

Parametric schedulability analysis of a launcher flight control system under reactivity constraints

Étienne André^{1,2,3} , Emmanuel Coquard⁴, Laurent Fribourg⁵,
Jawher Jerray⁶  and David Lesens⁴

¹Université de Lorraine, CNRS, Inria, LORIA, Nancy, France

²JFLI, CNRS, Tokyo, Japan

³National Institute of Informatics, Tokyo, Japan

⁴ArianeGroup SAS, Les Mureaux, France

⁵Université Paris-Saclay, LSV, CNRS, ENS Paris-Saclay, France

⁶Université Sorbonne Paris-Nord, LIPN, CNRS, UMR 7030, F-93430, Villetaneuse, France

Abstract

The next generation of space systems will have to achieve more and more complex missions. In order to master the development cost and duration of such systems, an alternative to a manual design is to automatically synthesize the main parameters of the system. In this paper, we present an approach for the specific case of the scheduling of the flight control of a space launcher. The approach requires two successive steps: (1) the formalization of the problem to be solved in a parametric formal model and (2) the synthesis of the model parameters with a tool. We first describe the problem of the scheduling of a launcher flight control, then we show how this problem can be formalized with parametric stopwatch automata; we then present the results computed by the parametric timed model checker IMITATOR. We enhance our model by taking into consideration the time for switching context, and we compare the results to those obtained by other tools classically used in scheduling.

Keywords— scheduling, real-time systems, model checking, parameter synthesis, IMITATOR

1 Introduction

Real-time systems combine concurrent behaviors with hard timing constraints. An out-of-date reply is often considered as invalid even if its content is correct. For *critical* real-time systems, if a time constraint is violated, then the consequences may

This manuscript is the author version of the manuscript of the same name published in *Fundamenta Informatica* 182(1). This is an extended version of the manuscript published in the proceedings of the 19th International Conference on Application of Concurrency to System Design (ACSD 2019). The final authenticated version is available at <https://doi.org/10.3233/FI-2021-2065>. This work is partially supported by the ANR national research program PACS (ANR-14-CE28-0002) and ERATO HASUO Metamathematics for Systems Design Project (No. JPMJER1603), JST.

be disastrous. Thus, a formal verification phase is essential in order to statically guarantee that all the tasks will be executed in their allocated time, and that the system will return results within the times guaranteed by the specification.

Assessing the absence of timing constraints violations is even more important when the system can be hardly controlled once launched. This is especially true in the aerospace area, where a system can only very hardly be modified or even rebooted after launching.

The next generation of space systems will have to achieve more and more complex missions. In order to master the development cost and duration of such systems, an alternative to a manual design is to automatically synthesize the main parameters of the system. While verifying a real-time system is already a notoriously difficult task, we tackle here the harder problem of *synthesis*, i. e., to automatically synthesize a part of the system so that it meets its specification. We notably focus on the synthesis of admissible *timing* values.

1.1 Contribution

In this paper, we address the specific case of the scheduling of the flight control of a space launcher. Our approach requires two successive steps:

1. the formalization of the problem to be solved in a parametric formal model and,
2. the synthesis of the model parameters with a tool.

We first describe the problem of the scheduling of a launcher flight control; then we formalize this problem with parametric stopwatch automata, an extension of timed automata [AD94] with parameters [AHV93] and stopwatches (“the ability to stop clocks”) [CL00; Sun+13]; third, we present the results computed by the IMITATOR [And21] tool. We compare our results with those obtained by other tools classically used in scheduling. A key aspect is the verification and synthesis under some *reactivity constraints*: the time from a data generation by sensor’s measurement which is considered as an input to its output must always be less than a threshold. The solution we propose is compositional, in the sense that the reactivity constraints can be checked independently. We consider both an instantaneous switch from one thread to another, and more general systems where the switch between two threads has a CPU cost due to the copy of data between the contexts of each thread.

We propose here a solution to the problems using an extension of parametric timed automata (PTA), which are an extension of finite state automata with clocks and parameters [AHV93]. The general class of PTA is notoriously undecidable, and notably the mere problem of deciding whether at least one parameter valuation allows the system to reach a global state is undecidable, even over discrete time [AHV93], with a single integer-valued parameter over dense time [Ben+15], or with a single clock compared to parameters [Mil00] (see [And19] for a survey). Still, some decidable subclasses were proposed (e. g., [Hun+02; BL09; AL17; ALR18]), notably in the field of scheduling real-time systems [CPR08; And17]. In spite of these undecidability results, the use of parametric timed automata for solving various concrete problems was recently considered, in frameworks such as hardware verification [Che+09], analysis of music scores [FJ13], monitoring [AHW18] or software product lines testing [Lut+19]. We show here that this formalism is also useful for solving concrete scheduling problems—such as the one considered here.

1.2 Outline

After discussing related works in [Section 2](#), [Section 3](#) presents the problem we aim at solving. [Section 4](#) recalls parametric stopwatch automata. [Section 5](#) exposes our modeling; we extend our solution in [Section 6](#) in a compositional manner, and in [Section 7](#) to enhance the model with context switch times. [Section 8](#) gives the results obtained, while [Section 9](#) makes a comparison with other tools of the literature (solving only a part of the problem). [Section 10](#) concludes the paper.

2 Related works

2.1 Scheduling

A long line of works in the last five decades has been devoted to the problem of scheduling analysis of real-time systems with various flavors. Several analytical methods were proposed to study the schedulability for a particular situation. Such analytical methods need to be tuned for each precise setting (uniprocessor or multiprocessor, scheduling policy, absence or presence of offsets, jitters, etc.). Most of them do not cope well with uncertainty. For example, in [\[BB97\]](#), three methods for the schedulability analysis with offsets are proposed. In [\[BB04\]](#), an efficient approach for testing schedulability for RMS (rate monotonic) in the case of (uniprocessor) analysis is proposed, through a “parameter” (different from our timing parameters) to balance complexity versus acceptance ratio.

2.2 Scheduling with model checking

Schedulability with model checking is a trend that started as early as the first works on timed model checking (e. g., [\[WME92; AHV93; AD94; YMW97; CC99\]](#)), and grew larger since the early 2000s.

A natural model to perform schedulability analysis is (extensions of) timed automata (TA) [\[AD94\]](#). On the negative side, the cost of state space explosion often prevents the verification of very large real-time systems. On the positive side, they allow for more freedom, and can model almost any system with arbitrarily complex constraints; in addition, despite the cost of state space explosion, they can be used to verify small to medium-size systems for which no other method is known to apply.

In [\[AM01; AM02\]](#), (acyclic) TA are used to solve job-shop problems. The preemption is encoded in [\[AM02\]](#) with *stopwatches*, while keeping some decidability results. In [\[AAM06\]](#), scheduling is performed using TA. Timed automata allow to model naturally and verify more complex systems, which are not captured so easily in traditional formalisms for schedulability analysis.

In [\[NWX99; Fer+07\]](#), task automata are proposed as a formalism extending TA to ease the modeling (and the verification) of uniprocessor real-time systems: in some cases, the schedulability problem of real-time systems is transformed into a reachability problem for standard TA and it is thus decidable. This allows applying model-checking tools for TA to schedulability analysis with several types of tasks and most types of scheduler.

In [\[Sun+14\]](#), hierarchical scheduling systems are encoded using linear hybrid automata, a model that generalizes TA. This approach outperforms analytical methods in terms of resource utilization. In [\[SL14\]](#), linear hybrid automata are used to perform schedulability analysis for multiprocessor systems under a global fixed priority

scheduler: this method is more scalable than existing exact methods, and shows that analytical methods are too pessimistic.

In [Fan+16], a schedulability analysis method is introduced using the model of *timed regular task automata* (using under-approximated WCETs) and then using nested timed automata; this method is shown to be exact.

The problem we solve here shares similarities with analyses done in [For+10; Mik+10]. An important difference between [For+10; Mik+10] and our case study comes from the fact that, here, there are two distinct notions of “thread” and “processing”, while in [For+10; Mik+10] there was only one notion called “task”. Most importantly, none of these works consider timing parameters.

2.3 Scheduling with parameters

When some of the design parameters are unknown or imprecise, the analysis becomes much harder. Model checking with parameters can help to address this. In [CPR08], PTA are used to encode real-time systems so as to perform parametric schedulability analysis. A subclass (with bounded offsets, parametric WCETs but constants deadlines and periods) is exhibited that gives exact results. In contrast, our work allows for parameterized deadlines; in addition, reactivities are not considered in [CPR08].

In [Fri+12], we performed robust schedulability analysis on an industrial case study, using the inverse method for PTA [And+09; AS13] implemented in IMITATOR. While the goal is in essence similar to the one in this manuscript, the system differs: [Fri+12] considers multiprocessor, and preemption can only be done at fixed instants, which therefore resembles more Round Robin than real FPS. In [Sun+13], we showed that PTA-based methods are significantly more complete than existing analytical methods to handle uncertainty. In [SAL15], we solved an industrial challenge by Thales using IMITATOR; in [And+19], we verified an industrial asynchronous leader election algorithm using IMITATOR, with additional abstractions.

In [Le+13], the analysis is not strictly parametric, but concrete values are iterated so as to perform a cartography of the schedulability regions. However, the resulting analysis of the system is incomplete.

In [Bér+16], timed automata are extended with multi-level clocks, of which exactly one at a time is active. The model enjoys decidability results, even when extended with polynomials and parameters, but it remains unclear whether concrete classes of real-time systems can actually be modeled.

The aforementioned task automata were extended in [And17] to *parametric* task automata; some schedulability problems remain decidable in this setting, i.e., it is possible in some cases to decide whether the set of valuations ensuring schedulability is empty or not. In addition, procedures for exhibiting schedulability regions are proposed and implemented.

Finally, ROMÉO [Lim+09] also allows for parametric schedulability analysis using parametric time Petri nets [TLR09], with applications to critical real-time systems [Par+16].

3 Description of the system and problem

The flight control of a space launcher is classically composed of three algorithms:

1. The *navigation* computes the current position of the launcher from the sensor’s measurement (such as inertial sensors);

1	processing	Navigation	(Meas	: in)	is period	(5ms);	end;
2	processing	Guidance			is period	(60ms);	end;
3	processing	Control	(Cmd	: out)	is period	(10ms);	end;
4	processing	Monitoring	(Safeguard:	out)	is period	(20ms);	end;

Figure 1: An example of a flight control system

2. The *guidance* computes the most optimized trajectory from the launch pad to the payload release location;
3. The *control* orientates the thruster to follow the computed trajectory.

Due to the natural instability of a space launcher, strict real-time requirements have to be respected by the implementation of the flight control: frequency of each algorithm and reactivity between the sensor’s measurement acquisition and the thruster’s command’s sending.

The case study described in this paper is a simplified version of a flight control composed of a navigation, a guidance, a control and a monitoring algorithms; these four parts are called *processings*¹ in the following. Each processing has a name and a required rational-valued period; in our setting, the processing deadline is equal to the period. A processing can potentially read data from the avionics bus (“in” data) and/or write data to the same avionics bus (“out” data). Fig. 1 shows an example of such a system (all the numerical data provided in this paper are only examples that do not necessarily correspond to an actual system).

3.1 Threads and deterministic communications

The software components of the system are physically deployed on a single processor [OGL06]. Processings are allocated on *threads* run by the processor.

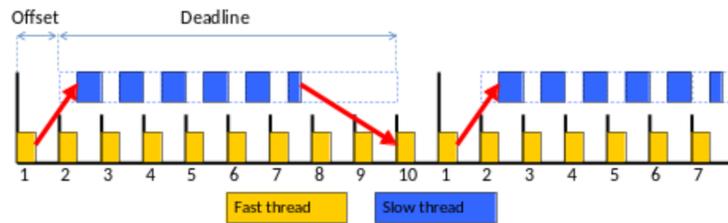


Figure 2: The communication between threads

Fig. 2 exemplifies the way data are exchanged between two threads. The fast thread (in yellow) has a period of 1 time unit. This period defines the time granularity of the system (this implies that the offset of the fast thread is 0 and that its deadline is 1). In this example, the slow thread (in blue) has an offset of 1 (its start is delayed by 1 cycle compared to the start of the fast thread), a period of 10 and a deadline of 8. The numbers from 1 to 10 denote the index of the period of the fast thread within

¹Following the vocabulary used within ArianeGroup. Technically, a processing is a *node* in SCADE, i. e., a subprogram activated cyclically, with a frequency, an activation condition and inputs/outputs.

the period of the slow thread. The first communication between the fast thread and the slow one is performed at the end of the first *period*; this explains that, although the second occurrence of the fast thread finishes before the first occurrence of the slow thread, this is the *first* occurrence of the fast thread which is communicated to the slow thread. Similarly, in order to ensure the determinism and taking into the priority between the threads, the communication between the slow thread and the fast thread is performed at the *deadline* of the slow thread, i.e., at the end of cycle 9 (offset + deadline). That is, the first occurrence of the fast thread to receive data from the slow thread is not the one starting at $t = 8$ nor at $t = 9$, but the one starting at $t = 10$.

In our setting, all the thread periods are *harmonic*, i.e., a thread period is a multiple of the period of the thread just faster (they pairwise divide each other). In other terms, for a system that contains a set of threads $\mathbf{t}_1, \dots, \mathbf{t}_k$, all the thread periods are considered harmonic if for every thread \mathbf{t}_j (for all $j \in \{1, \dots, k\}$), the period PT_j of \mathbf{t}_j is a multiple of the periods of all the threads of smaller period, that is, PT_j is a multiple of the periods of all threads $\mathbf{t}_i \in \{\mathbf{t}_1, \dots, \mathbf{t}_k \mid PT_i < PT_j\}$. In our case, the harmonic assumption on threads will not affect the modeling of the system in Section 5; however, it may be used to reduce the number of clocks and considerably decrease the computation time of our approach.

In addition, in order to ensure the determinism of the scheduling (which facilitates the verification of the system), the threads work in a synchronous manner:

- The inputs of a thread are read *at its start*; that is, no inputs are read during the execution of the thread.
- The outputs of a thread are provided *at its deadline*; that is, not only no outputs are provided during the execution of the thread, but the output is also not provided as soon as the thread terminates—if it terminates before its deadline—but only at its deadline.

Switch time In our case study, the *switch* time, i.e., the time needed by the CPU to copy memory information when changing threads, is $500 \mu s$.

3.2 Reactivities

To ensure the controllability of the launcher, a *reactivity*² is required between a data read from the avionics bus (a measurement) and a data written to the avionics bus (a command). A reactivity imposes a maximum bound on the time required by these data to “travel” from the measurement to the command. This concept is quite similar to that studied in [For+10] (without timing parameters).

Definition 3.1 (reactivity) *A reactivity constraint imposes an upper bound from a data read from the avionics bus to a data written to the avionics bus, where the sequence of the path of the data represents a precedence constraint.*

Several paths are potentially possible between a read data and a written data. Fig. 3 shows an example of such reactivities.

Reactivities too must follow the deterministic communication model from Section 3.1. Consider the execution of threads and processings in Fig. 4 (the values of periods and WCETs are given for illustration purpose, and do not correspond to the ones from our case study). Consider the reactivity imposing that the sequence of data

²In the literature, the term “reactivity” is also referred to as “latency” (see, e.g., [For+10]).

1	reactivity Meas → Navigation → Guidance → Control → Cmd is 150ms;
2	reactivity Meas → Navigation → Control → Cmd is 15ms;
3	reactivity Meas → Navigation → Monitoring → Safeguard is 55ms;

Figure 3: Some typical reactivities

“Meas → Navigation → Guidance → Control → Cmd” should be equal to or less than 5. Due to the data being communicated at the end of each thread only, the Guidance processing (marked with green “G” in Fig. 4) does not receive the data from the third execution of the Navigation processing (marked with “N” in red), as the data of the third Navigation will be sent at the end of the thread T1 period, but from the second execution of Navigation. Therefore, in Fig. 4, the only path of interest is the path of the data starting from the second execution of Meas, going to the second execution of Navigation, then going to the (only) execution of Guidance, and then finishing in the third execution of Control, before being written to the third occurrence of Cmd. