Models for Efficient Timed Verification

François Laroussinie

LSV / ENS de Cachan – CNRS UMR 8643

Monterey Workshop - “Composition of embedded systems”
Model checking

System \rightarrow \text{Formalizing step} \rightarrow \text{Properties}

\{ ? \} \models \phi
We want...

- **Expressive models**: communication, variables, timing constraints, probabilities...
- **Expressive specification languages**: natural, powerful, intuitive...
Model checking

We want...

- **Expressive models**: communication, variables, timing constraints, probabilities.
- **Expressive specification languages**: natural, powerful, intuitive.

AND EFFICIENT ALGORITHM!

Main limit of model checking = state explosion problem
State explosion problem

An example: a “protocol”…
State explosion problem

An example: a “protocol”…

![Diagram](attachment:image.png)
State explosion problem

An example: a “protocol”…

1

2

3

4

a buffer

a sender
State explosion problem

An example: a “protocol”…

A sender

A buffer

A receiver
State explosion problem

An example: a “protocol”…

A sender

A buffer

A receiver

A server
State explosion problem

An example: a “protocol”…
State explosion problem

An example: a “protocol”…

- A sender
- A buffer
- A receiver
- A server
State explosion problem
State explosion problem

This has motivated

- symbolic methods
- heuristics:
  - on-the-fly algorithms
  - acceleration
  - partial order
  - ...
- abstraction
- ...

Model checkers exist and they work rather nicely!
Verification of compositions of embedded systems

Two difficulties:

- the composition...

- Usually we need to address verification of quantitative aspects:

  - timing constraints (Real-time systems)
  - probabilities
  - costs
  - data
  - ...

Verification of compositions of embedded systems

Two difficulties:

- the composition . . .

- Usually we need to address verification of quantitative aspects:
  - timing constraints (Real-time systems)
  - probabilities
  - costs
  - data
  - . . .

Each one may induce a complexity blow-up.

This can be measured by the structural complexity of verification problems: for ex. the composition and the state explosion problem.
This state explosion holds for synchronized systems, systems with boolean variables, 1-safe Petri nets etc.

These are “non-flat” systems.

In practice, a model $S$ is described as a non-flat system (whose operational semantics is a “flat” transition system $T$).
State explosion problem

This state explosion holds for synchronized systems, systems with boolean variables, 1-safe Petri nets etc.

→ These are “non-flat” systems.

In practice, a model $S$ is described as a non-flat system (whose operational semantics is a “flat” transition system $T$)

### Complexity of model checking non-flat systems

<table>
<thead>
<tr>
<th>Model checking</th>
<th>$T \models \Phi$</th>
<th>”non-flat syst.” $\models \Phi$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reachability</td>
<td>NLOGSPACE-C</td>
<td>PSPACE-C</td>
</tr>
<tr>
<td>$CTL$ model checking</td>
<td>P-C</td>
<td>PSPACE-C</td>
</tr>
<tr>
<td>AF $\mu$-calculus</td>
<td>P-C</td>
<td>EXPTIME-C</td>
</tr>
</tbody>
</table>

(Papadimitriou, Vardi, Kupferman, Wolper, Rabinovich, . . .)
Challenge

Define new formalisms to model and verify the embedded systems, with...

- timing constraints
- probabilities
- costs, dynamic variables
- ...
- extended specification languages
Challenge

Define new formalisms to model and verify the embedded systems, with...

- timing constraints
- probabilities
- costs, dynamic variables
- ...

- extended specification languages

- request $\Rightarrow$ $\Delta F$ grant
- request $\Rightarrow$ $\Delta F < 200$ grant
- request $\Rightarrow$ $P > 0.9 F < 200$ grant
- request $\Rightarrow$ $\langle \text{Agt}_1 \rangle P > 0.9 F < 200$ grant
Challenge

Define new formalisms to model and verify the embedded systems, with...

- timing constraints
- probabilities
- costs, dynamic variables
- ... 

