Verification of timed and distributed systems: models, algorithms and implementability

Pierre-Alain Reynier

LSV, CNRS & ENS Cachan, France

Cachan, June 21, 2007
Why should we do verification?

**Critical systems** = systems whose faults may have serious consequences.

⇒ must have a **correct** behavior.
Examples

- **should verify**
  - “If the train is on the level crossing, then the gate is closed”

- **should verify**
  - “The right dose of radiation is delivered during the right delay“

- **should verify**
  - “If an alarm occurs, then the security procedure is activated within the right delay“
Formal methods

The most used validation approach is test:

- Not precise enough
- Does not permit an exhaustive testing of the system

Need for formal methods:
- Model-based testing
- Static analysis
- Theorem proving
- Model-checking

These approaches allow to prove mathematically the property
Model-checking: presentation

Does the system satisfy its specification?
Model-checking: presentation

Does the system satisfy its specification?
Model-checking: presentation

Does the system satisfy its specification?

\[(\neg F \text{ Error}) \land (G F \leq OK)\]
Model-checking: presentation

Does the system satisfy its specification?

\[ (\neg F \text{ Error}) \land (G F \preceq_1 \text{OK}) \]

Model-Checking Algorithm
Model-checking: some important issues

- **Development of models:** Need for adequate models
  - study of properties of models
  - comparison of expressiveness of models
  - development of translations

- **Algorithmic of verification:** Need for efficient algorithms
  - symbolic representations
  - abstractions
  - partial order techniques

→ Find a good compromise between expressiveness and efficiency.
Timed and distributed systems

- “timed”: explicit notion of time which allows a quantitative evaluation of time elapsing.
  - “A signal is always followed by an alarm.”
  - “A signal is always followed by an alarm within 4 time units.”

- “distributed”: description of the system as a composition of local subsystems which can communicate.
  - allows a smaller and more comprehensive description of complex systems.
(Networks of) Timed Automata [Alur and Dill 90]

System: \( \text{TRAIN}_1 \parallel \cdots \parallel \text{TRAIN}_n \parallel \text{GATE} \parallel \text{CONTROLLER} \)

Property: \( \neg \mathbf{F} (\text{TRAIN}_i.\text{On} \land \text{GATE}.\text{Open}) \)
(Networks of) Timed Automata [Alur and Dill 90]

Clock values:

\[
\begin{align*}
  x_i &\quad 0 \\
  y &\quad 0 \\
  z &\quad 0
\end{align*}
\]
(Networks of) Timed Automata [Alur and Dill 90]

Clock values:

\[
\begin{align*}
x_i & \quad 0 \quad \rightarrow \quad 10.2 \\
y & \quad 0 \quad \rightarrow \quad 10.2 \\
z & \quad 0 \quad \rightarrow \quad 10.2
\end{align*}
\]
(Networks of) Timed Automata [Alur and Dill 90]

Clock values:

\[
\begin{array}{c|c|c|c|c}
 x_i & 0 & 10.2 & \text{App} & 0 \\
y & 0 & 10.2 & 10.2 & 10.2 \\
z & 0 & 10.2 & 0 & 0 \\
\end{array}
\]
(Networks of) Timed Automata [Alur and Dill 90]

Train

\[ x_i < 30 \]

Closed

\[ 20 < x_i < 30, \varepsilon, \{x_i\} \]

Far

\[ 10 < x_i < 20, \text{Exit!} \]

On

\[ x_i < 20 \]

Controller

Exit?

\[ z \leq 50 \]

Exit?, \{z\}

\[ z = 50, \text{Up!} \]

App?, \{z\}

\[ z \leq 10, \text{Down!} \]

Exit?

\[ z \leq 10 \]

App?

