A unified formalism for monoprocessor schedulability analysis under uncertainty

Étienne André

LIPN, Université Paris 13, CNRS, France
Context: Verifying critical real-time systems

real-time systems:
- Systems for which not only the correctness but also the timely answer is important
Context: Verifying critical real-time systems

- **Critical real-time systems:**
  - Systems for which not only the correctness but also the timely answer is important
  - Failures (in correctness or timing) may result in dramatic consequences
Real-time system

A real-time system is made of a set of tasks to execute on a processor.
Real-time system

A real-time system is made of a set of tasks to execute on a processor.

A task is characterized by:
- \( B \): its best-case execution time
- \( W \): its worst-case execution time
- \( D \): its relative deadline
Real-time system

A real-time system is made of a set of tasks to execute on a processor.

A task is characterized by:
- \( B \): its best-case execution time
- \( W \): its worst-case execution time
- \( D \): its relative deadline

Tasks have instances that can be activated...
- periodically
- sporadically (usually with a minimum interarrival time)
- or following more complex patterns (e.g., activation following the completion of another task instance)
Activated instances are **queued**
When the processor is idle, which instance in the queue should be executed?
→ decision made by the **scheduler**
Activated instances are **queued**
When the processor is idle, which instance in the queue should be executed?
\[\rightarrow\] decision made by the **scheduler**

The scheduler can be **preemptive**

- The execution of a lower priority task can be **interrupted** when a instance of a task with **higher priority** is activated
- After completion of the higher priority task, the lower priority task resumes
**Example: preemptive fixed priority scheduler (FPS)**

<table>
<thead>
<tr>
<th>Task</th>
<th>B</th>
<th>W</th>
<th>D</th>
<th>priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>low</td>
</tr>
<tr>
<td>$t_2$</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>high</td>
</tr>
</tbody>
</table>

![Diagram showing the execution of tasks $t_1$ and $t_2$.]$t_1$ misses its deadline.
Example: preemptive fixed priority scheduler (FPS)

<table>
<thead>
<tr>
<th>Task</th>
<th>B</th>
<th>W</th>
<th>D</th>
<th>priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>low</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>high</td>
</tr>
</tbody>
</table>

![Diagram of task execution]

- \( t_1 \) misses its deadline.

\( \text{É. André (Université Paris 13)} \)
Example: preemptive fixed priority scheduler (FPS)

<table>
<thead>
<tr>
<th>Task</th>
<th>B</th>
<th>W</th>
<th>D</th>
<th>priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>low</td>
</tr>
<tr>
<td>$t_2$</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>high</td>
</tr>
</tbody>
</table>
Example: preemptive fixed priority scheduler (FPS)

<table>
<thead>
<tr>
<th>Task</th>
<th>B</th>
<th>W</th>
<th>D</th>
<th>priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>low</td>
</tr>
<tr>
<td>$t_2$</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>high</td>
</tr>
</tbody>
</table>
Example: preemptive fixed priority scheduler (FPS)

<table>
<thead>
<tr>
<th>Task</th>
<th>B</th>
<th>W</th>
<th>D</th>
<th>priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>low</td>
</tr>
<tr>
<td>$t_2$</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>high</td>
</tr>
</tbody>
</table>
Example: preemptive fixed priority scheduler (FPS)

<table>
<thead>
<tr>
<th>Task</th>
<th>B</th>
<th>W</th>
<th>D</th>
<th>priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>low</td>
</tr>
<tr>
<td>$t_2$</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>high</td>
</tr>
</tbody>
</table>

Task $t_1$ misses its deadline
Example: earliest deadline first (EDF)

<table>
<thead>
<tr>
<th>Task</th>
<th>B</th>
<th>W</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>$t_2$</td>
<td>2</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>
Example: earliest deadline first (EDF)

<table>
<thead>
<tr>
<th>Task</th>
<th>B</th>
<th>W</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>$t_2$</td>
<td>2</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

Diagram showing the tasks $t_1$ and $t_2$ on a timeline with their respective deadlines and weights.
**Example: earliest deadline first (EDF)**

<table>
<thead>
<tr>
<th>Task</th>
<th>B</th>
<th>W</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>2</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>
Example: shortest job first (SJF)

