# A unified formalism for monoprocessor schedulability analysis under uncertainty*

Étienne André

Université Paris 13, LIPN, CNRS, UMR 7030, F-93430, Villetaneuse, France

**Abstract.** The schedulability analysis of real-time systems (even on a single processor) is a very difficult task, which becomes even more complex (or undecidable) when periods or deadlines become uncertain. In this work, we propose a unified formalism to model monoprocessor schedulability problems with several types of tasks (periodic, sporadic, or more complex), most types of schedulers (including EDF, FPS and SJF), with or without preemption, in the presence of uncertain timing constants. Although the general case is undecidable, we exhibit a large decidable subclass. We demonstrate the expressive power of our formalism on several examples, allowing also for robust schedulability.

**Keywords:** schedulability analysis, real-time systems, timing parameters

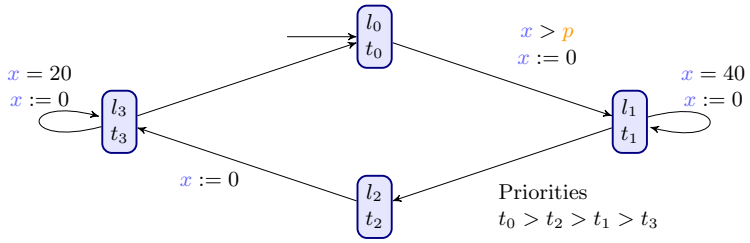
## 1 Introduction

The schedulability problem for real-time systems consists in checking whether, for a given set of tasks bound by some constraints (precedence between tasks, periods...) and for a given scheduler, all tasks can finish their computation before their relative deadline. This problem is a very delicate task, even on a single processor, and becomes even more complex (or undecidable) when periods or deadlines become unknown or subject to uncertainty.

Timed automata (TAs) [AD94] are a powerful formalism to model and verify timed concurrent systems, by extending finite-state automata with continuous variables ("clocks") that can be compared to constants in transitions ("guards") and locations ("invariants") or reset along transitions. Schedulability analysis with stopwatch automata (an extension of TAs) was proposed in [AM02]: although stopwatch automata are an undecidable formalism in general [CL00], jobshop scheduling using stopwatch automata is still possible [AM02].

Task automata (TaskA) were introduced in [NWY99] as an extension of TAs where discrete transitions can be labeled with tasks, that can have a worst

* This is the author version of the manuscript of the same name published in the proceedings of the International Workshop on Formal Methods for Industrial Critical Systems and Automated Verification of Critical Systems (FMICS-AVoCS 2017). The final version is available at http://dx.doi.org/10.1007/978-3-319-67113-0_7. This work is partially supported by the ANR national research program PACS (ANR-14-CE28-0002).




| Task | $B$ | $W$ | $D$ |
|---|---|---|---|
| $t_0$ | 0 | 1 | 2 |
| $t_1$ | 4 | 4 | 20 |
| $t_2$ | 0 | 1 | $p'$ |
| $t_3$ | 2 | 2 | 10 |

Fig. 1: Encoding semi-periodic and sporadic tasks using a PTaskA

case execution time and a deadline. Thanks to the expressive power of TAs, this formalism is richer than the traditional periodic tasks (characterized by their period) or sporadic tasks (characterized only by their minimal inter-arrival time). In addition, the schedulability problem ("is the TaskA schedulable for a given strategy?") is decidable for non-preemptive strategies, i. e., without the ability of the scheduler to temporarily suspend a task for a more urgent one. This formalism is enriched and slightly modified in [FKPY07], where the tasks become associated with locations (instead of transitions) and are also characterized with a minimum execution time. Although the schedulability problem for task automata of [FKPY07] is undecidable in general (for some preemptive strategies), the decidable case is large, including all non-preemptive strategies, and all strategies without task feedback (i. e., the precise finishing time of a task influences the release of another one) or when best-case and worst-case computation times of tasks are equal to each other.

*Example 1.* Consider the TaskA in Fig. 1 (from [FKPY07, Fig.2b]) with two tasks $t_1$ and $t_3$ which are similar to periodic tasks, though they alternate between each other. $D$, $B$ and $W$ denote the deadline, the best and worst-case execution time of each task, respectively. In addition, two sporadic tasks ($t_0$ and $t_2$) are interleaved between $t_1$ and $t_3$. Every time location $l_i$ is entered, an instance of $t_i$ is created. For preemptive fixed priority scheduling (FPS), the tasks ordered by decreasing priority order are $t_0 > t_2 > t_1 > t_3$. TaskA can help to solve the schedulability problem: e. g., for $p = 10$ and $p' = 4$ and FPS strategy, is the system schedulable?

*Contributions.* Task automata cannot be used anymore if some of the timing constants are uncertain (for instance due to clock drift) or if they are unknown – which rules out the verification at early design stage. In this work, we extend task automata with timing parameters, i. e., unknown constants, as a unified formalism to model monoprocessor schedulability problems with several types of tasks (periodic, sporadic, or more complex). Most types of schedulers, including EDF (earliest-deadline first), FPS (fixed-priority) and SJF (shortest job first), with or without preemption, can be used. Most importantly, uncertain or unknown timing constants can be used thanks to timing parameters. Although

the general case is undecidable, we exhibit a large decidable subclass. We then propose a method that, given a parametric task automaton and a scheduling strategy, synthesizes parameter valuations for which the system is schedulable. For example, for what valuations of $p, p'$ is the PTaskA in Fig. 1 schedulable? We demonstrate the applicability of our formalism using the parametric real-time model-checker IMITATOR [AFKS12] augmented with an ad-hoc extension, and show that it can also address robust schedulability.