- extended specification languages
  - request $\Rightarrow$ $\text{AF}$ grant
  - request $\Rightarrow$ $\text{AF}_{<200}$ grant
  - request $\Rightarrow$ $\mathbb{P}_{>0.9} F_{<200}$ grant
  - request $\Rightarrow$ $\langle \text{Agt}_1 \rangle \mathbb{P}_{>0.9} F_{<200}$ grant

without increasing (too much) the complexity!
Efficient timed model-checking

**obj:** find timed models and timed specification languages for which verification can be done efficiently.

→ To model simply timed systems

→ To use to abstract complex timed systems

→ To combine with other quantitative extensions
Outline

1. Timed specification languages

2. Timed models
   - Discrete-time models
   - Timed automata

3. Conclusion
Outline

1. Timed specification languages

2. Timed models
   - Discrete-time models
   - Timed automata

3. Conclusion
Classical verification problems

- Reachability of a control state
- \( S \sim S' \) : (bi)simulation, etc.
- \( L(S) \subseteq L(S) \) : language inclusion
- \( (S|A_T) + \) reachability : testing automata
- \( S \models \Phi \) with \( \Phi \) a temporal logic formula : model checking
Timed properties

Temporal properties:

“Any problem is followed by an alarm”

With \textbf{CTL}:

\[ \text{AG} \left( \text{problem} \Rightarrow \text{AF alarm} \right) \]
Timed properties

Temporal properties:

“Any problem is followed by an alarm”

With CTL: $\text{AG} \left( \text{problem} \Rightarrow \text{AF alarm} \right)$

Timed properties:

“Any problem is followed by an alarm in at most 10 time units”
Timed properties

Temporal properties:

“Any problem is followed by an alarm”

With CTL:

\[ \text{AG} \left( \text{problem} \Rightarrow \text{AF alarm} \right) \]

Timed properties:

“Any problem is followed by an alarm in at most 10 time units”

With TCTL:

\[ \text{AG} \left( \text{problem} \Rightarrow \text{AF}_{\leq 10} \text{ alarm} \right) \]
We use TCTL whose formulae are built from:

- atomic propositions in Prop (For ex. Alarm, Problem, ...)
- boolean combinators (\(\land, \lor, \neg\)), and
- temporal operators tagged with timing constraints: \(E \_ U_{\sim c}\) and \(A \_ U_{\sim c}\) with \(\sim \in \{<, \leq, =, \geq, >\}\) and \(c \in \mathbb{N}\).

+ all the standard abbreviations: \(AG_{\sim c}, AF_{\sim c}\) etc.
We use *TCTL* whose formulae are built from:

- atomic propositions in Prop (For ex. *Alarm*, *Problem*, ...)
- boolean combinators ($\land$, $\lor$, $\neg$), and
- temporal operators tagged with timing constraints: $E_{\sim}U_{\sim}c$ and $A_{\sim}U_{\sim}c$ with $\sim \in \{<, \leq, =, \geq, >\}$ and $c \in \mathbb{N}$.

+ all the standard abbreviations: $AG_{\sim}c$, $AF_{\sim}c$ etc.

\[
s \models E\varphi U_{\sim}c \psi \iff \exists \rho = \rho' \cdot \rho'' \in \text{Exec}(s) \text{ with } s \xrightarrow{\rho'} s' \text{ and } \text{Time} (\rho') \sim c \text{ and } s' \models \psi, \\
\text{and } s'' \models \varphi \text{ for all } s <_{\rho} s'' <_{\rho} s'
\]
1 Timed specification languages

2 Timed models
   - Discrete-time models
   - Timed automata

3 Conclusion
Questions

Studying timing constraints & complexity

Two problems:

1. Are there simpler models – with discrete time – for which model checking can be efficient?
2. Are there classes of Timed Automata which allow efficient model checking?
Timed transition systems

= a transition system + a notion of time
⇒ used to define the semantics of real-time systems

Let $\mathbb{T}$ be a time domain: $\mathbb{N}$, $\mathbb{R}_+$ or $\mathbb{Q}_+$.

Timed transition system

$\mathcal{T} = \langle S, s_0, \rightarrow, l \rangle$

- $S$ is an (possibly infinite) set of states, $s_0 \in S$
- $\rightarrow \subseteq S \times \mathbb{T} \times S$
- $l : S \rightarrow 2^{\text{Prop}}$ : assigns atomic propositions to states
Timed transition systems

= a transition system + a notion of time
⇒ used to define the semantics of real-time systems

Let $\mathbb{T}$ be a time domain: $\mathbb{N}$, $\mathbb{R}_+$ or $\mathbb{Q}_+$.

Timed transition system

$\mathcal{T} = \langle S, s_0, \rightarrow, I \rangle$

- $S$ is an (possibly infinite) set of states, $s_0 \in S$
- $\rightarrow \subseteq S \times \mathbb{T} \times S$
- $I : S \rightarrow 2^{\text{Prop}}$: assigns atomic propositions to states

Every finite run $\sigma$ in $\mathcal{T}$ has a finite duration $\text{Time}(\sigma)$. 
Outline

1 Timed specification languages

2 Timed models
   - Discrete-time models
   - Timed automata

3 Conclusion
It is possible to use classical Kripke structures as timed models. There is no inherent concept of time: time elapsing is encoded by events.

