Model-checking

Does the system satisfy the property?

Modelling

Time!

Context: verification of embedded critical systems

Time

- naturally appears in real systems
- appears in properties (for ex. bounded response time)

→ Need of models and specification languages integrating timing aspects
A Case for Dense-Time

Time domain: discrete (e.g. $N$) or dense (e.g. $Q^*$)
- A compositionality problem with discrete time
- Dense-time is a more general model than discrete time

Roadmap

- Timed automata, decidability issues
- Some extensions of the model
- Implementation of timed automata

Timed automata

- A finite control structure + variables (clocks)
- A transition is of the form:

  ![Diagram of timed automaton]

  - Enabling condition
  - Reset to zero

  An enabling condition (or guard) is:

  \[ g ::= x \sim c \mid x \negsim c \mid g \land g \]

  where $\sim \in \{>,=,\geq\}$
**Timed automata (example)**

\[ x, y : \text{clocks} \]

\[ x \leq 5, a, y := 0 \quad x - y > 3, b, x := 0 \]

![Diagram of timed automata](image1)

\[ \ell_0 \xrightarrow{a} \ell_1 \xrightarrow{b} \ell_2 \]

\[
\begin{array}{c|cccc}
   & x & y \\
\hline
\ell_0 & 0 & 4.1 & 4.1 & 5.5 & 0 \\
\ell_1 & 0 & 4.1 & 0 & 14 & 14 \\
\end{array}
\]

**clock valuation**

**Timed automata (example)**

\[ x, y : \text{clocks} \]

\[ x \leq 5, a, y := 0 \quad x - y > 3, b, x := 0 \]

![Diagram of timed automata](image2)

\[ \ell_0 \xrightarrow{a} \ell_1 \xrightarrow{b} \ell_2 \]

\[
\begin{array}{c|cccc}
   & x & y \\
\hline
\ell_0 & 0 & 4.1 & 4.1 & 5.5 & 0 \\
\ell_1 & 0 & 4.1 & 0 & 14 & 14 \\
\end{array}
\]

\[ \Rightarrow \text{timed word } (a,4.1)(b,5.5) \]
**TA Semantics**

- $A = (\Sigma, L, X, \rightarrow)$ is a TA
- **Configurations:** $(\ell, v) \in L \times T^X$ where $T$ is the time domain
- **Timed Transition System:**
  - **action transition:** $(\ell, v) \xrightarrow{a} (\ell', v')$ if $\exists \ell \xrightarrow{a, r} \ell' \in A$ s.t. $v \models g$
  - **delay transition:** $(q, v) \xrightarrow{d} (q, v + d)$ if $d \in T$

**Verification**

- **Emptiness problem:** is the language accepted by a timed automaton empty?
  - **Problem:** the set of configurations is infinite
    - classical methods cannot be applied
  - **Positive key point:** variables (clocks) have the same speed
Verification

Emptiness problem: is the language accepted by a timed automaton empty?

Problem: the set of configurations is infinite → classical methods can not be applied

Positive key point: variables (clocks) have the same speed

Theorem: The emptiness problem for timed automata is decidable. It is \( PSPACE \)-complete. [Alur & Dill 1990’s]

Method: construct a finite abstraction

The region abstraction

Equivalence of finite index

“compatibility” between regions and constraints
The region abstraction

Equivalence of finite index

✓ “compatibility” between regions and constraints
✓ “compatibility” between regions and time elapsing

→ a bisimulation property

The region abstraction

Equivalence of finite index

✓ “compatibility” between regions and constraints
✓ “compatibility” between regions and time elapsing

→ a bisimulation property

The region abstraction

Equivalence of finite index

✓ “compatibility” between regions and constraints
✓ “compatibility” between regions and time elapsing

The region abstraction

Equivalence of finite index

✓ “compatibility” between regions and constraints
✓ “compatibility” between regions and time elapsing

→ a bisimulation property

Timed Models: From Theory to Implementation – p. 11
The region abstraction

Equivalence of finite index

region defined by
$I_x = ]1; 2[, I_y = ]0; 1[
{x} \times {y}$

successor regions

✓ “compatibility” between regions and constraints
✓ “compatibility” between regions and time elapsing