Clock values:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_i )</td>
<td>0</td>
<td>10.2</td>
<td>App</td>
</tr>
<tr>
<td>( y )</td>
<td>0</td>
<td>10.2</td>
<td>( \rightarrow 10.2 )</td>
</tr>
<tr>
<td>( z )</td>
<td>0</td>
<td>10.2</td>
<td>( \rightarrow 10.2 )</td>
</tr>
</tbody>
</table>
(Networks of) Timed Automata [Alur and Dill 90]

Introduction

Development of models

Algorithmic of verification

Conclusion

Train

$x_i < 30$

App!, \{x_i\}

Far

$10 < x_i < 20$, Exit!

On

$20 < x_i < 30$, \(\varepsilon\), \{x_i\}

GATE

$y < 10$

Open

$y < 10$, a

Down

$y < 10$, \(\varepsilon\)

Up

$y < 10$

Controller

Exit?

$z \leq 50$

App!, \{z\}

Exit?, \{z\}

$z = 50$, Up!

C1

$z \leq 10$, Down!

App?, \{z\}

C0

$z \leq 10$

App?

Exit?

Clock values:

\[
\begin{array}{cccccccc}
\text{Train}_i & x_i & 0 & 10.2 & & & & 8 & 8 \\
y & 0 & 10.2 & & \text{App} & 0 & 8 & 18.2 & \text{Down} & 8 \\
z & 0 & 10.2 & & & 0 & 8 & 8 & &
\end{array}
\]
(Networks of) Timed Automata [Alur and Dill 90]

Train_i

- **App!, \{x_i\}**
  - x_i < 30
  - 20 < x_i < 30, \(\varepsilon\), \{x_i\}
- **Far**
  - 10 < x_i < 20, Exit!
- **On**
  - x_i < 20

Gate

- **Open**
  - Down?, \{y\}
  - y < 10
- **Down**
- **Up**
  - Up?, \{y\}
  - y < 10
- **Closed**

Controller

- **Exit?**
  - z ≤ 50
  - z = 50, Up!
- **App?, \{z\}**
- **Exit?**
  - z ≤ 10
  - z ≤ 10, Down!
- **App?**

Clock values:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>10.2</th>
<th>0</th>
<th>8</th>
<th>17.7</th>
</tr>
</thead>
<tbody>
<tr>
<td>x_i</td>
<td>0</td>
<td>↑10.2</td>
<td>App</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>y</td>
<td>0</td>
<td>10.2</td>
<td>10.2</td>
<td>8</td>
<td>17.7</td>
</tr>
<tr>
<td>z</td>
<td>0</td>
<td>10.2</td>
<td>0</td>
<td>8</td>
<td>17.7</td>
</tr>
</tbody>
</table>
(Networks of) Timed Automata [Alur and Dill 90]

Clock values:

<table>
<thead>
<tr>
<th>(x_i)</th>
<th>(y)</th>
<th>(z)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>10.2</td>
<td>10.2</td>
<td>10.2</td>
</tr>
<tr>
<td>0.2</td>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>9.7</td>
<td>17.7</td>
</tr>
<tr>
<td>17.7</td>
<td>17.7</td>
<td>17.7</td>
</tr>
</tbody>
</table>
1 Introduction
   - Why verification?
   - Model-checking
   - Timed and distributed systems