For example:

- each transition = one time unit (Emerson et al.)
- or: a “tick” proposition labels states where one t.u. elapses.

Timed model checking can be efficient (polynomial-time)!

Durational Transition Graphs extend these models without loosing efficiency.
Durational Transition Graph

- **Init state**
  - Transition: [0,∞)
  - Duration: [7,45]
  - Transitions:
    - To **New Card**: [25,50]
    - To **Wait for Pincode**: [0,∞)

- **New Card**
  - Duration: [25,50]
  - Transitions:
    - To **Wait for Pincode**: [25,50]
    - To **Code OK**: [25,50]

- **Wait for Pincode**
  - Duration: [25,50]
  - Transitions:
    - To **Read code**: [25,50]
    - To **Bad code**: [0,7]

- **Read code**
  - Transition: [0,∞)
  - Duration: [10;15]
  - Transitions:
    - To **Bad code**: [0,7]
    - To **Return card**: [5,10]

- **Code OK**
  - Duration: [0,7]
  - Transitions:
    - To **Ask amount**: [0,7]

- **Ask amount**
  - Duration: [0,7]
  - Transitions:
    - To **Wait for Pincode**: [0,7]

- **Return card**
  - Duration: [5,10]
  - Transitions:
    - To **Wait for Pincode**: [0,7]

- **Money+card**
  - Duration: [0,366]
  - Transitions:
    - To **Init state**: [0,∞)
    - To **New Card**: [50,110]
\( \mathbb{T} = \mathbb{N} \)

A durational transition graph \( S \) is \( \langle Q, R, I \rangle \) where

- \( Q \) is a (finite) set of states,
- \( R \subseteq Q \times I \times Q \) is a total transition relation with duration
- \( I : Q \rightarrow 2^{\text{Prop}} \) labels every state with a subset of \( \text{Prop} \).

\( I = \) set of intervals “\([n, m]\)” or “\([n, \infty)\)” (with \( n, m \in \mathbb{N} \))

\( q \xrightarrow{[n,m]} q' : \) “moving from \( q \) to \( q' \) takes some duration \( d \) in \( [n, m] \).”

Two semantics are possible...
Two semantics for DTGs

Consider the run: $\text{Init\_State} \xrightarrow{15} \text{New\_Card} \xrightarrow{0} \text{Wait\_for\_Pin\ C.} \xrightarrow{20} \ldots$
Two semantics for DTGs

Consider the run: $\text{Init\_State} \overset{15}{\rightarrow} \text{New\_Card} \overset{0}{\rightarrow} \text{Wait\_for\_Pin \ C} \overset{20}{\rightarrow} \ldots$

- **Jump** semantics:
  - date: 0 $\leadsto$ Init\_State
  - date: 15 $\leadsto$ New\_Card no intermediary states!
  - $\ldots$

⊕ Simple semantics, no extra states.
 Nodo always natural. Difficult to synchronize two DTGs.
Two semantics for DTGs

Consider the run: Init_State $\xrightarrow{15}$ New_Card $\xrightarrow{0}$ Wait_for_Pin_C. $\xrightarrow{20}$ ...

- **Jump** semantics:
  - date: 0 $\leadsto$ Init_State
  - date: 15 $\leadsto$ New_Card no intermediary states!
  - ...

  ⊞ Simple semantics, no extra states.
  ⊞ Not always natural. Difficult to synchronize two DTGs.

- **Continuous** semantics:
  - date: 0...14 $\leadsto$ Init_state
  - date: 15 $\leadsto$ New_card
  - date: 15...34 $\leadsto$ Wait_for_Pin_C
  - ...

jump vs continuous semantics
jump vs continuous semantics

DTG $S$

jump sem. of $S$:
jump vs continuous semantics

DTG $S$

jump sem. of $S$:

$q, 0$ \(\xrightarrow{1} q, 1\) \(\xrightarrow{1} q, 2\) \(\xrightarrow{1} q, 3\) \(\xrightarrow{1} \ldots\)

$(q, i)$ : “$i$ time units have already been spent in $q$”

continuous sem. of $S$:

$(q, 0)$ \(\xrightarrow{1} q, 1\) \(\xrightarrow{1} q, 2\) \(\xrightarrow{1} q, 3\) \(\xrightarrow{1} \ldots\)

$(q, i)$ : “$i$ time units have already been spent in $q$”
Outline

1 Timed specification languages

2 Timed models
   - Discrete-time models
   - Timed automata

3 Conclusion
\[ A = \text{an automaton (locations and transitions) } + \text{clocks} \]
Timed automata - definition [Alur & Dill]