→ a bisimulation property

Time-Abstract Bisimulation

∀ a

∀ d > 0

∀ d > 0

δ(d)
\[
\forall d \geq 0 \quad \delta(d)
\]
\[
\exists d' > 0 \quad \delta(d')
\]

\[
(a_1, v_1) \xrightarrow{h_1} (a_2, v_2) \xrightarrow{h_2} (a_3, v_3) \cdots
\]

\[
(\ell_0, R_0) \xrightarrow{h_1} (\ell_1, R_1) \xrightarrow{h_2} (\ell_2, R_2) \cdots
\]

with \(v_i \in R\) for all \(i\).
**Time-Abstract Bisimulation**

\[
\forall a \quad a \\
\exists a \quad a
\]

\[
\forall d > 0 \quad \delta(d) \\
\exists d' > 0 \quad \delta(d')
\]

\[
(\ell_0, v_0) \xrightarrow{a_1} (\ell_1, v_1) \xrightarrow{a_2} (\ell_2, v_2) \xrightarrow{a_3} \ldots \]

\[
(\ell_0, R_0) \xrightarrow{a_1} (\ell_1, R_1) \xrightarrow{a_2} (\ell_2, R_2) \xrightarrow{a_3} \ldots
\]

with \(v_i \in R_i\) for all \(i\).

**Remark:** We can not check real-time properties with a time-abstract bisimulation. We need to add clocks for the formula we want to check.

---

**The region automaton**

\[\text{timed automaton } \otimes \text{ region abstraction}\]

\[
\ell \xrightarrow{a, c=0} \ell' \text{ is transformed into:}
\]

\[
(\ell, R) \xrightarrow{\sigma} (\ell', R') \text{ if there exists } R'' \in \text{Succ}^*(R) \text{ s.t.}
\]

\[R'' \subseteq g\]

\[\lbrack c \leftarrow 0 \rbrack R'' \subseteq R'\]

\[\rightarrow \text{ time-abstract bisimulation}\]

\[\mathcal{L}(\text{reg. aut.}) = \text{UNTIME}(\mathcal{L}(\text{timed aut.}))\]

where \(\text{UNTIME}((a_1, t_1)(a_2, t_2)\ldots) = a_1 a_2 \ldots\)

---

**An example [AD 90's]**

---

**PSPACE-Easyness**

\[
\text{The size of the region graph is in } \mathcal{O}(\lvert X \rvert 2^{|X|})\
\]

\[\checkmark \text{ One configuration: a discrete location } + \text{ a region} \]
**PSPACE-Easyness**

- The size of the region graph is in $\mathcal{O}(|X|2^{|X|})$.
- One configuration: a discrete location + a region
  - a discrete location: log-space
  - a region:
    - an interval for each clock
    - an interval for each pair of clocks

- By guessing a path: needs only to store two configurations

- Needs polynomial space
PSPACE-Easyness

- The size of the region graph is in $O(|X| \cdot 2^{|X|})$.

- One configuration: a discrete location + a region
  - a discrete location: log-space
  - a region:
    - an interval for each clock
    - an interval for each pair of clocks
  \( \Rightarrow \) needs polynomial space

- By guessing a path: needs only to store two configurations

  \( \Rightarrow \) in $\text{NPSPACE}$, thus in $\text{PSPACE}$

PSPACE-Hardness

\[ M_{\text{LBTM}} \in \text{NP} \quad \text{s.t.} \quad M \text{ accepts } w_0 \text{ iff the final state of } M_{w_0} \text{ is reachable} \]

\[ w_0 \rightarrow C_j \]

\[ \{x_j, y_j\} \]

- $C_j$ contains a "a" if $x_j = y_j$
- $C_j$ contains a "b" if $x_j < y_j$

(These conditions are invariant by time elapsing)

\( \Rightarrow \) proof taken in [Aceto & Laroussinie 2002]

PSPACE-Hardness (cont.)

If $q \xrightarrow{a \delta B} q'$ is a transition of $M$, then for each position $i$ of the tape, we have a transition

\[ (q, i) \xrightarrow{g, r, 0} (q', i') \]

where:

- $g$ is $x_i = y_i$ (resp. $x_i < y_i$) if $a = a$ (resp. $a = b$)
- $r = \{x_i, y_i\}$ (resp. $r = \{x_i\}$) if $a = a$ (resp. $a = b$)
- $i' = i + 1$ (resp. $i' = i - 1$) if $\delta$ is right and $i < n$ (resp. left)

Enforcing time elapsing: on each transition, add the condition $t = 1$ and clock $t$ is reset.

