'"
    with if ~=≤ ∧ ~'=≤ then ∼"=≤ else ∼"=<
    note that c and c' range over N ∪ ∞

(∼,c)≤(∼',c') iff
    either c<c'
    or c=c' and (∼=< or ∼'=≤)
The zone graph: illustration

\[ Z_0 = (l_0, \{ x=y=0 \}) \]
\[ Z_1 = (l_0, \text{Future}(Z_0) \cap \{ x\leq 1 \}) = (l_0, \{ 0\leq x=y\leq 1 \}) \]
\[ Z_2 = \text{Future}((l_1, Z_1 \cap \{ x\leq 1 \})) \cap \{ x\leq 1 \} = (l_1, \{ 0\leq x=y\leq 1 \}) \]
\[ Z_3 = \text{Future}((l_1, R[Z_1 \cap \{ x=1 \}; \{ y \}])) \cap \{ x\leq 1 \} = (l_1, \{ x=1\land y=0 \}) \]
\[ Z_4 = \text{Future}((l_2, Z_3 \cap \{ 1\leq x\land y\leq 0 \})) = (l_2, \{ 0\leq y=x-1 \}) \]
The zone graph: exercise

Build the zone graph of the above TA
The zone graph: non termination

\[ x \geq 1 \land y = 1; a; y := 0 \]

[Diagram of a zone graph with transitions and labels]
The zone graph: extrapolation

During the building of a zone substitute to every constraint ($K$ is the maximal constant):

- $x - y \sim c$ the constraint $x - y \sim K$ when $c < -K$
- $x - y \sim c$ the constraint $x - y \sim \infty$ when $c > K$

Extrapolation enlarges the zone with "equivalent" configurations (not straightforward) and ensures termination.

$Z_1 = (l_0, \{ 0 \leq x = y \leq 1 \})$

$Z_2' = (l_0, \{ 0 \leq y = x - 1 \leq 1 \})$ is unchanged by extrapolation

$Z_3' = (l_0, \{ 0 \leq y = x - 2 \leq 1 \})$ becomes $Z_3 = (l_0, \{ 0 \leq y \leq 1 = \wedge y \leq x - 1 \})$

$Z_4' = (l_0, \{ 0 \leq y \leq 1 = \wedge y \leq x - 2 \})$ becomes $Z_3$
The "backward" zone graph always terminates since the coefficients of the zones remain bounded.

Is \((l_2, x=2, y=3)\) reachable in the following TA?
The "backward" zone graph: exercise

Build a "backward" zone graph of the above TA to check whether \((l_0, x=3, y=2)\) is reachable.
Timed Computation Tree Logic TCTL (1)

TCTL has been introduced in order to apply model-checking algorithms similar to those of CTL.

A formula of TCTL includes clocks which may be in the scope of some reset operator. A formula is said closed if all its clocks are in such a scope otherwise it is said open.

The atomic properties are:

- the locations of a TA or of a NTA (it could also include events or propositions labelling locations)
- a constraint whose syntax is $x \sim c$ with $x$ a clock of the TA or of the formula, where $\sim \in \{<, \leq\}$ and $c$ is some constant

TCTL includes the boolean operators.
The reset operator has the form $\mathbf{x} \cdot f$ where $\mathbf{x}$, a formula clock, occurs in $f$ (otherwise it is irrelevant).

**Observation:** the next operator has no meaning when related to a delay step. So TCTL will not use it.

By equivalence, we can restrict ourselves to two temporal operators:

- $\mathbf{A} f U g$
- $\mathbf{E} f U g$

The "weak until" could be added with an appropriate semantics.
Formulas are evaluated on *extended* configurations \((l, v, w)\) where \(l\) is a (vector of) location(s), \(v\) is a TA clock valuation and \(w\) is a formula clock valuation.

A timed sequence of the TA starting from an extended configuration let the formula clocks synchronously evolve.

A configuration \((l, v)\) fulfills \(\phi\) if \((l, v, 0) \models \phi\) with 0 being the null vector. An *extended* configuration \((l, v, w)\) fulfills:

- \(l'\), if \(l'\) occurs in \(l\)
- \(x \sim c\), if \(v(x) \sim c\) or \(w(x) \sim c\) depending on the type of \(x\)
- \(x.\phi\), if \((l, v, w') \models \phi\) where \(w'(x) = 0\) and \(w'(x') = w(x')\) for \(x' \neq x\)
An extended configuration fulfills formula $A_f U g$ (resp. $E_f U g$) if every (resp. at least one) maximal sequence meets a configuration $c_f$, such that:

- every configuration $c_f'$ preceding $c_f$, fulfills $f \lor g$
- $c_f \models g$

**Observation:** The previous configurations may fulfill either $f$ or $g$ due to the dense-time hypothesis.

For sake of simplicity, we do not distinguish between *divergent* and *zeno* maximal sequences. Corresponding algorithms are similar with additional technical details.
Some formulas of TCTL