*Related work.* Schedulability analysis under uncertainty, i. e., with uncertain or unknown parameters, attracted recent attention. In [CPR08], parametric timed automata (PTAs) [AHV93] are used to perform parametric schedulability analysis: whereas the general case is unsurprisingly undecidable, the authors exhibit a subclass for which the schedulability-synthesis (i. e., synthesizing all valuations for which the system is schedulable) can be performed exactly.

In [BHJL13] parametric interrupt timed automata are proposed: this class inspired by PTAs is such that, at any time, at most one clock is active. This class allows a kind of preemption, and the reachability-emptiness problem is decidable.

In [SSL+13], we used parametric stopwatch automata (PSwA) to analyze a distributed real-time system with a preemptive fixed-priority strategy; while the analytical methods are faster, they are often incomplete, while the PSwA method implemented in a former version of IMITATOR turns out to be exact (sound and complete) on a set of case studies. This justifies the use of parametric model checking techniques instead of analytical techniques in order to analyze real-time systems under uncertainty. Finally, in [ALS15], IMITATOR was able to output the exact answer to an industrial challenge by Thales with uncertain periods, whereas other approaches were not able to compute this result (with the exception of one simulation-based approach, which did obtain the exact result without however the ability to assess its optimality).

Different from these previous works, are contribution aims at providing real-time system designers with a formalism natively including periods, deadlines and best- and worst-case computation times, and that also allows for uncertainty.

*Outline.* Section 2 recalls TaskA and introduces PTaskA. Section 3 studies the decidability of PTaskA. Section 4 introduces the modeling with PTaskA. Section 5 presents the practical translation into IMITATOR and Section 6 describes experiments. Section 7 concludes the paper.

## 2 Preliminaries: Task Automata

In this section, we mainly recall *task automata* from [FKPY07] (with some modifications in the syntax to better fit our framework), and introduce our parametric extension.

### 2.1 Clocks, parameters and constraints

Let $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}_+$ and $\mathbb{R}_+$ denote the sets of non-negative integers, integers, non-negative rational and non-negative real numbers respectively.

Throughout this paper, we assume a set $\mathcal{X} = \{x_1, \ldots, x_H\}$ of *clocks*, i. e., real-valued variables that evolve at the same rate. A clock valuation is a function $\mu : \mathcal{X} \to \mathbb{R}_+$. We write $\mathbf{0}$ for the clock valuation that assigns 0 to all clocks. Given $d \in \mathbb{R}_+$, $\mu + d$ denotes the valuation such that $(\mu + d)(x) = \mu(x) + d$, for all $x \in \mathcal{X}$. Given $R \subseteq \mathcal{X}$, we define the *reset* of a valuation $\mu$, denoted by $[\mu]_R$, as follows: $[\mu]_R(x) = 0$ if $x \in R$, and $[\mu]_R(x) = \mu(x)$ otherwise.

We assume a set $\mathcal{P} = \{p_1, \ldots, p_M\}$ of *parameters*, i. e., unknown rational-valued constants. A parameter *valuation* $v$ is a function $v : \mathcal{P} \to \mathbb{Q}_+$.

In the following, we assume $\lhd \in \{<, \leq\}$ and $\bowtie \in \{<, \leq, \geq, >\}$. A *$\mathcal{P}$-guard* $g$ is a constraint over $\mathcal{X} \cup \mathcal{P}$ defined by a conjunction of inequalities of the form $x \bowtie z$, where $z$ is either a parameter or a constant in $\mathbb{Q}_+$. A non-parametric guard is a $\mathcal{P}$-guard over $\mathcal{X}$ only, i. e., defined by a conjunction of inequalities of the form $x \bowtie d$.

We may assume bounds on the parameters; a parameter $p$ is *bounded* if its valuation domain is of the form $[a, \infty)$ or $[a, b]$ with $a, b \in \mathbb{N}$.

### 2.2 Tasks

Let $\mathcal{T} = \{t_1, t_2, \cdots\}$ be a set of tasks. Each task is characterized by three *timings*, i. e., constants in $\mathcal{P} \cup \mathbb{Q}_+$: *i*) $B$: its best-case execution time, *ii*) $W$: its worst-case execution time, and *iii*) $D$: its relative deadline (i. e., the latest time after the release of the task by which it must be completed). Given a task $t$ and a parameter valuation $v$, we denote by $v(t)$ the task where the parameters in the timings (i. e., $B$, $W$ and $D$) are replaced with their value in $v$.

Each task can have several *instances*, i. e., copies of the same task. An instance of task $t$ is written $(t, b, w, d)$ where $b \in \mathbb{R}_+$ (resp. $w \in \mathbb{R}_+$) is the best-case (resp. worst-case) remaining computation time, and $d \in \mathbb{R}_+$ the remaining time before the deadline.

### 2.3 Parametric task automata

Let us define parametric task automata as an extension of task automata defined in [FKPY07], where we allow the use of parameters in guards and invariants.[1]

**Definition 1 (PTaskA).** *A* parametric task automaton *(hereafter PTaskA) is a tuple $(\mathcal{T}, \Sigma, \mathcal{L}, l_0, \mathcal{X}, x_{done}, \mathcal{P}, I, T, E)$, where: i) $\mathcal{T}$ is a set of tasks, ii) $\Sigma$ is a set of actions, iii) $\mathcal{L}$ is a finite set of locations, iv) $l_0 \in \mathcal{L}$ is the initial location, v) $\mathcal{X}$ is a finite set of clocks, vi) $x_{done} \in \mathcal{X}$ is a special clock to be reset only when a task finishes, vii) $\mathcal{P}$ is a finite set of parameters, viii) $I$ is*

[1] As this definition is a contribution of this paper, it would better fit outside of the preliminaries section; however, it is convenient to define it first so as to then define task automata, and (parametric) timed automata in a straightforward manner.

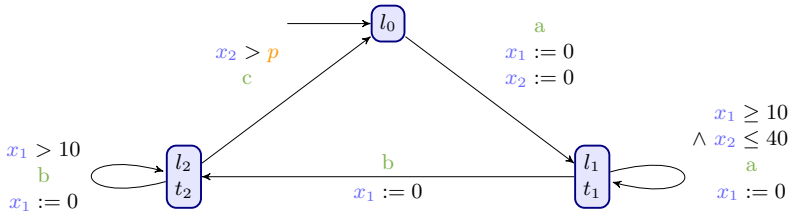



Fig. 2: An example of a PTaskA (inspired by [FKPY07, Fig.1])

*the invariant, assigning to every $l \in \mathcal{L}$ a $\mathcal{P}$-guard $I(l)$, ix) $T : \mathcal{L} \rightharpoonup \mathcal{T}$ is the partial task function, assigning to some locations a task, x) $E$ is a finite set of edges $e = (l, g, a, R, l')$ where $l, l' \in \mathcal{L}$ are the source and target locations, $a \in \Sigma$, $R \subseteq \mathcal{X}$ is a set of clocks to be reset, and $g$ is a $\mathcal{P}$-guard.*

$T$ is a partial function, and therefore some locations may be associated with no task. Also note that we define at most one task per location. Several tasks can be encoded in a straightforward manner by using several consecutive locations in 0-time. Also note that the parameters can be used both in the guards and invariants of the automaton, and/or in the task timings.

A PTaskA is said to have *no task feedback* if none of its guards and invariants contain $x_{done}$.

Given a PTaskA $A$ and a parameter valuation $v$, we denote by $v(A)$ the non-parametric *task automaton* (TaskA) where all occurrences of a parameter $p_i$ (in task timings, guards and invariants) have been replaced by $v(p_i)$. We will denote a TaskA using a tuple $(\mathcal{T}, \Sigma, \mathcal{L}, l_0, \mathcal{X}, x_{done}, I, T, E)$, with all elements defined as Definition 1 except that guards and invariants are non-parametric guards, and that all $B$, $W$, and $d$ in the tasks of $\mathcal{T}$ are non-parametric.

A TaskA is said to have *exact computation times* if $B = W$ for all tasks.

*Example 2.* Fig. 2 describes a PTask with 2 clocks, and 2 tasks: $t_1$, an instance of which is activated every time the PTaskA enters $l_1$, and $t_2$ (in $l_2$). For $t_1$, we have $B = 1$, $W = 2$ and $D = 10$; for $t_2$, $B = 2$, $W = p'$ and $D = 8$. Note that our formalism allows one to define parameters both in the automaton ($p$) and the task timings ($p'$). This PTaskA has no task feedback ($x_{done}$ is not used).

Basically, this PTaskA can create in $l_1$ between 1 and 5 instances of $t_1$ (but no more frequently than every 10 time units); then, it moves to $l_2$ where it can remain as long as wished, creating instances of $t_2$ (again no more frequently than every 10 time units). Eventually, the PTaskA can move back to the initial location no sooner than $p$ time units since the entering of $l_1$.

Intuitively, this PTaskA will be schedulable only if $p'$ (W of $t_2$) is not too large, and only when $p$ is not too small (otherwise one may loop too fast through the automaton for all tasks to terminate before their deadline).

### 2.4 (Parametric) timed and stopwatch automata

A parametric timed automaton (PTA) is a PTaskA for which $\mathcal{T} = \emptyset$. Similarly, a timed automaton (TA) is a TaskA for which $\mathcal{T} = \emptyset$.[2]

Lower-bound/upper-bound parametric timed automata (L/U-PTAs), proposed in [HRSV02], restrict the use of parameters in the model. A parameter is said to be an *upper-bound parameter* if, whenever it is compared with a clock, it is necessarily compared as an upper bound, i. e., it only appears in inequalities of the form $x \lhd p$. Conversely, a parameter is a *lower-bound parameter* if it is only compared with clocks as a lower bound, i. e., of the form $p \lhd x$. An L/U-PTA is a PTA where the set of parameters is partitioned into upper-bound parameters and lower-bound parameters.

Finally, TAs can be extended into stopwatch automata with the additional ability to stop some clocks in selected locations [CL00]. Similarly, PTAs can be extended into parametric stopwatch automata (PSwAs) [SSL+13]. We assume that (P)SwAs are equipped with *diagonal constraints*, i. e., guards made of a conjunction of inequalities of the form $x_i - x_j \bowtie \sum_{1 \leq i \leq M} \alpha_i p_i + d$, with $\alpha_i, d \in \mathbb{Q}$.

### 2.5 Task queue and scheduling strategy

A *task queue* is a sequence of instances of the form $\big((t_1, b_1, w_1, d_1), (t_2, b_2, w_2, d_2), \cdots\big)$. Given a non-parametric task set $\mathcal{T}$, let $\mathcal{Q}_\mathcal{T}$ denote all possible task queues. A *scheduling strategy* is a function $\mathsf{Sch} : \mathcal{T} \times \mathcal{Q}_\mathcal{T} \to \mathcal{Q}_\mathcal{T}$ that, given a task and a current task queue, inserts a new instance of this task into the task queue, while preserving the order of the other task instances in the queue. Famous scheduling strategies are EDF (earliest deadline first), FPS (fixed-priority scheduling) and SJF (shortest job first).

**Definition 2.** *A strategy is* non-preemptive *if it can never insert a new task instance as the first element of a non-empty queue. A strategy is* preemptive *if it can insert a new task instance as the first element of the queue, provided its task name is different from the name of every task in the queue, i. e., the current running task and all preempted tasks in the queue.*

*Example 3.* Let $q = \big((t_1, 1.4, 2.4, 3), (t_2, 2.5, 3.5, 4.2)\big)$. Assume a task $t_3$ where $B = 1, W = 1$ and $D = 10$. Then $\mathsf{EDF}(t_3, q) = \big((t_1, 1.4, 2.4, 3), (t_2, 2.5, 3.5, 4.2), (t_3, 1, 1, 10)\big)$, whereas preemptive $\mathsf{SJF}(t_3, q) = \big((t_3, 1, 1, 10), (t_1, 1.4, 2.4, 3), (t_2, 2.5, 3.5, 4.2)\big)$.

To be general enough, we only assume that schedulers must be encoded using a SwA. Also, the decision to insert a new task instance into the queue must be made only by comparing task timings of the new task instance with each of the existing instances (and possibly by looking at the discrete part of the queue, i. e., the ordering of the task names). This is not strong an assumption: for example, note that EDF, FPS and SJF (preemptive or not) all meet these criteria.

[2] In the literature, TAs are often defined using integer constants in guards and invariants; it is well-known that using rationals preserves decidability results, as rationals can be translated to integers using an appropriate constants rescaling.

### 2.6 Semantics of task automata

A configuration is a triple $(l, \mu, q)$ for location $l$, clock valuation $\mu$ and queue $q$.

**Definition 3 (Semantics of TaskA [FKPY07]).** *Given a scheduling strategy* $\mathsf{Sch}$*, the semantics of a TaskA* $A = (\mathcal{T}, \Sigma, \mathcal{L}, l_0, \mathcal{X}, x_{done}, I, T, E)$ *is a labeled transition system with initial state* $(l_0, \mathbf{0}, [])$ *and transitions defined as follows:*

- $(l, \mu, q) \xrightarrow{a}_{\mathsf{Sch}} (l', [\mu]_R, \mathsf{Sch}(T(l), q))$ *if* $(l, g, a, R, l') \in E$*,* $\mu \models g$ *and* $[\mu]_R \models I(l')$*, (*$\models$ *denotes satisfiability)*
- $(l, \mu, []) \xrightarrow{\delta}_{\mathsf{Sch}} (l', \mu + \delta, [])$ *if* $\delta \in \mathbb{R}_+$ *and* $(\mu + \delta) \models I(l)$*,*
- $(l, \mu, (t, b, w, d) :: q) \xrightarrow{\delta}_{\mathsf{Sch}} (l', \mu + \delta, \mathsf{Run}((t, b, w, d) :: q, \delta))$ *if* $\delta \in \mathbb{R}_+$*,* $\delta \leq w$ *and* $(\mu + \delta) \models I(l)$*, and*
- $(l, \mu, (t, b, w, d) :: q) \xrightarrow{fin}_{\mathsf{Sch}} (l, [\mu]_{\{x_{done}\}}, q)$ *if* $b \leq 0 \leq w$ *and* $[\mu]_{\{x_{done}\}} \models I(l)$*.*

*where* $[]$ *denotes the empty queue,* $::$ *is the list "cons" operator, and* $fin \notin \Sigma$ *is a fresh action name denoting task completion.*

The transition relation $\rightarrow$ is parameterized by the scheduler $\mathsf{Sch}$, as the strategy impacts the choice of the insertion into the queue of a new task instance. The first rule defines a discrete transition. The second rule defines time elapsing for the empty queue. The third rule defines time elapsing for a non-empty queue; $\mathsf{Run}(q, \delta)$ decreases by $\delta$ the value of all $d$ in $q$, as well as the $b$ and $w$ of its first element. The fourth rule defines the task completion, and resets $x_{done}$ as this clock is reset iff a task has completed.

### 2.7 Decidability of task automata

A TaskA is *schedulable* for a given strategy if, for all possible executions of the TaskA, all task instances meet their deadlines, i. e., they finish before their deadline, for any computation time within $[B, W]$.

Let us recall the main results from [FKPY07].

**Theorem 1 ([FKPY07, Theorems 1–4]).** *The problem of checking schedulability is decidable when relative to: i) a non-preemptive scheduling for TaskA; or ii) a preemptive scheduling strategy for TaskA without task feedback or with exact computation times.*

The decidability is obtained by encoding the scheduler $\mathsf{Sch}$ for this decidable class into a timed automaton, or a timed automaton with bounded subtraction, denoted by $A_{\mathsf{enc}}(\mathsf{Sch})$. Then, the synchronous product automaton $A \parallel A_{\mathsf{enc}}(\mathsf{Sch})$ is constructed. Finally, it is shown that the system is schedulable iff a special location (which corresponds to a deadline miss) is not reachable in $A \parallel A_{\mathsf{enc}}(\mathsf{Sch})$. The result follows from the decidability of the reachability in both timed automata [AD94] and timed automata with bounded subtraction [FKPY07].

**Theorem 2 ([FKPY07, Theorems 5]).** *The problem of checking schedulability is undecidable with (preemptive)* EDF, FPS, SJF*.*

## 3 Decidability and undecidability

In this section, we address the following 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?

### 3.1 Undecidability

The following undecidability results derive from two well-known results: *i*) the reachability-emptiness problem is undecidable for PTAs with at least three parametric clocks (clocks that are indeed compared to a parameter somewhere in the model) and a single parameter [Mil00] (although many variants of this result exist, see [And16] for a survey); and *ii*) general schedulability analysis is undecidable for TaskA [FKPY07].