Initialization: \( \text{init} \xrightarrow{t_1, r_0 = 0} (q_0, 1) \) where $r_0 = \{x_1 | w_0[0] = b\} \cup \{t\}$

Termination: \( (q_f, i) \xrightarrow{\text{end}} \)

A Model Not Far From Undecidability

- Universality is undecidable [Alur & Dill 90's]
- Inclusion is undecidable [Alur & Dill 90's]
- Determinizability is undecidable [Tripakis 2003]
- Complementability is undecidable [Tripakis 2003]
- ...

QEST'04 – Tutorial – September 2004
Timed Models: From Theory to Implementation – p. 15

QEST'04 – Tutorial – September 2004
Timed Models: From Theory to Implementation – p. 16

QEST'04 – Tutorial – September 2004
Timed Models: From Theory to Implementation – p. 17

QEST'04 – Tutorial – September 2004
Timed Models: From Theory to Implementation – p. 18
A Model Not Far From Undecidability

- Universality is undecidable
- Inclusion is undecidable
- Determinizability is undecidable
- Complementability is undecidable

An example of non-determinizable TA:

\[
\begin{align*}
    a, x & := 0 \\
    x & = 1, a
\end{align*}
\]

A Model Not Far From Undecidability

- Universality is undecidable
- Inclusion is undecidable
- Determinizability is undecidable
- Complementability is undecidable

Partial conclusion

\[\rightarrow\] a timed model interesting for verification purposes

Numerous works have been (and are) devoted to:

- the “theoretical” comprehension of timed automata
- extensions of the model (to ease the modelling)
  - expressiveness
  - analyzability
- algorithmic problems and implementation

Some extensions of the model

- adding constraints of the form \(x - y \leq c\)
- adding silent actions
- adding constraints of the form \(x + y \leq c\)
- adding new operations on clocks
Adding diagonal constraints

Adding diagonal constraints (cont.)

Open question: is this construction "optimal"?
In the sense that timed automata with diagonal constraints are exponentially more concise than diagonal-free timed automata.

Adding silent actions

Decidability: yes (actions has no influence on the previous construction)
Expressiveness: strictly more expressive!
Adding constraints of the form \( x + y \sim c \)

\[ x + y \sim c \quad \text{and} \quad x \sim c \]  

[Bérard, Dufourd 2000]

- **Decidability**: for two clocks, decidable using the abstraction

- for four clocks (or more), undecidable!

- **Expressiveness**: more expressive! (even using two clocks)

\[ x + y = 1, \; a, \; x \equiv 0 \]

\[ \{(a^n, t_1 \ldots t_n) \mid n \geq 1 \text{ and } t_i = 1 - \frac{1}{a^n}\} \]

The two-counter machine

**Definition.** A two-counter machine is a finite set of instructions over two counters \((x \text{ and } y)\):

- **Incrementation**:
  \[
  (p) : \quad x := x + 1; \; \text{goto} \; (q)
  \]

- **Decrementation**:
  \[
  (p) : \quad \text{if } x > 0 \text{ then } x := x - 1; \; \text{goto} \; (q) \text{ else goto} \; (r)
  \]

**Theorem.** [Minsky 67] The emptiness problem for two counter machines is undecidable.

Undecidability proof

- **Increment of counter \( c \):**
  \[
  x_0 \geq 2, \; u \cdot x_2 \equiv 1, \; c, \; x_2 \equiv 0
  \]
  \[
  x_2 := 0
  \]
  \[
  u = 1, \; x_0 \equiv 0, \; u \cdot x \equiv 0
  \]
  \[
  x_0 := 2, \; c, \; x_2 \equiv 0
  \]
  \[
  u \cdot x_2 = 1
  \]
  ref for \( c \) is \( x_0 \)

- **Decrement of counter \( c \):**
  \[
  x_0 \geq 2, \; u \cdot x_2 \equiv 1, \; c, \; x_2 \equiv 0
  \]
  \[
  x_2 := 0
  \]
  \[
  u = 1, \; x_0 \equiv 0, \; u \cdot x_2 \equiv 0
  \]
  ref for \( c \) is \( x_2 \)

We will use 4 clocks:
- \( u \), "tic" clock (each time unit)
- \( x_0, x_1, x_2 \): reference clocks for the two counters

"\( x \): reference for \( c \)" \( \equiv \) "the last time \( x \) has been reset is the last time action \( c \) has been performed"

[Bérard, Dufourd 2000]
Adding constraints of the form $x + y \sim c$

- Two clocks: decidable! using the abstraction

- Four clocks (or more): undecidable!