Time elapses continuously, i.e. this formula is true if there is no maximal time stationary sequence from the configuration.

$$x. (Ax=0 U x>0)$$

Whenever a train is detected, the gate will be closed at most 5 t.u. after.

$$AG(x. ((Be f_1 v Be f_2) \Rightarrow AF(Close \land x \leq 5)))$$

The gate is open and will be immediately closed.

$$(\neg Close) \land x. ((At u e U (Close \land x=0)))$$

Write a TCTL formula meaning that the gate will be closed in at most 5 t.u. and will stay closed for at least 10 t.u.
Model checking a formula $\phi$ of TCTL

Build an extended region graph,

- whose maximal constant takes into account the constants occurring in $\phi$.
- obtained by duplicating all the reachable regions w.r.t. the possible values of the formula clocks
- and completing by the appropriate edges

Label its regions with the subformulas of $\phi$ bottom up.

The atomic properties and the reset operator are straightforwardly labelled.

Use standard algorithms to check $A_p U q$ and $E_p U q$ when $p$ and $q$ are (transformed into) atomic properties.

The reachability and the TCTL model-checking problems are $P$-space complete.
Model checking: illustration

\[ l_0 \xrightarrow{x \geq 1; a; x := 0} l_1 \]

\[ x \leq 1 \quad x \leq 2 \]

\[ y. (E l_0 U (l_1 \land y = 2)) \]

The green regions are the ones which fulfill \( E l_0 U (l_1 \land y = 2) \)
Model checking LTL-like languages (1)

Since a LTL formula can be transformed into an equivalent Büchi automaton,

- we can interpret a TA $\mathcal{B}$ with acceptance conditions as a "timed formula" (here over the labels of transitions)
- and define the satisfaction of a TA $\mathcal{A}$ by the "formula" $\mathcal{B}$ as the language inclusion $L(\mathcal{A}) \subseteq L(\mathcal{B})$

This reduces the problem of model checking to the inclusion problem.

Let us take $\mathcal{A}$ as the universal TA, i.e. which accepts any timed sequence. Then this particular model checking problem is the universality problem: is $L(\mathcal{B})$ the language of all timed sequences?

We now show that this problem is undecidable.
Two-counter machines

A two-counter machine is a set of instructions which manage two positive integer counters.

A jump

An incrementation

A decrementation

A conditional jump

The stop

Restriction: decrementations must follow a conditional jump
Proof of undecidability (1)

The termination problem of the two-counter machines is undecidable. Let the computation of the machine be:
\[(I(0)=I_1,v1(0)=0,v2(0)=0)\ldots(I(n),v1(n),v2(n))\ldots\]

Then a timed sequence is said to be a simulating sequence iff:

- At time $n$, there is exactly an occurrence of $a_{I(n)}$
- During $]n,n+1[$ there are exactly $v1(n)$ occurrences of $b1$ and $v2(n)$ occurrences of $b2$
- If $I(n)$ does not update counter $i$, then all the occurrences of $b_i$ in $]n,n+1[$ and $]n+1,n+2[$ are related by 1 t.u.
- If $I(n)$ increments counter $i$, then all the occurrences of $b_i$ in $]n+1,n+2[$ except the last one and those of $]n,n+1[$ and are related by 1 t.u.
- If $I(n)$ decrements counter $i$, then all the occurrences of $b_i$ in $]n,n+1[$ except the last one and those of $]n+1,n+2[$ are related by 1 t.u.
One builds a TA which accepts every timed sequence except the simulating sequences leading to the stop instruction.

So the program terminates iff the language of the corresponding TA is not the universal language.

This TA accepts the same timed sequences as the ones of a set of TAs, each one accepting the timed sequences which violate one of the condition to be a simulating sequence leading to the stop instruction.
Violation of conditions (1)

Two transitions at the same instant

No $a_1$ at instant 0

No $a_k \in A$ at integer instant $i > 0$
Violation of conditions (2)

Some $a_k \in A$ at a non integer instant (assuming that they occur at integer instants)

An occurrence of $b_1$ or $b_2$ during $]0,1[$
Violation of conditions (3)

Let $I_m: cpt_2 ++; goto I_n$ be an instruction

$a_m$ at some instant and no $a_n$, 1 t.u. later

It "misses" some $b_1$ in the second interval
Violation of conditions (4)

Let $I_m: c p t_2++; g o t o I_n$ be an instruction

It "misses" some $b_1$ in the first interval

It "misses" some $b_2$ in the second interval
Violation of conditions (5)

Let $I_m: cpt_2 ++; goto I_n$ be an instruction

The last $b_2$ in the second interval matches one in the first interval

true; $a_m; x:=0$  
true; $x<1; b_2; y:=0$  
y=1; $b_2; \emptyset$

true; $\sum; \emptyset$  
true; $\sum; \emptyset$  
true; $\sum; \emptyset$  
$x \geq 2; \sum; \emptyset$

$\forall x < 2; \sum \backslash b_2; \emptyset$
Violation of conditions (6)