**Theorem 3 (Undecidability).** *The schedulability-emptiness problem is undecidable for PTaskA with at least three parametric clocks and a single timing parameter, whatever the scheduling strategy.*

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

*Proof.* It is known that reachability emptiness is undecidable with at least three parametric clocks and one parameter [Mil00]. That is, we can encode a 2-counter machine using a PTA such that the machine halts iff a special location in the PTA is reachable. We reuse this construction by adding no task in the PTA, except to the special location, where we add two tasks with $B = W = D = 1$, activated in 0-time (adding two tasks requires a second additional location with an urgent transition). Now, if the special location is reachable, the system is necessarily non-schedulable (the first task will complete within 1 time unit, and the second one will immediately miss its deadline). Conversely, if the special location is unreachable, no task is ever activated and the system is necessarily schedulable. The result follows from the undecidability of the halting problem for 2-counter machines.

For the second part, it suffices to consider a PTaskA with a single parameter never used in the model, since non-parametric-schedulability analysis is already undecidable in general [FKPY07] (using possibly preemptive scheduling strategies). □

*Remark 1.* In the first part of Theorem 3, we require three parametric clocks in the model. Note that the scheduler translates itself into a PTA with several (possibly parametric) clocks; therefore, it is likely that the undecidability result holds for less clocks in the PTaskA. Exhibiting better bounds (which does not have huge practical applications though) is the subject of future work.

### 3.2 Decidability

In the non-parametric setting, the number of instances of a task $t$ (with timings $B$, $W$, $D$) is intuitively bounded by $\lceil D/W \rceil$; indeed, when the number of instances exceeds this bound, the queue will be overflown in the sense that it will be impossible to finish that many instances before the deadline $D$. Therefore, as soon as the queue exceeds this value, the system is non-schedulable and therefore, it is sufficient to consider a bounded queue for schedulability analysis. However, this reasoning does not hold for general PTaskA, as $W$ can be arbitrarily small, and $D$ arbitrarily large. This motivates the following definition.

**Definition 4.** *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 $a, b \geq 0$.*

That is, the $W$ cannot be 0, and the deadline cannot be infinite. Therefore, the maximum number of instances to be considered for a task is bounded by $\lceil \max(D)/\min(W) \rceil$, where max (resp. min) denotes the upper (resp. lower) bound of a parameter.

*Example 4.* The PTaskA in Fig. 2 trivially meets the schedulable-boundedness assumption, as necessarily $p' \geq B = 2 > 0$. In addition, the maximum number of instances necessary to check schedulability is $10/2 = 5$ for $t_1$ and $8/2 = 4$ for $t_2$.

We then slightly restrain the use of parameters in PTaskA in the following definition, following the similar restriction in L/U-PTAs.

**Definition 5.** *A PTaskA is an* L/U-PTaskA *if its parameters set is partitioned into lower-bound parameters and upper-bound parameters.*

**Theorem 4 (Decidability).** *The schedulability-emptiness problem is decidable for L/U-PTaskAs with schedulable-bounded parameters, for non-preemptive* FPS *and* SJF*, and non-preemptive* EDF *without parametric deadlines.*

*Proof.* Let us first show that Sch can be encoded into an L/U-PTA $A_{\mathsf{enc}}(\mathsf{Sch})$. Thanks to the restriction in Definition 4, we note that it is sufficient to consider a bounded number of instances for each task. Therefore, there is a bounded number of possible discrete queues. These combinations can be encoded using a finite number of locations in the L/U-PTA (more pragmatically, both in [NWY99,FKPY07] and in our implementation, we use shared global variables such as Booleans, integers, or lists, that act as syntactic sugar for extra locations). Whenever the queue exceeds its bounded size, we add a transition to a special error location.

Then, we follow the same encoding as in [FKPY07] for non-preemptive strategies: we create one clock per possible task instance (of which the number is bounded). Whenever $x > D \wedge x \leq W$, where $x$ denotes a task instance in a given location encoding a queue where this instance is indeed active, we add a transition to the error location, as this task instance missed its deadline. For FPS, this

encoding is such that $D$ and $B$ are always compared to clocks as lower-bounds, and are therefore lower-bound parameters, whereas $W$ is an upper-bound parameter. This gives that $A_{\mathsf{enc}}(\mathsf{Sch})$ is a (finite) L/U-PTA. For EDF, we need to compare expressions such as $D_i - x_i \bowtie D_j - x_j$; by forbidding parametric deadlines, the model remains again an L/U-PTA. Now, since $A$ is itself an L/U-PTA, the product $A \parallel A_{\mathsf{enc}}(\mathsf{Sch})$ is an L/U-PTA, where the error location is reachable for all valuations iff there exists no parameter valuation for which the system is schedulable. The result follows from the fact that the problem of knowing whether a location is reachable for all valuations is decidable for L/U-PTAs [And16].

For SJF, we have to compare the B and W with each other, but that can be done "statically" by considering all possible orderings, which gives a finite union of L/U-PTAs; we can show using a monotonicity property that the universality of each of these constrained L/U-PTAs is decidable. □

The class of PTaskA in Theorem 4 is large. Indeed, the assumption of schedulable-bounded parameters is more than reasonable: both an infinite deadline and a 0-time WCET seem doubtful cases. In addition, the L/U assumption is not much restrictive either: first note that any PTaskA with no parameter in the automaton (but with parametric timings in the tasks, except for deadlines for EDF) fits into this class. Second, this assumption mainly consists in disallowing equality with parameters in the PTaskA, which does not seem much a restriction. Both Fig. 1 (except for EDF, unless $p'$ is valuated) and Fig. 2 (for all strategies) fit into this class.

*Remark 2.* The decidability of the schedulability-emptiness problem does not necessarily mean that one is able to *synthesize* all parameter valuations. In fact, it was shown in [JLR15] that the synthesis is in general intractable for L/U-PTAs: more precisely, it is (in general) impossible to represent the set of valuations for which a given location is reachable in an L/U-PTA using a finite union of polyhedra. However, we can mitigate this in two ways. First, the non-emptiness is constructive: that is, if the set of valuations for which the system is schedulable is not empty, then one is certain to exhibit immediately a set of valuations (maybe incomplete though) using procedures from [HRSV02]. When synthesizing all valuations is out of reach, exhibiting at least some is also of interest. Second, we have in fact a more pragmatical goal, as the subject of the next section will be to synthesize valuations not only for this decidable subclass, but for the general class of PTaskA – maybe not all such valuations (due to Theorem 3) but as many as possible.

## 4 Schedulability analysis for parametric task automata

In this section, we adopt a more pragmatical view. Since we only constrain a scheduler to be encoded using a stopwatch automaton, we therefore directly translate any scheduler (preemptive or not) into a (parametric) stopwatch automaton. Even in the decidable cases (where we showed that stopwatches are not needed), we potentially use stopwatches.

As noted in [FKPY07], most scheduling strategies can fit into timed (or stopwatch) automata, and therefore fit into parametric stopwatch automata when extended with parameters.

However, the discrete part of the queue might require an unbounded number of locations. Whereas in the non-parametric case, a sufficient bound can be computed which is sufficient for schedulability, this does not hold anymore in the parametric case. Therefore, in the remainder of the paper, we always assume the mild assumption of schedulable-boundedness of Definition 4, and therefore we can infer a bound on the length of the tasks queue.

We do not go into full details for encoding strategies, as this was (partially) done in [NWY99,FKPY07], and would require lengthy details; we however give the general idea below.

*General idea.* We will consider the synchronous product of two PSwAs in parallel: the actual PTaskA $A$, and the translation of the scheduler $\mathsf{Sch}$ into a second PSwA $A_{\mathsf{enc}}(\mathsf{Sch})$. As noted earlier, a PTaskA is just a PTA, where some locations activate task instances. Therefore, the PTaskA can be transformed into an almost-identical PSwA (without stopwatches), by labeling each edge going into a location where task $t$ is activated by a fresh action $Act_t$. Then, the scheduler will synchronize on actions $Act_t$, and manage the tasks queue according to its strategy.

The locations of $A_{\mathsf{enc}}(\mathsf{Sch})$ are all possible configurations of the discrete part of the tasks queue, of which there is a finite number thanks to the schedulable-boundedness assumption. At any time, if the size of the queue overflows the maximal queue size implied by the schedulable-boundedness assumption, $A_{\mathsf{enc}}(\mathsf{Sch})$ will go to a special error location, which denotes that the system is non-schedulable.

We use the following clocks for $A_{\mathsf{enc}}(\mathsf{Sch})$. First, for each task $t_i$, we use a unique clock (say $x_i$), that serves to measure the unique running instance; note that at most one task instance of task $t_i$ has a non-zero time of already executed computation, from the definition of non-preemptive and preemptive strategies (from Definition 2). These task clocks may be stopped (which is why we require stopwatches): in fact, they will always be stopped, unless an instance of the current task is currently being executed. These task clocks are initially 0, run when an instance of the task is executed, and are reset when such an instance is completed.

Second, for each task instance, we create one clock. For example, clock $x_i^j$ denotes the clock for the instance $j$ of task $t_i$. Thanks to the schedulable-boundedness assumption, we know the maximum required number of instances per task. These task instance clocks are never stopped; whenever an instance $j$ of task $t_i$ is active, if this instance misses its deadline $D_i$ (which can be tested using a guard $x_i^j > D_i$), $A_{\mathsf{enc}}(\mathsf{Sch})$ is sent to a special error location.

We now briefly review the specificities of the three scheduling strategies.

$\mathsf{EDF}$ *scheduler.* In order to encode $\mathsf{EDF}$, one must identify the task instance with an earliest deadline: for example, if $D_i - x_i^j < D_{i'} - x_{i'}^{j'}$, then instance $j$ of task $t_i$

has an earlier deadline than instance $j'$ of task $t_{i'}$ and should be executed first. This can be tested thanks to the diagonal constraints in PSwAs.

FPS *scheduler.* The fixed-priority scheduling strategy is encoded directly on the discrete part of $A_{\mathsf{enc}}(\mathsf{Sch})$. When a new instance of task $t_i$ is activated (action $Act_t$), if that task has a higher priority than the task currently executed (say $t_{i'}$), then the scheduler temporarily stops $t_{i'}$ and starts executing $t_i$; otherwise, the scheduler keeps executing $t_{i'}$ and inserts a new instance of $t_i$ into the queue.

SJF *scheduler.* In order to encode SJF, one must identify the task with the shortest job: for example, if $W_i - x_i < W_{i'} - x_{i'}$, then the running instance of task $t_i$ has a shorter job than the running of task $t_{i'}$ and should be executed first.

Using the above construction $A_{\mathsf{enc}}$, we have:

**Proposition 1.** *Given a PTaskA $A$ and strategy* Sch*, the system is schedulable for all valuations for which the error location is unreachable in $A \parallel A_{\mathsf{enc}}(\mathsf{Sch})$.*

# 5 Parameter synthesis for PTaskA using IMITATOR

## 5.1 IMITATOR

IMITATOR [AFKS12] is a parametric model checker for networks of PSwAs extended with various features, including global rational-valued variables, strong broadcast synchronization, and linear clock assignments (instead of being reset to 0, a clock $x$ can also be set, e. g., to $x' + p$). The symbolic computations are performed using polyhedra [BHZ08]. IMITATOR implements various algorithms; the one used here is EFsynth ("reachability synthesis" [AHV93,JLR15]). EFsynth is in fact a semi-algorithm: it is not guaranteed to terminate but, if it does, then its result is exact.

## 5.2 Translation into parametric stopwatch automata

In our translation of the scheduler into a PSwA, we extensively use stopwatches, even in the decidable cases. The reason is that, while the semantics of SwA cannot be encoded using Difference Bound Matrices (DBMs, a popular data structure rendering TAs very efficient) in the non-parametric setting, however they come for free in the parametric setting (as stopwatches can be encoded into polyhedra, which usually encode the semantics of PTAs).

In order to reduce the state space, we also implemented several optimizations using the expressive power of IMITATOR: *i*) The queue is not implemented into locations, but using a set of variables. In contrast to [NWY99] where Booleans are used to denote whether a task instance is active or not, we use a single integer for each task $t_i$, that encodes the number of active instances for $t_i$. *ii*) We also use stopwatches as much as possible: whenever a instance clock denotes an inactive instance, it is set to 0 and stopped so as to not create unnecessary diverging relations with the other clocks.

## 6 Experiments

As writing such a scheduler quickly becomes tedious and error-prone, we implemented an external program (650 lines of Python) that takes as input on the one hand a scheduling strategy Sch and on the other hand the list of tasks of the PTaskA $A$ (with their timings, their priority (for FPS), their maximum number of instances...), and automatically generates the corresponding PSwA $A_{\mathsf{enc}}(\mathsf{Sch})$ in the IMITATOR input format. Then, it suffices to pass to IMITATOR the model made of $A$ and $A_{\mathsf{enc}}(\mathsf{Sch})$.

We used IMITATOR 2.9.1 for our experiments. All the subsequent analyses terminate in (at most) a few seconds on a MacBook Pro i7 2.67GHz.[3]

In this section, we consider a preemptive FPS scheduler. All the results are exact – although the preemptive FPS scheduler is clearly beyond the decidable class of Theorem 4.

*Non-parametric analysis.* Quite trivially, our framework allows for non-parametric analysis. Setting $p = 10$ and $p' = 4$, IMITATOR concludes that the PTaskA in Fig. 1 is schedulable for preemptive FPS. With priorities $t_0 > t_1 > t_2 > t_3$, the system becomes non-schedulable.

*Mixing parameters.* Let us go back to Fig. 2. First, we set $p = 100$, and we obtain that the system is schedulable for $p' \in [2, 3]$. Second, we set $p' = 3$, and we obtain that the system is schedulable for $p \geq 42$. This confirms both intuitions that $p'$ should be not too large, and $p$ large enough for the system to be schedulable. Finally, we run an analysis with both parameter dimensions, which gives:

$$p' \in [2,3] \wedge p \geq 42 \quad \vee \quad p' = 2 \wedge p \in [8,42) \quad \vee \quad p' > 2 \wedge p < 42 \wedge p \geq 36 + 2 \times p'$$

A graphical representation output by IMITATOR is given in Fig. 3a (where `p100` stands for $p$ and `Q_WCET` for $p'$).

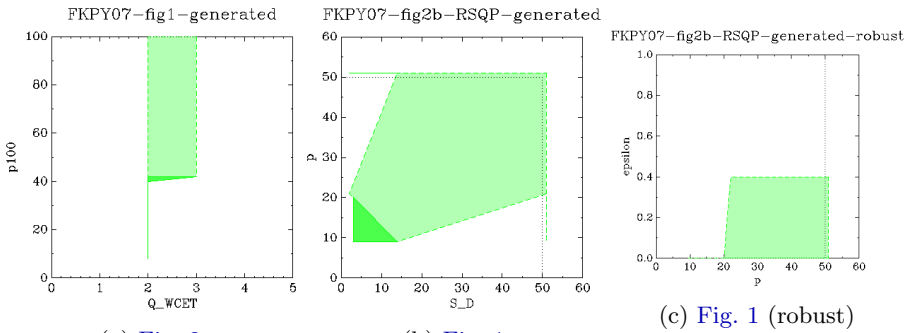
Concerning Fig. 1, setting $p' = 4$ yields $p \geq 9$, while a parametric schedulability analysis on both dimensions gives

$$p \geq 9 \wedge p' \geq 2 \wedge p + p' \geq 23 \quad \vee \quad p \geq 9 \wedge p' \geq 3 \wedge p + p' < 23$$

A graphical representation output by IMITATOR is given in Fig. 3b (where `S_D` stands for $p'$).

*Robustness analysis.* Finally, we can perform robustness analysis: often, TAs and their extension are not by default robust, i. e., they can require infinitely precise behaviors. It may happen that a property holds only if all timing constants are implemented exactly as they were specified. In contrast, robustness analysis (see, e. g., [BMS13]) consists in checking whether there exists some $\epsilon > 0$ for which, when all guards may be enlarged by $\epsilon$, the system still meets its property. To check whether Fig. 1 is robustly schedulable, we modify the system as

[3] Sources, binaries, models and results are available at https://www.imitator.fr/static/FMICS17.




(a) Fig. 2 (b) Fig. 1 (c) Fig. 1 (robust)

Fig. 3: Visualization of parametric schedulability zones

follows: any constraint (in both $A$ and $A_{\mathsf{enc}}(\mathsf{Sch})$) of the form $x \leq z$, $x \geq z$, or $x = z$, where $z \in \mathbb{Q}_+ \cup \mathcal{P}$, is transformed into $x \leq z + \epsilon$, $x \geq z - \epsilon$, or $z - \epsilon \leq x \leq z + \epsilon$, respectively (and similarly for strict constraints). Applying this modification to Fig. 1 with $p = 10$ and $p' = 4$, by adding a fresh parameter $\epsilon$, gives the constraint $\epsilon = 0$: that is, this system is *not* robustly schedulable. Any modification (even infinitesimal) of the timing constants may render the system non-schedulable.

We can combine parametric schedulability with robustness analysis: keeping both $p$ and $\epsilon$ gives

$$p \geq 9 \wedge \epsilon = 0 \quad \vee \quad \epsilon \leq \frac{2}{5} \wedge p \geq 20 + 5\epsilon \wedge p \geq 19 + 8\epsilon$$

That is, the system is not schedulable for $p < 9$, is schedulable but not robustly for $p \in [9, 20]$ and becomes robust from 20. Note that our constraint even gives by how much the guards can be enlarged (the value depends on $p$ and never exceeds $\frac{2}{5}$). A graphical representation output by IMITATOR is given in Fig. 3c.

## 7 Conclusion

We introduced a unified and concise model for parametric schedulability analysis for (non-)preemptive strategies on a monoprocessor. While the general case is undecidable, we exhibited a decidable subclass, and our implementation terminates with an exact result on benchmarks even outside of the decidable class.

While formal methods with timing parameters might not scale to verify the schedulability of very large systems with all details, we believe they can provide designers with first schedulability results on subparts of the system, or to derive timing bounds on abstractions of it. Designing ad-hoc abstractions for our framework is on our agenda.

There is still some open space between our decidability result (Theorem 4) and our undecidability results (Theorem 3). A promising way to improve the

knowledge of decidability would be to show that L/U-parametric timed automata with bounded subtractions are decidable, which would allow to extend our decidable subclass of PTaskA. Conversely, a good candidate for undecidability is non-preemptive strategies without the schedulable-boundedness assumption.

So far, whereas the scheduler is automatically generated, the PTaskA still needs to be manually constructed. A natural future work is therefore to propose on the one hand a library of patterns (periodic tasks, sporadic tasks...), and on the other hand an automated translation from existing formalisms.

Of course, handling multiprocessor scheduling is on our agenda, as well as mixed-criticality scheduling. Finally, we would also like to consider a parameterization of a recent extension of TaskA [FLSC16].

## References


AD94. Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

AFKS12. Étienne André, Laurent Fribourg, Ulrich Kühne, and Romain Soulat. IMITATOR 2.5: A tool for analyzing robustness in scheduling problems. In *FM*, volume 7436 of *Lecture Notes in Computer Science*, pages 33–36. Springer, 2012.

AHV93. Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. Parametric real-time reasoning. In *STOC*, pages 592–601. ACM, 1993.

ALS15. Étienne André, Giuseppe Lipari, and Youcheng Sun. Verification of two real-time systems using parametric timed automata. In *WATERS*, 2015.

AM02. Yasmina Adbeddaïm and Oded Maler. Preemptive job-shop scheduling using stopwatch automata. In *TACAS*, volume 2280 of *Lecture Notes in Computer Science*, pages 113–126. Springer-Verlag, 2002.

And16. Étienne André. What's decidable about parametric timed automata? In *FTSCS*, volume 596 of *Communications in Computer and Information Science*, pages 52–68. Springer, 2016.

BHJL13. Béatrice Bérard, Serge Haddad, Aleksandra Jovanovic, and Didier Lime. Parametric interrupt timed automata. In *RP*, volume 8169 of *Lecture Notes in Computer Science*, pages 59–69. Springer, 2013.

BHZ08. Roberto Bagnara, Patricia M. Hill, and Enea Zaffanella. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Science of Computer Programming*, 72(1–2):3–21, 2008.

BMS13. Patricia Bouyer, Nicolas Markey, and Ocan Sankur. Robustness in timed automata. In *RP*, volume 8169 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2013.

CL00. Franck Cassez and Kim Guldstrand Larsen. The impressive power of stopwatches. In *CONCUR*, volume 1877 of *Lecture Notes in Computer Science*, pages 138–152. Springer, 2000.

CPR08. Alessandro Cimatti, Luigi Palopoli, and Yusi Ramadian. Symbolic computation of schedulability regions using parametric timed automata. In *RTSS*, pages 80–89. IEEE Computer Society, 2008.

FKPY07. Elena Fersman, Pavel Krcál, Paul Pettersson, and Wang Yi. Task automata: Schedulability, decidability and undecidability. *Information and Computation*, 205(8):1149–1172, 2007.

FLSC16. Bingbing Fang, Guoqiang Li, Daniel Sun, and Hongming Cai. Schedulability analysis of timed regular tasks by under-approximation on WCET. In *SETTA*, volume 9984 of *Lecture Notes in Computer Science*, pages 147–162, 2016.

HRSV02. Thomas Hune, Judi Romijn, Mariëlle Stoelinga, and Frits W. Vaandrager. Linear parametric model checking of timed automata. *Journal of Logic and Algebraic Programming*, 52-53:183–220, 2002.

JLR15. Aleksandra Jovanović, Didier Lime, and Olivier H. Roux. Integer parameter synthesis for timed automata. *IEEE Transactions on Software Engineering*, 41(5):445–461, 2015.

Mil00. Joseph S. Miller. Decidability and complexity results for timed automata and semi-linear hybrid automata. In *HSCC*, volume 1790 of *Lecture Notes in Computer Science*, pages 296–309. Springer, 2000.

NWY99. Christer Norström, Anders Wall, and Wang Yi. Timed automata as task models for event-driven systems. In *RTCSA*, pages 182–189. IEEE Computer Society, 1999.

SSL+13. Youcheng Sun, Romain Soulat, Giuseppe Lipari, Étienne André, and Laurent Fribourg. Parametric schedulability analysis of fixed priority real-time distributed systems. In *FTSCS*, volume 419 of *Communications in Computer and Information Science*, pages 212–228. Springer, 2013.