Adding new operations on clocks

Several types of updates: $x := y + c, x < c, x > c$, etc...
**Adding new operations on clocks**

Several types of updates: \( x := y + c, x := c, x := c \), etc...

- The general model is undecidable.
- Only decrementation also leads to undecidability
  - Incrementation of counter \( z \)
  - Decrementation of counter \( x \)

---

**Decidability**

\( y := 0 \) \( y := 1 \) \( x - y < 1 \)

Image by \( y := 1 \)

\( \rightarrow \) the bisimulation property is not met

The classical region automaton construction is not correct.

---

**Decidability (cont.)**

\( A \leadsto \) Diophantine linear inequations system

- is there a solution?
- if yes, belongs to a decidable class

Examples:

- constraint \( x \sim c \)
- constraint \( x - y \sim c \)
- update \( x := y + c \)
- update \( x := c \)

The constants \((\max_x)\) and \((\max_{x,y})\) define a set of regions.

---

**Decidability (cont.)**

\( y := 0 \) \( y := 1 \) \( x - y < 1 \)

\[ \begin{align*}
\max_x \geq 0 \\
\max_x \geq 0 + \max_{x,y} \\
\max_y \geq 1 \\
\max_x \geq 1 + \max_{x,y} \\
\max_{x,y} \geq 1 \\
\end{align*} \]

\( \Rightarrow \)

\[ \begin{align*}
\max_x &= 2 \\
\max_y &= 1 \\
\max_{x,y} &= 1 \\
\max_{x,y} &= -1 \\
\end{align*} \]

The bisimulation property is met.
What’s wrong when undecidable?

Decrementation \( x := x - 1 \)

\[ \max_x \leq \max_x - 1 \]
What's wrong when undecidable?

Decrementation $x ::= x - 1$

$max_x \leq max_x - 1$

etc...

Decidability (cont.)

<table>
<thead>
<tr>
<th>Diagonal-free constraints</th>
<th>General constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x ::= c, x ::= y$</td>
<td>PSPACE-complete</td>
</tr>
<tr>
<td>$x ::= x + 1$</td>
<td>PSPACE-complete</td>
</tr>
<tr>
<td>$x ::= y + c$</td>
<td>Undecidable</td>
</tr>
<tr>
<td>$x ::= x - 1$</td>
<td>Undecidable</td>
</tr>
<tr>
<td>$x + c$</td>
<td>PSPACE-complete</td>
</tr>
<tr>
<td>$x &gt; c$</td>
<td>Undecidable</td>
</tr>
<tr>
<td>$x \sim y + c$</td>
<td>PSPACE-complete</td>
</tr>
<tr>
<td>$y + c &lt; x \times y + d$</td>
<td>Undecidable</td>
</tr>
<tr>
<td>$y + c &lt; x \times z + d$</td>
<td>Undecidable</td>
</tr>
</tbody>
</table>

[Bouyer,Dufourd,Fleury,Petit 2000]
Implementation of Timed Automata

- analysis algorithms
- the DBM data structure
- a bug in the forward analysis

Notice

The region automaton is not used for implementation:
- suffers from a combinatorics explosion
  (the number of regions is exponential in the number of clocks)
- no really adapted data structure

Algorithms for “minimizing” the region automaton have been proposed...

[Alur & Co 1992] [Tripakis,Yovine 2001]

...but on-the-fly technics are preferred.
Reachability analysis

- forward analysis algorithm:
  compute the successors of initial configurations

Reachability analysis

- backward analysis algorithm:
  compute the predecessors of final configurations
The exact backward computation terminates and is correct!
Note on the backward analysis

If \( \mathcal{A} \) is a timed automaton, we construct its corresponding set of regions. Because of the bisimulation property, we get that:

"Every set of valuations which is computed along the backward computation is a finite union of regions"

The exact backward computation terminates and is correct!

Note on the backward analysis (cont.)