Let $I_m : \text{cpt}_2 ++; \text{goto } I_n$ be an instruction

It "misses" two $b_2$ in the first interval
Violation of conditions (7)

The computation does not "stop"

Design the TA corresponding to instruction

\[ I^m : \text{if } \text{cpt}_1 > 0 \text{ then goto } I^n_{n1} \text{ else goto } I^n_{n2} \]
An alternative way would be to:

- interpret a TA $B$ with acceptance conditions as the negation of a "timed formula" (say $\mathcal{F}$)
- and define the satisfaction of a TA $A$ by the "formula" $\mathcal{F}$ as the language equation $L(A) \cap L(B) = \emptyset$

As one can (easily) build the automaton $A \otimes B$ which accepts $L(A) \cap L(B)$ by a synchronized product, this reduces the problem of model checking to the emptiness problem.

The emptiness problem is solved by:

- building the region automaton of the TA
- checking the Büchi requirement on the region automaton
Complementation of TA (1)

However how do we specify $f$?

- if $f$ is equivalent to some TA $B'$ then it accepts the complementary language of the one of $B$
- so the undecidability result entails that we cannot effectively compute $B'$!

But even worse TA are not closed under complementation.

The complementary language $L$ is the set of words with no pair of "a" occurrences separated by 1.
Complementation of TA (2)

Let $\mathcal{A}$ be TA with $c$ constraints then, in a configuration of a timed sequence at time $\tau$, one can partition $[\tau, \infty[$ in at most $2^c$ intervals such that inside an interval the value of every guard is invariant.

Assume that $\mathcal{A}$ accepts $L$.

Take a timed sequence of $L$, $(a, \tau_1)\ldots(a, \tau_n)(a, \tau_{n+1})\ldots(a, \tau_{2n-1})$ with $0<\tau_i<\tau_{i+1}<1$, $0<1+\tau_i<\tau_{n+i}<1+\tau_{i+1}$ and $n>2^c$. Take the partition $[1, \infty[$ at time 1. There is an interval which contains two points $1+\tau_i$ and $1+\tau_{i+1}$.

Show that $\mathcal{A}$ accepts the word

$$(a, \tau_1)\ldots(a, \tau_{n+i-1})(a, 1+\tau_i)(a, \tau_{n+i+1})\ldots(a, \tau_{2n-1})$$
The undecidability result comes from the fact that we can specify the exact delay between events.

MITL is an LTL-like language with the following operators.

The atomic properties are the events or the locations of a TA or of a NTA.

MITL include the boolean operators.

MITL has one temporal operator $\mathsf{fU}_I\mathsf{g}$, with $I$ a non negative interval such that its bounds are integer (including $\infty$) and must be distinct.
The temporal operator of MITL

A timed sequence fulfills $f \mathcal{U}_{I} g$ iff there is a configuration $c f$ of the timed sequence occurring at time $\tau \in I$ such that:

- The timed subsequence starting at $c f$ fulfills $g$
- Any time subsequence starting previously fulfills $f$

*(in fact the semantics is slightly more complex for expressiveness reasons)*

Note that some form of exact delays are expressible in MITL:

$$
\neg f \mathcal{U}_{[c,c]} f \iff G_{[0,c]} \neg f \land F_{[0,c]} f
$$

The MITL model-checking problem is $Exp$-space complete and $P$-space complete when for every interval one bound is either 0 or $\infty$ *(via the building of an equivalent TA with acceptance conditions)*.

*Why $G_{[0,\infty]} (a \Rightarrow \neg F_{[1,1]} a)$ is not expressible in MITL?*
MITL model-checking: an illustration

\[ f = G_{[0,1]} (p \Rightarrow F_{[1,2]} q) \]

\( q \) does not occur in \([1, 3]\)
q occurs in $[1, 2[$ and does not occur in $[2, 3[$

\[
\begin{align*}
\text{true; } \sum; \emptyset & \quad \text{x<1; } \varepsilon; y:=0 \\
& \quad \text{y=1; } q; \emptyset \\
& \quad \text{x<1; } \sum \setminus p; \emptyset \\
& \quad \text{x\geq3; } \sum; \emptyset \quad \lor \quad \text{x<3; } \sum \setminus q; \emptyset
\end{align*}
\]

q does not occur in $[1, 2[$ and occurs in $[2, 3[$

\[
\begin{align*}
\text{true; } \sum; \emptyset & \quad \text{x<1; } \varepsilon; y:=0 \\
& \quad \text{y=2; } q; \emptyset \\
& \quad \text{true; } \sum \setminus p; \emptyset \\
& \quad \text{x<1; } \sum; \emptyset \quad \lor \quad 1 \leq x; \sum \setminus q; \emptyset \\
& \quad \text{true; } \sum; \emptyset
\end{align*}
\]
MITL model-checking: an illustration

$q$ occurs in $[1, 2[$ and occurs in $[2, 3[$
Some references