2 Development of models
   - Overview
   - Comparison of TA and TdPN

3 Algorithmic of verification
   - Unfolding of NTA
   - Reachability in TA with diagonal constraints

4 Conclusion
1 Introduction
   - Why verification?
   - Model-checking
   - Timed and distributed systems

2 Development of models
   - Overview
   - Comparison of TA and TdPN

3 Algorithmic of verification
   - Unfolding of NTA
   - Reachability in TA with diagonal constraints

4 Conclusion
Extensions of Timed Automata

- updates
- diagonal constraints
- silent transitions

Timed Automata
Extensions of Timed Automata

+ updates
+ diagonal constraints
+ silent transitions

Timed Automata

Timed Extensions of Petri Nets

+ read arcs

Time Petri Nets

Extensions of Timed Automata

Timed Extensions of Petri Nets


**Extensions of Timed Automata**

- + updates
- + diagonal constraints
- + silent transitions

**Timed Automata**

**Timed Extensions of Petri Nets**

- + read arcs

**Time Petri Nets**
Introduction

Development of models

Algorithmic of verification

Conclusion

Extensions of Timed Automata

+ updates

+ diagonal constraints

+ silent transitions

Timed Automata

Timed Extensions of Timed Automata

Timed Petri Nets

+ read arcs

Timed Petri Nets

Timed Extensions of Petri Nets

PhD Defense Pierre-Alain Reynier
Extensions of Timed Automata

+ updates

+ diagonal constraints

+ silent transitions

Timed Automata

Timed Extensions of Petri Nets

+ read arcs

Timed Petri Nets
Introduction

- Why verification?
- Model-checking
- Timed and distributed systems

Development of models

- Overview
- Comparison of TA and TdPN

Algorithmic of verification

- Unfolding of NTA
- Reachability in TA with diagonal constraints

Conclusion
Timed Petri Nets [Bolognesi, Lucidi, and Trigila 90]

Studied in [Ruiz et al 99], [Abdulla et al 01]

A TdPN is a Petri Net extended in the following manner:

- each token has an age,
- each arc is labelled by an interval to add timing constraints on the ages of the tokens which are consumed or produced,
- the transitions carry the actions.

\[ Acc = (p_2 = 0) \]

Accepting condition:

\[ \wedge \mathit{p} \mathit{p} \sim c \quad \text{(finite words)} \]
\[ \text{repeated configurations} \]
\[ \text{(infinite words)} \]
Timed Petri Nets - semantics

Acc = \( p_2 = 0 \)

timed word:

\[
\begin{align*}
p_1 & | 0; 0 \\
p_2 & | 0; 0
\end{align*}
\]
Timed Petri Nets - semantics

\[
\begin{align*}
&\text{Acc } = \ (p_2 = 0) \\
&\text{timed word:}
\end{align*}
\]
Timed Petri Nets - semantics

Acc = \( (p_2 = 0) \)

timed word: \((a, 1)\)

\[\begin{array}{l}
p_1 \\
[1, 2] \\
t_1, a \\
[0, 1[ \\
p_2 \\
[2, 3] \\
t_2, b \\
[1, 3] \\
\end{array}\]
Timed Petri Nets - semantics

Acc = (p_2 = 0)

timed word:

(a, 1)

\[
\begin{align*}
p_1 & \quad [1, 2] \\
& \quad (0; 0) \\
& \quad (1) \\
& \quad 1; 1 \\
& \quad a \\
& \quad 1 \\
& \quad (1.4) \\
& \quad 2.4 \\
p_2 & \quad [0, 1[ \\
& \quad [2, 3] \\
& \quad 0; 0 \\
& \quad (1) \\
& \quad 1; 1 \\
& \quad a \\
& \quad 0.6; 1 \\
& \quad (1.4) \\
& \quad 2; 2.4
\end{align*}
\]
Timed Petri Nets - semantics

Acc = (p_2 = 0)

timed word:

(a, 1)(b, 2.4)

\[
\begin{array}{c|cccc}
  & 0; 0 & 1; 1 & 1 & 2.4 \\
p_1 & 0; 0 & \rightarrow & a & b \\
p_2 & \rightarrow & 1; 1 & 0.6; 1 & 2; 2.4 \\
   & \rightarrow & 2 & 2 \\
\end{array}
\]
Timed Petri Nets - semantics

\[ \text{Acc} = (p_2 = 0) \]

timed word:

\[(a, 1)(b, 2.4)\]
Timed Petri Nets - semantics

Acc = (p_2 = 0)

timed word:

(a, 1)(b, 2.4)(b, 3.4)
Encoding a TA into a TdPN: the favorable case

If clocks are reset each time they are tested:

\[ x \geq 2, a, x := 0 \]

\[ \xrightarrow{\ell_1} x \xrightarrow{\ell_2} \]

\[ [2, +\infty[ \xrightarrow{]0, 0]} \]

\[ \xrightarrow{\ell_1} \xrightarrow{\ell_2} \]

\[ \Rightarrow \]

To deal with the general case, we introduce Read Arcs:

Read Arcs = arcs that check the presence and timing of tokens, but without consuming them.