Let \( R \) be a region. Assume:

- \( \nu \in \overline{R} \) (for ex. \( \nu + t \in R \))
- \( \nu \equiv_{\mathit{reg}} \nu' \)

There exists \( t' \) s.t. \( \nu' + t' \equiv_{\mathit{reg}} \nu + t \), which implies that \( \nu + t' \in R \) and thus \( \nu \in \overline{R} \).
Note on the backward analysis (cont.)

If $A$ is a timed automaton, we construct its corresponding set of regions.

Because of the bisimulation property, we get that:

"Every set of valuations which is computed along the backward computation is a finite union of regions"

But, the backward computation is not so nice, when also dealing with integer variables...

$i := j \cdot k + \ell \cdot m$

Forward analysis of TA

$g, a, C := 0$

zones $Z$ $[C \leftarrow 0](Z \cap g)$

A zone is a set of valuations defined by a clock constraint

$\varphi := x \sim c \mid x - y \sim c \mid \varphi \land \varphi$

Forward analysis of TA

$g, a, C := 0$

zones $Z$ $[C \leftarrow 0](Z \cap g)$

Forward analysis of TA

$g, a, C := 0$

zones $Z$ $[C \leftarrow 0](Z \cap g)$
Forward analysis of TA

\[ g, a, C := 0 \]

zones \[ Z \]

\[ g \]

\[ \frac{y := 0, x := 0}{x \geq 1 \land y = 1, y := 0} \]

\[ \Rightarrow \text{a termination problem} \]

Non termination of the forward analysis

\[ y := 0, x := 0 \]

\[ x \geq 1 \land y = 1, y := 0 \]

\[ \Rightarrow \text{an infinite number of steps...} \]
“Solutions” to this problem

(f.ex. in [Larsen,Pettersson,Yi 1997] or in [Daws,Tripakis 1998])

- inclusion checking: if \( Z \subseteq Z' \) and \( Z' \) still handled, then we don't need to handle \( Z \)
  \[ \Rightarrow \text{correct w.r.t. reachability} \]

- convex-hull approximation: if \( Z \) and \( Z' \) are computed then we overapproximate using \( "Z \cup Z" \).
  \[ \Rightarrow \text{"semi-correct" w.r.t. reachability} \]

- extrapolation, a widening operator on zones

“Solutions” to this problem (cont.)

- convex-hull approximation: if \( Z \) and \( Z' \) are computed then we overapproximate using \( "Z \cup Z" \).
  \[ \Rightarrow \text{"semi-correct" w.r.t. reachability} \]

- activity: eliminate redundant clocks
  \[ \Rightarrow \text{correct w.r.t. reachability} \]

\[ q \xrightarrow{a,C\in\mathbb{G}} q' \quad \Rightarrow \quad \text{Act}(q) = \text{clocks}(g) \cup (\text{Act}(q') \setminus C) \]

\[ \ldots \]
The DBM data structure

DBM (Difference Bounded Matrix) data structure [Dill 1989]

\[(x_1 \geq 3) \land (x_2 \leq 5) \land (x_1 - x_2 \leq 4)\]

\[
\begin{array}{c|ccc}
 x_0 & x_1 & x_2 \\
\hline
+\infty & -3 & +\infty \\
+\infty & +\infty & 4 \\
5 & +\infty & +\infty \\
\end{array}
\]

- Existence of a normal form

\[
\begin{array}{c}
5 \\
2 \\
3 4 9 \\
\end{array}
\]

All previous operations on zones can be computed using DBMs

The extrapolation operator

Fix an integer \(k\) \(("*" represents an integer between \(-k\) and \(+k\))

\[
\begin{array}{ccc}
* & \ast & \ast \\
\hline
\ast & \ast & \ast \\
\ast & \ast & \ast \\
\end{array}
\sim
\begin{array}{ccc}
* & \ast & \ast \\
\hline
\ast & \ast & \ast \\
-k & \ast & \ast \\
\end{array}
\]

- "intuitively", erase non-relevant constraints

- \(\Rightarrow\) ensures termination
The extrapolation operator

Fix an integer \( k \) \( (\# \) represents an integer between \(-k\) and \(+k\) \)

\[
\begin{bmatrix}
\# \\
\# \\
\# \\
\# \\
\# \\
\end{bmatrix}
\sim
\begin{bmatrix}
\# \\
\# \\
\# \\
\# \\
\# \\
\end{bmatrix}
\]

✓ “intuitively”, erase non-relevant constraints

✓ ensures termination

Challenge

Propose a good constant for the extrapolation:
✓ keep the correctness of the forward computation

Solution by the past: maximal constant appearing in the automaton
✓ Several correctness proofs can be found
✓ Implemented in tools like UPPAAL, KRONOS, RT-SPIN...
✓ Successfully used on real-life examples

However…
A problematic automaton

\[ x_3 \leq 3 \]
\[ x_1, x_3 := 0 \]
\[ x_2 := 0 \]
\[ x_1 = 2, x_1 := 0 \]
\[ x_2 = 2, x_2 := 0 \]
\[ Error \]
\[ x_2 - x_1 > 2 \]
\[ x_3 - x_1 < 2 \]
\[ Error \]

A problematic automaton

\[ x_3 \leq 3 \]
\[ x_1, x_3 := 0 \]
\[ x_2 := 0 \]
\[ x_1 = 2, x_1 := 0 \]
\[ x_2 = 2, x_2 := 0 \]
\[ The \ loop \]

A problematic automaton

\[ x_3 \leq 3 \]
\[ x_1, x_3 := 0 \]
\[ x_2 := 0 \]
\[ x_1 = 2, x_1 := 0 \]
\[ x_2 = 2, x_2 := 0 \]
\[ Error \]
\[ x_2 - x_1 > 2 \]
\[ x_3 - x_1 < 2 \]

A problematic automaton

\[ x_3 \leq 3 \]
\[ x_1, x_3 := 0 \]
\[ x_2 := 0 \]
\[ x_1 = 2, x_1 := 0 \]
\[ x_2 = 2, x_2 := 0 \]
\[ The \ loop \]

The problematic zone

\[ v(x_1) = 0 \]
\[ v(x_2) = d \]
\[ v(x_3) = 2a + 5 \]
\[ v(x_4) = 2a + 5 + d \]

\[ v(x_1) \]
\[ v(x_3) \]
\[ x_4 \]
\[ [1; 3] \]
\[ [2a + 5] \]
\[ [3] \]

implies

\[ x_1 - x_2 = x_3 - x_4. \]
The problematic zone

If $a$ is sufficiently large, after extrapolation:

$$[1; 3] \implies x_1 - x_2 = x_3 - x_4.$$
General abstractions

Criteria for a good abstraction operator $Abs$:

- Easy computation
  $Abs(Z)$ is a zone if $Z$ is a zone

- Finiteness of the abstraction
  $(Abs(Z) \mid Z$ zone$)$ is finite

- Completeness of the abstraction
  $Z \subseteq Abs(Z)$

- Soundness of the abstraction
  the computation of $(Abs \circ Post)^*$ is correct w.r.t. reachability

For the previous automaton, no abstraction operator can satisfy all these criteria!

Why that?

Assume there is a "nice" operator $Abs$.

The set $(M$ DBM representing a zone $Abs(Z))$ is finite.

$\Rightarrow k$ the max. constant defining one of the previous DBMs

We get that, for every zone $Z$,

$Z \subseteq Extra_k(Z) \subseteq Abs(Z)$
Problem!

Open questions:
- which conditions can be made weaker?
- find a clever termination criterium?
- use an other data structure than zones/DBMs?

What can we cling to?

Diagonal-free:
- only guards \( x \sim c \)
  (no guard \( x \sim y \sim c \))

Theorem: the classical algorithm is correct for diagonal-free timed automata.

General:
- both guards \( x \sim c \) and \( x \sim y \sim c \)

Proposition: the classical algorithm is correct for timed automata that use less than 3 clocks.
  (the constant used is bigger than the maximal constant...)

Conclusion & Further Work

- Decidability is quite well understood.
- Needs to understand better the geometry of the reachable state space.

- data structures for both dense and discrete parts

- Some other current challenges:
  - controller synthesis
  - implementability issues (program synthesis)

To be continued...
Bibliography


[BDFP00a] Bouyer, Dufourd, Fleury, Petit. Are Timed Automata Updatable? CAV00 (LNCS 1855).


[DY96] Daws, Yovine. Reducing the Number of Clock Variables of Timed Automata. RTSS’96.


Kronos: [http://www-verimag.imag.fr/TEMPORISE/kronos/](http://www-verimag.imag.fr/TEMPORISE/kronos/)