<table>
<thead>
<tr>
<th>Task</th>
<th>B</th>
<th>W</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>$t_2$</td>
<td>2</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>
Example: shortest job first (SJF)

<table>
<thead>
<tr>
<th>Task ( t_1 )</th>
<th>B</th>
<th>W</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_1 )</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>2</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

Two tasks, \( t_1 \) and \( t_2 \), with different deadlines.
Example: shortest job first (SJF)

<table>
<thead>
<tr>
<th>Task</th>
<th>B</th>
<th>W</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_1$</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>$t_2$</td>
<td>2</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

Task $t_1$ misses its deadline
Schedulability analysis

Definition (schedulability analysis)

Given a real-time system and a scheduling policy, the schedulability analysis checks whether the system is schedulable (i.e., all tasks meet their deadline) for all possible behaviors.
Schedulability analysis

Definition (schedulability analysis)

Given a real-time system and a scheduling policy, the schedulability analysis checks whether the system is schedulable (i.e., all tasks meet their deadline) for all possible behaviors.

All possible behaviors:

- Depends on the periods, interarrival rates, dependencies between tasks...
Problem: schedulability analysis under uncertainty

Problem: what if some timing constants (deadlines, execution times, periods, interarrival times...) are unknown or known with a limited precision?
Problem: schedulability analysis under uncertainty

Problem: what if some timing constants (deadlines, execution times, periods, interarrival times...) are unknown or known with a limited precision?

Objective

Propose a framework for the monoprocessor schedulability analysis of real-time systems under uncertainty
Outline

1. Parametric task automata
2. Decidability and undecidability
3. Schedulability under uncertainty
4. Conclusion and perspectives
Outline: Parametric task automata

1. Parametric task automata
   - Task automata
   - Parametric task automata

2. Decidability and undecidability

3. Schedulability under uncertainty

4. Conclusion and perspectives
Task automaton (TaskA)

- Finite state automaton (sets of locations)
Task automaton (TaskA)

- Finite state automaton (sets of locations and actions)
Task automaton (TaskA)

- Finite state automaton (sets of *locations* and *actions*) with a set $X$ of *clocks* as in timed automata [Alur and Dill, 1994]
- Real-valued variables evolving linearly at the same rate
Task automaton (TaskA)

- Finite state automaton (sets of locations and actions) with a set $X$ of clocks as in timed automata [Alur and Dill, 1994]
  - Real-valued variables evolving linearly at the same rate

- Features
  - Location invariant: property to be verified to stay at a location

![Task automaton diagram](image-url)
Task automaton (TaskA)

- Finite state automaton (sets of locations and actions) with a set $X$ of clocks as in timed automata [Alur and Dill, 1994]
  - Real-valued variables evolving linearly at the same rate

- Features
  - Location invariant: property to be verified to stay at a location
  - Transition guard: property to be verified to enable a transition

\[ x = 20 \]
\[ b \]
\[ l_3 \]
Task automaton (TaskA)

- Finite state automaton (sets of locations and actions) with a set $X$ of clocks as in timed automata [Alur and Dill, 1994]
  - Real-valued variables evolving linearly at the same rate

- Features
  - Location invariant: property to be verified to stay at a location
  - Transition guard: property to be verified to enable a transition
  - Clock reset: some of the clocks can be set to 0 at each transition
Task automaton (TaskA)

- Finite state automaton (sets of locations and actions) with a set \( X \) of clocks as in timed automata [Alur and Dill, 1994]
- Real-valued variables evolving linearly at the same rate
- A set \( \mathcal{T} \) of tasks [Norström et al., 1999, Fersman et al., 2007]

- Features
  - Location invariant: property to be verified to stay at a location
  - Transition guard: property to be verified to enable a transition
  - Clock reset: some of the clocks can be set to 0 at each transition
Concrete semantics of task automata

- **Concrete state** of a TaskA: triple \((l, w, q)\), where
  - \(l\) is a location,
  - \(w\) is a valuation of each clock
  - \(q\) is a task queue made of instances of \(T\)
    - Instance: task, remaining BCET and WCET, remaining deadline

Example: \(\left(l_1, (x=1.2, \ y=3.7), [(t_0, 0, 0.5, 1.5), (t_1, 4, 4, 18.5)(t_1, 4, 4, 19.5)] \right)\)