- already defined in the untimed framework,
- introduced recently for TdPN in [Srba05].
Introduction of read arcs

\[ Acc = \text{true} \]

\[ p \xrightarrow{[0, 1]} a \equiv x \leq 1, a \]

The accepted languages are

\[
\begin{align*}
L_1^* &= \{(a, \tau_1) \ldots (a, \tau_n) \mid 0 \leq \tau_1 \leq \ldots \leq \tau_n \leq 1\} \\
L_1^\omega &= \{(a, \tau_1) \ldots (a, \tau_n) \ldots \mid 0 \leq \tau_1 \leq \ldots \leq \tau_n \leq \ldots \leq 1\}
\end{align*}
\]

\[ \Rightarrow \text{This defines the model of RA-TdPN, and we prove:} \]

Theorem

\[ TA \approx \text{safe RA-TdPN} \approx \text{bounded RA-TdPN} \]
Expressiveness of RA-TdPN

The natural question raised by the introduction of read arcs is:

Are read-arcs necessary?

The results we obtain depend heavily on the timed language equivalence we consider. We introduce the following ones:

- $\ast$-equivalence: finite words
- $\omega$-equivalence: infinite words
- $\omega_{nz}$-equivalence: non-Zeno infinite words

Zeno = infinite number of actions in a finite delay.
Infinite timed words: a discriminating language

\[ \text{Acc} = \text{true} \]

\[ L_1 = \{(a, \tau_1) \ldots (a, \tau_n) \ldots \mid 0 \leq \tau_1 \leq \ldots \leq \tau_n \leq \ldots \leq 1\} \]

**Lemma**

\(L_1\) is recognized by no TdPN (even unbounded).

**Corollary**

RA-TdPN \(\geq\omega\) TdPN

**Remark:** \(L_1\) is a Zeno language.
Finite and non-Zeno infinite timed words

We propose effective transformations for removing read-arcs:

Theorem (Finite timed words)

\[ \text{RA-TdPN} \equiv^* \text{TdPN} \]

Using non-Zenoness property, one can extend this result:

Theorem (Non-Zeno infinite timed words)

\[ \text{RA-TdPN} \equiv^{\omega_{nz}} \text{TdPN} \]

Remark: Integrality and boundedness are preserved.
Finite timed words

▶ For previous example:

\[ \text{Acc} = \text{true} \quad \text{Acc} = p_1 + p_2 \leq 0 \]

▶ Generalization: normalization procedure for RA-TdPN which reduces the study to five “simple” patterns.
Summary for RA-TdPN: finite and infinite non-Zeno t.w.

We have considered three features of (bounded) RA-TdPNs:

- read arcs,
- general resets vs 0-resets,
- integral nets.

Picture for finite and infinite non-Zeno timed words:

\[
\text{RA-TdPN} \equiv_{*,\omega_{n\mathbb{Z}}} \text{TdPN} \equiv_{*,\omega_{n\mathbb{Z}}} \text{0-reset TdPN}
\]

These equalities also hold for bounded nets and for integral nets.
Summary for RA-TdPN: infinite timed words

\[ \text{RA-TdPN} \equiv_{\omega} 0\text{-reset RA-TdPN} \]

\[ \text{integral RA-TdPN} \]

\[ \land_{\omega} \]

\[ \text{0-reset integral RA-TdPN} \]

\[ \sqrt{3} \]

\[ \text{integral TdPN} \equiv_{\omega} 0\text{-reset integral TdPN} \]

\[ \land_{\omega} \]

\[ \text{TdPN} \equiv_{\omega} 0\text{-reset TdPN} \]

\[ \land_{\omega} \]
Comparison of TA and TdPN

For finite and non Zeno infinite timed words:

- $\text{TA} \equiv_{*,\omega_{\mathbb{Z}}}$ bounded TdPN
- $\text{TA} <_{*,\omega_{\mathbb{Z}}}$ TdPN

For infinite timed words:

- $\text{TA} >_{\omega}$ bounded TdPN
- $\text{TA} \not<_{\omega}$ TdPN

⇒ Surprisingly, a token is not as powerful as a clock,
⇒ For Zeno timed words, read arcs are necessary.
# Introduction
- Why verification?
- Model-checking
- Timed and distributed systems

# Development of models
- Overview
- Comparison of TA and TdPN

# Algorithmic of verification
- Unfolding of NTA
- Reachability in TA with diagonal constraints

# Conclusion
1. Introduction
   - Why verification?
   - Model-checking
   - Timed and distributed systems

2. Development of models
   - Overview
   - Comparison of TA and TdPN

3. Algorithmic of verification
   - Unfolding of NTA
   - Reachability in TA with diagonal constraints

4. Conclusion
Objectives

⇒ Apply partial order techniques to timed systems.

In this work:

- Partial order techniques = unfoldings
- Timed systems = Networks of timed automata (NTA)

Classical difficulties:

- Merge time and concurrency
- Handle urgency (invariants in TA)
Unfolding of discrete-event systems

Unfolding of a system = net describing its concurrent behaviors.

Usually infinite,
but: existence of a finite and complete prefix.

→ Well known for discrete-event systems such as Petri Nets, or networks of automata (McMillan, Esparza et al, ...)

→ We can decide reachability, transition enabling, deadlocks...

Remark 1: Very efficient for safe Petri Nets

Remark 2: Networks of Automata give safe Petri Nets!

→ There remains to handle time!
Examples

Every $b$ occurrence must occur within 1 t.u. after an $a$ occurrence or the beginning of the execution.
Examples

\[
\begin{align*}
\ell_0 & \quad y \geq 3, \quad \ell_1 \\
\ell_1 & \quad \ell_2 & \quad \ell_3 \\
\ell_2 & \quad a & \quad b \\
\ell_3 & \quad \ell_1
\end{align*}
\]

\[
\begin{align*}
\ell_0 & \quad y \leq 1 \\
\ell_0 & \quad \ell_1 \\
\ell_1 & \quad \ell_2 \\
\ell_2 & \quad \ell_1
\end{align*}
\]

\[
\begin{align*}
\ell_0 & \quad a \\
\ell_0 & \quad c_1 \\
\ell_0 & \quad c_2 \\
\ell_1 & \quad b
\end{align*}
\]

\[
\begin{align*}
\ell_0 & \quad \ell_1 \\
\ell_1 & \quad \ell_2 \\
\ell_1 & \quad \ell_3
\end{align*}
\]

The firing of \(a\) is only possible if \(c_2\) fired, and not \(c_1\).
Using read arcs again!

Our idea: use the translation from NTA to RA-TdPN.

Read arcs increase concurrency.

For unfoldings, read arcs:
(1) enlarge the set of places in input of an event,
(2) may express concurrent access of two events to the same place.

More precisely:
- Point (1) induces duplication of events,
- Point (2) preserves concurrency as much as possible.
Examples continued

Here, any $a$ occurrence yields a new $b$ occurrence.
Examples continued

Here again, our unfolding represents new occurrences of $a$, and the causality between $a$ and $c_2$ appears.
Examples continued

Here again, our unfolding represents new occurrences of \( a \), and the causality between \( a \) and \( c_2 \) appears.
Going further

- **Adding time**: we attach zones to events which exactly characterize timed executions of the NTA.

- **Computation of a finite and complete prefix**:
  - Subclass of **bounded NTA**: no extrapolation makes possible the use of [Winkowski’02] algorithm for read-arc Petri nets.
    - no modification of the structure
    - complicated algorithm, restricted case.
  
- **General case**: requires to enlarge the set of synchronized events to allow application of extrapolation.
  - efficient algorithm
  - reduces concurrency
1 Introduction
   - Why verification?
   - Model-checking
   - Timed and distributed systems

2 Development of models
   - Overview
   - Comparison of TA and TdPN

3 Algorithmic of verification
   - Unfolding of NTA
   - Reachability in TA with diagonal constraints

4 Conclusion
Diagonals and forward analysis

Consists in adding constraints $x - y \sim c$ in guards of TA.

- Useful in practice,
- Does not increase expressive power of the model
- ... but adds conciseness.

Impact on reachability analysis:

- Classical reachability algorithm (FRA) for TA is not correct in presence of diagonals [Bouyer’03].
- FRA computes an over-approximation of the reachable states.

→ Solution based on the removal of diagonal constraints leads to an algorithm with exponential over-cost [BY03].
Counter-example guided abstraction refinement approach

Advantages of our approach:

- same characteristics as FRA (forward, on-the-fly...)
- removes as few diagonals as possible
- no over-cost if no false positive
Counter-example guided abstraction refinement approach

\[ \mathcal{A} \]
\[ \mathcal{D} := \emptyset \]

Safe \[ \Rightarrow \] yes

FRA(\( \mathcal{A}, \mathcal{D} \)) \[ \Rightarrow \] no

\[ \mathcal{D} := \mathcal{D} \cup \{ g \} \]

select a guard \( g \) \[ \Rightarrow \] no

consistent ? \[ \Rightarrow \] yes \[ \Rightarrow \] Unsafe

Advantages of our approach:

- same characteristics as FRA (forward, on-the-fly...)
- removes as few diagonals as possible
- no over-cost if no false positive
Implementation in UppAal

The algorithm has been implemented in UppAal and is compared with the existing algorithm of [BY03] (systematic removal)

Models considered are:
- an extension of Fischer protocol with diagonals
- a distributed version of the counter-example of [Bouyer’03]

<table>
<thead>
<tr>
<th></th>
<th>Fischer3</th>
<th>Fischer 4</th>
<th>B(2,1)</th>
<th>B(2,2)</th>
<th>B(2,3)</th>
<th>B(1,1)</th>
<th>B(1,2)</th>
<th>B(1,3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[BY03]</td>
<td>0.4</td>
<td>560</td>
<td>2080</td>
<td>2100</td>
<td>2100</td>
<td>165</td>
<td>165</td>
<td>165</td>
</tr>
<tr>
<td>CEGAR</td>
<td>0.02</td>
<td>11.6</td>
<td>140</td>
<td>460</td>
<td>835</td>
<td>67</td>
<td>115</td>
<td>176</td>
</tr>
</tbody>
</table>

→ Systematic removal is expensive.
→ Extra-cost due to analysis is negligible.
1 Introduction
   • Why verification?
   • Model-checking
   • Timed and distributed systems

2 Development of models
   • Overview
   • Comparison of TA and TdPN

3 Algorithmic of verification
   • Unfolding of NTA
   • Reachability in TA with diagonal constraints

4 Conclusion
Contributions

- **Development of models:**
  - Undecidability results for $\text{TA}_\varepsilon$ [submitted Fund. Inf.]
  - Translation from extended TA to TPN [ACSD’06]
  - Comparison of TA and TdPN [ICALP’06, submitted I&C]

- **Algorithmic of verification:**
  - Unfoldings for NTA [ATVA’06]
  - Reachability analysis of $\text{TA}_{x-y\sim c}$ [FORMATS’05]

- **Implementability:**
  - Model-checking algorithm for LTL over TA w.r.t. enlarged semantics [LATIN’06]
Conclusion (1)

**Models**

- **TA against Petri nets:**
  - Explicit clocks of TA are more suitable for describing complex timing constraints.
  - Timed extensions of Petri nets are more suitable for distributed systems.

- **Role of urgency:**
  - Find a model of Petri nets with urgency for which the coverability problem is decidable.
  - Extend relations between TA and TdPN to bisimulation.
Conclusion (2)

- Algorithms
  - Partial order techniques for timed systems.
  - Extend unfolding techniques to enhance partial order techniques for NTA, e.g. with abstraction.

- Implementability
  - Develop algorithms for richer logics
  - Build a link with standard semantics
  - Find an “easy” criterion for implementability which reduces robust model-checking to standard model-checking.