\( \mathcal{A} = \text{an automaton (locations and transitions)} + \text{clocks} \)

**Clocks** progress synchronously with time (Time domain = \( \mathbb{R}_+ \))

**Transitions**: \( \ell \xrightarrow{g,a,r} \ell' \in T \) with:

- \( g \) is the guard,
- \( a \) is the label,
- \( r \) is the set of clocks to be reset to 0
\[ A = \text{an automaton (locations and transitions) + clocks} \]

Clocks progress synchronously with time (Time domain = \( \mathbb{R}_+ \))

Transitions: \( \ell \xrightarrow{g,a,r} \ell' \in T \) with:
- \( g \) is the guard,
- \( a \) is the label,
- \( r \) is the set of clocks to be reset to 0

Semantics:

▲ States: \((\ell, v)\) where \( v \) is a valuation for the clocks
$A = \text{an automaton (locations and transitions) } + \text{ clocks}$

Clocks progress synchronously with time (Time domain = $\mathbb{R}_+$)

Transitions: $\ell \xrightarrow{g,a,r} \ell' \in T$ with:
• $g$ is the guard,
• $a$ is the label,
• $r$ is the set of clocks to be reset to 0

Semantics:

▲ States: $(\ell, v)$ where $v$ is a valuation for the clocks

▲ Action transition:
$(\ell, v) \xrightarrow{a} (\ell', v') \iff \exists \ell \xrightarrow{g,a,r} \ell' \in A, v \models g, v' = v[r \leftarrow 0]$
Timed automata - definition  [Alur & Dill]

\[ A = \text{an automaton (locations and transitions) + clocks} \]

Clocks progress synchronously with time (Time domain = \( \mathbb{R}_+ \))

Transitions: \( \ell \xrightarrow{g,a,r} \ell' \in T \) with:
- \( g \) is the guard,
- \( a \) is the label,
- \( r \) is the set of clocks to be reset to 0

Semantics:

▲ States: \((\ell, v)\) where \( v \) is a valuation for the clocks

▲ Action transition:
\[ (\ell, v) \xrightarrow{a} (\ell', v') \iff \begin{cases} \exists \ell \xrightarrow{g,a,r} \ell' \in A, \\ v \models g, \quad v' = v[r \leftarrow 0] \end{cases} \]

▲ Delay transition:
\[ \forall t \in \mathbb{R}_+, (\ell, v) \xrightarrow{t} (\ell, v + t) \iff \forall 0 \leq t' \leq t, v + t' \models \text{Inv}(\ell) \]
Model checking for TA

$\mathcal{A}$ defines an **infinite** timed transition system.

Decision procedures are based on the **region graph** technique (Alur, Courcoubetis, Dill)

From $\mathcal{A}$ and $\Phi$, one defines an equivalence $\equiv_{\mathcal{A},\Phi}$ s.t.:

- $\nu \equiv_{\mathcal{A},\Phi} \nu' \Rightarrow \left( (\ell, \nu) \models \Phi \text{ iff } (\ell, \nu') \models \Phi \right)$

- $\mathbb{R}_+^x/\equiv_{\mathcal{A},\Phi}$ is finite

A **region** = An equivalence class of $\equiv_{\mathcal{A},\Phi}$

We can reduce $\mathcal{A} \models \Phi$ to $\left( \mathcal{A} \times \mathbb{R}_+^x/\equiv_{\mathcal{A},\Phi} \right) \models \Phi$

...and use a standard model checking algorithm.
Model checking for TA

\( \mathcal{A} \) defines an infinite timed transition system.

Decision procedures are based on the region graph technique (Alur, Courcoubetis, Dill)

From \( \mathcal{A} \) and \( \Phi \), one defines an equivalence \( \equiv_{\mathcal{A},\Phi} \) s.t.:

1. \( v \equiv_{\mathcal{A},\Phi} v' \Rightarrow (\ell, v) \models \Phi \iff (\ell, v') \models \Phi \)
2. \( \mathbb{R}_+^X/\equiv_{\mathcal{A},\Phi} \) is finite

A region = An equivalence class of \( \equiv_{\mathcal{A},\Phi} \)

We can reduce \( \mathcal{A} \models \Phi \) to \( (\mathcal{A} \times \mathbb{R}_+^X/\equiv_{\mathcal{A},\Phi}) \models \Phi \)

...and use a standard model checking algorithm.

! The size of \( \mathbb{R}_+^X/\equiv_{\mathcal{A},\Phi} \) is in \( O(|X|! \cdot M^{|X|}) \)!
The region abstraction

\[ X = \{x, y\} \]

\[ M_x = 3 \text{ and } M_y = 2 \]
The region abstraction

- “compatibility” between regions and constraints $x \sim k$ and $y \sim k$

$X = \{x, y\}$

$M_x = 3$ and $M_y = 2$
The region abstraction

$X = \{x, y\}$

$M_x = 3$ and $M_y = 2$

- “compatibility” between regions and constraints $x \sim k$ and $y \sim k$
- “compatibility” between regions and time elapsing
The region abstraction

- "compatibility" between regions and constraints $x \sim k$ and $y \sim k$
- "compatibility" between regions and time elapsing

$X = \{x, y\}$
$M_x = 3$ and $M_y = 2$
The region abstraction

- "compatibility" between regions and constraints $x \sim k$ and $y \sim k$
- "compatibility" between regions and time elapsing

$X = \{x, y\}$
$M_x = 3$ and $M_y = 2$
The region abstraction

$X = \{ x, y \}$

$M_x = 3$ and $M_y = 2$

region defined by

$I_x = ]1; 2[\), $I_y = ]0; 1[\)

$\{ x \} < \{ y \}$

- “compatibility” between regions and constraints $x \sim k$ and $y \sim k$
- “compatibility” between regions and time elapsing
The region abstraction

\[ X = \{x, y\} \]
\[ M_x = 3 \text{ and } M_y = 2 \]

Region defined by
\[ I_x = ]1; 2[ \text{, } I_y = ]0; 1[ \]
\[ \{x\} < \{y\} \]

Successor by delay transition:
\[ I_x = ]1; 2[ \text{, } I_y = [2; 2] \]

- “compatibility” between regions and constraints \( x \sim k \) and \( y \sim k \)
- “compatibility” between regions and time elapsing
An example [AD 90’s]
Complexity of timed verification

- Reachability in TA is **PSPACE-C** (Alur & Dill)
- Reachability in TA with three clocks is **PSPACE-C** (Courcoubetis & Yannakakis)
- Reachability in TA with constants in \{0, 1\} is **PSPACE-C** (Courcoubetis & Yannakakis)
- Model checking **Timed CTL** over TA is **PSPACE-C** (Alur, Courcoubetis & Dill)
- Model checking **AF Timed \( \mu \)-calculus** over TA is **EXPTIME-C** (Aceto & Laroussinie)

\( \text{(But tools (Kronos, UppAal) exist and have been applied successfully for verifying industrial case studies.)} \)
One/two clock timed automata

1. Model checking $TCTL$ on 1C-TA is PSPACE-complete.
2. Reachability in 1C-TA is NLOGSPACE-complete.
3. Model checking $TCTL_{\leq, \geq}$ on 1C-TA is P-complete.

1. Reachability in 2C-TA is NP-hard.
2. Model checking $CTL$ on 2C-TA is PSPACE-complete.
Outline

1. Timed specification languages

2. Timed models
   - Discrete-time models
   - Timed automata

3. Conclusion
## Conclusion

<table>
<thead>
<tr>
<th></th>
<th>discrete time</th>
<th>dense time</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTG $^{0/1}$</td>
<td>DTG [LMS06]</td>
<td>1C-TA [LMS04]</td>
</tr>
<tr>
<td>[LST03]</td>
<td>jump / cont.</td>
<td>2C-TA [LMS04]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TA [ACD93]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[CY92]</td>
</tr>
<tr>
<td>Reachability</td>
<td><strong>NLOGSPACE-C</strong></td>
<td>NP-hard</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PSPACE-C</td>
</tr>
<tr>
<td>$TCTL_{\leq,\geq}$</td>
<td><strong>P-complete</strong></td>
<td>PSPACE-C</td>
</tr>
<tr>
<td>$TCTL$</td>
<td>P-compl.</td>
<td>$\Delta_2^p$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PSPACE-complete</td>
</tr>
<tr>
<td>$TCTL_c$</td>
<td></td>
<td>PSPACE-complete</td>
</tr>
</tbody>
</table>
• No efficient algorithm for "TCTL+clocks".
• As soon as there are two clocks, a complexity blow-up occurs for any verification problem.
• For TCTL, model-checking may be efficient only for the simple DTGs with durations in \{0, 1\}.

• For TCTL\(_{\leq, \geq}\), it is possible to have efficient model-checking algorithm for any kind of DTG and 1C-TA.
• The previous result can be extended to probabilistic timed systems (Probabilistic DTG and 1-clock Probabilistic TA).