- **Concrete run**: alternating sequence of **concrete states** and **actions** or **time elapse** according to a given scheduler
Parametric task automaton (PTaskA)

- Task automaton

<table>
<thead>
<tr>
<th>Task</th>
<th>B</th>
<th>W</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_0 )</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>( t_1 )</td>
<td>4</td>
<td>4</td>
<td>20</td>
</tr>
<tr>
<td>( t_2 )</td>
<td>0</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>( t_3 )</td>
<td>2</td>
<td>2</td>
<td>10</td>
</tr>
</tbody>
</table>

Priorities: \( t_0 > t_2 > t_1 > t_3 \)
Parametric task automaton (PTaskA)

- Task automaton extended with a set $P$ of timing parameters, that can be used
  - in the automaton, and/or
  - in the tasks $B$, $W$ and $D$

![Diagram showing states and transitions of the Parametric task automaton (PTaskA) with parameters $x$, $p$, and priorities $t_0 > t_2 > t_1 > t_3$.]

<table>
<thead>
<tr>
<th>Task</th>
<th>$B$</th>
<th>$W$</th>
<th>$D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>$t_1$</td>
<td>4</td>
<td>4</td>
<td>20</td>
</tr>
<tr>
<td>$t_2$</td>
<td>0</td>
<td>1</td>
<td>$p'$</td>
</tr>
<tr>
<td>$t_3$</td>
<td>2</td>
<td>2</td>
<td>10</td>
</tr>
</tbody>
</table>
Problems of interest

Parametric task automata can model real-time systems... that are periodic, sporadic, or correspond to more complex behaviors with unknown deadlines, periods, execution times or known with limited precision
Problems of interest

Parametric task automata can model real-time systems...

- that are periodic, sporadic, or correspond to more complex behaviors
- with unknown deadlines, periods, execution times
- or known with limited precision

Some problems of interest

- For what values of the parameters is the system schedulable?
- Or even can we find at least one such valuation?
- Or is the system robustly schedulable? [Markey, 2011]
Outline: Decidability and undecidability

1. Parametric task automata
2. Decidability and undecidability
3. Schedulability under uncertainty
4. Conclusion and perspectives
Decision problem

**Schedulability-emptiness problem:**

**Input:** A PTaskA $A$ and a scheduling strategy $Sch$

**Problem:** is the set of valuations $v$ for which $v(A)$ is schedulable for strategy $Sch$ empty?
An undecidability result

Theorem (Undecidability)

*The schedulability-emptiness problem is undecidable for general PTaskA.*
An undecidability result

**Theorem (Undecidability)**

The schedulability-emptiness problem is undecidable for general \( \text{PTaskA} \).

Two reasons:

- Parametric task automata are at least as expressive as parametric timed automata [Alur et al., 1993] for which most non-trivial problems are undecidable [André, 2017].

- The schedulability of general non-parametric task automata is undecidable [Fersman et al., 2007] (in particular when using preemption).
Restricting a bit the formalism

Definition

A PTaskA has **schedulable-bounded parameters** if, for each task \( t \), its worst-case execution time \( W \) is bounded in \([a, \infty)\) or \([a, b]\) with \( a > 0 \), and its deadline \( D \) is bounded in \([a, b]\), with \( a, b \geq 0 \).

Necessary to bound the task queue
Restricting a bit the formalism

**Definition**

A PTaskA has **schedulable-bounded parameters** if, for each task \( t \), its worst-case execution time \( W \) is bounded in \( [a, \infty) \) or \( [a, b] \) with \( a > 0 \), and its deadline \( D \) is bounded in \( [a, b] \), with \( a, b \geq 0 \).

Necessary to bound the task queue

**Definition**

A PTaskA is an **L/U-PTaskA** if its parameters set is partitioned into lower-bound parameters (i.e., of the form \( p < x \) or \( p \leq x \)) and upper-bound parameters (i.e., of the form \( p > x \) or \( p \geq x \)).

Similar to L/U-parametric timed automata

[Hune et al., 2002]
A decidability result

Theorem (Decidability)

The schedulability-emptiness problem is **decidable** for L/U-PTaskAs with schedulable-bounded parameters

1. for non-preemptive FPS and SJF, and
2. non-preemptive EDF without parametric deadlines.
A decidability result

Theorem (Decidability)

The schedulability-emptiness problem is **decidable** for L/U-PTaskAs with schedulable-bounded parameters

1. for non-preemptive FPS and SJF, and
2. non-preemptive EDF without parametric deadlines.

Proof idea

Reusing the encoding of [Norström et al., 1999, Fersman et al., 2007] together with a decidability result for L/U-parametric timed automata proved in [André, 2017]
Outline: Schedulability under uncertainty

1. Parametric task automata

2. Decidability and undecidability

3. Schedulability under uncertainty
   - Parametric schedulability
   - Implementation in IMITATOR
   - Examples of analyses

4. Conclusion and perspectives
Parametric schedulability with parametric TaskA

Parametric schedulability reduces to reachability synthesis

- “Synthesize parameter valuations for which a deadline violation is reachable”
- Transform to parametric stopwatch automata [Sun et al., 2013]
  - The PTaskA itself can be seen as a parametric timed automaton
  - Any common scheduler can be transformed into a parametric stopwatch automaton
  - The system is made of the synchronous composition of both
Parametric schedulability with parametric TaskA

Parametric schedulability reduces to reachability synthesis

- “Synthesize parameter valuations for which a deadline violation is reachable”
- Transform to parametric stopwatch automata [Sun et al., 2013]
  - The PTaskA itself can be seen as a parametric timed automaton
  - Any common scheduler can be transformed into a parametric stopwatch automaton
  - The system is made of the synchronous composition of both

Undecidable in general... but we adopt a pragmatic approach and can use semi-algorithms or approximations
A practical encoding

In practice, we use extensions of parametric stopwatch automata

- Discrete global integer-valued variables (to model the queue)
- Extensive use of stopwatches
  - Also helps to reduce the state space!

Automatic translation of the scheduler into the IMITATOR input format
IMITATOR

- A tool for modeling and verifying real-time systems with unknown constants modeled with parametric timed automata [Alur et al., 1993]

- Communication through (strong) broadcast synchronization
- Rational-valued shared discrete variables
- Stopwatches, to model schedulability problems with preemption

Verification
- Computation of the symbolic state space
- (non-Zeno) parametric model checking (using a subset of TCTL)
- Language and trace preservation, and robustness analysis
- Parametric deadlock-freeness checking
IMITATOR
Under continuous development since 2008 [André et al., 2012]

A library of benchmarks
- Communication protocols
- Schedulability problems
- Asynchronous circuits
- ...and more

Free and open source software: Available under the GNU-GPL license
IMITATOR

Under continuous development since 2008 [André et al., 2012]

A library of benchmarks
- Communication protocols
- Schedulability problems
- Asynchronous circuits
- ...and more

Free and open source software: Available under the GNU-GPL license

Try it!

www.imitator.fr
Parametric schedulability analysis

For which values of $p$ and $p'$ is the system schedulable?
**Parametric schedulability analysis**

For which values of $p$ and $p'$ is the system schedulable?

\[
p \geq 9 \land p' \geq 2 \land p + p' \geq 23 \\
\lor \\
p \geq 9 \land p' \geq 3 \land p + p' < 23
\]
Robustness analysis

Definition

A real-time system is **robustly schedulable** if it remains schedulable even for infinitesimal variations of the timing constants (without parameters).
Robustness analysis

Definition

A real-time system is robustly schedulable if it remains schedulable even for infinitesimal variations of the timing constants (without parameters)

Methodology:

1. Replace any guard $x \leq c$ with $x \leq c + \epsilon$
2. Replace any guard $x \geq c$ with $x \geq c - \epsilon$
3. Synthesize admissible valuations for $\epsilon$
4. Check whether $\epsilon$ may be different from 0
Robustness analysis

Definition

A real-time system is robustly schedulable if it remains schedulable even for infinitesimal variations of the timing constants (without parameters).

Methodology:

1. Replace any guard $x \leq c$ with $x \leq c + \epsilon$
2. Replace any guard $x \geq c$ with $x \geq c - \epsilon$
3. Synthesize admissible valuations for $\epsilon$
4. Check whether $\epsilon$ may be different from 0

For our TaskA (fixing $p = 10$ and $p' = 4$), we get $\epsilon = 0$
For which values of $p$ is the system robustly schedulable?
Parametric schedulability and robustness

For which values of $p$ is the system robustly schedulable?

\[
p \geq 9 \land \epsilon = 0 \lor \\
\epsilon \leq \frac{2}{5} \land p \geq 20 + 5\epsilon \land p \geq 19 + 8\epsilon
\]
Parametric schedulability and robustness

For which values of $p$ is the system robustly schedulable?

$p \geq 9 \land \epsilon = 0$

$\lor$

$\epsilon \leq \frac{2}{5} \land p \geq 20 + 5\epsilon \land p \geq 19 + 8\epsilon$

We even know by how the system is robust (depending on $p$ and $\epsilon$)
Outline: Conclusion and perspectives

1. Parametric task automata
2. Decidability and undecidability
3. Schedulability under uncertainty
4. Conclusion and perspectives
Conclusion

Parametric task automata

- A unified, compact and expressive formalism to model and verify real-time systems under uncertainty

- Some undecidability results
- Some decidability results

Allow for

- Parametric schedulability
- Robustness analysis
- Robust parametric schedulability

Implementation in IMITATOR using an automated translation of the scheduler
Perspectives

Fill the decidability gap

■ Still some unknown between decidability and undecidability results

■ Our examples all fit into the undecidability cases... but analysis terminates with an exact answer

Design patterns for TaskA

■ Allow to build periodic tasks, sporadic tasks... easily using predefined building blocks

Multiprocessor

■ Extend the formalism to multiprocessor schedulability analysis

Mixed-criticality scheduling
Bibliography


References

A theory of timed automata.

Parametric real-time reasoning.
In *STOC*, pages 592–601. ACM.

André, É. (2017).
What’s decidable about parametric timed automata?

IMITATOR 2.5: A tool for analyzing robustness in scheduling problems.

Formal timing analysis of mixed music scores.
In *ICMC 2013 (International Computer Computer Music Conference)*.
References II


Additional explanation
Explanation for the 4 pictures in the beginning

Allusion to the Northeast blackout (USA, 2003)
Computer bug
Consequences: 11 fatalities, huge cost
(Picture actually from the Sandy Hurricane, 2012)

Error screen on the earliest versions of Macintosh

Allusion to the sinking of the Sleipner A offshore platform (Norway, 1991)
No fatalities
Computer bug: inaccurate finite element analysis modeling
(Picture actually from the Deepwater Horizon Offshore Drilling Platform)

Allusion to the MIM-104 Patriot Missile Failure (Iraq, 1991)
28 fatalities, hundreds of injured
Computer bug: software error (clock drift)
(Picture of an actual MIM-104 Patriot Missile, though not the one of 1991)
Some success stories with IMITATOR

- Modeled and verified an **asynchronous memory circuit** by ST-Microelectronics
  - Project ANR Valmem

- Parametric schedulability analysis of a prospective architecture for the flight control system of the **next generation of spacecrafts** designed at ASTRIUM Space Transportation [Fribourg et al., 2012]

- Formal timing analysis of **music scores** [Fanchon and Jacquemard, 2013]

- Solution to a challenge related to a **distributed video processing system** by Thales
Licensing
Source of the graphics used I

Title: Hurricane Sandy Blackout New York Skyline
Author: David Shankbone
Source: https://commons.wikimedia.org/wiki/File:Hurricane_Sandy_Blackout_New_York_Skyline.jpg
License: CC BY 3.0

Title: Sad mac
Author: Przemub
Source: https://commons.wikimedia.org/wiki/File:Sad_mac.png
License: Public domain

Title: Deepwater Horizon Offshore Drilling Platform on Fire
Author: ideum
Source: https://secure.flickr.com/photos/ideum/4711481781/
License: CC BY-SA 2.0

Title: DA-SC-88-01663
Author: imcomkorea
Source: https://secure.flickr.com/photos/imcomkorea/3017886760/
License: CC BY-NC-ND 2.0
Source of the graphics used II
License of this document

This presentation can be published, reused and modified under the terms of the license Creative Commons Attribution-ShareAlike 4.0 Unported (CC BY-SA 4.0)

(LATeX source available on demand)

Author: Étienne André

https://creativecommons.org/licenses/by-sa/4.0/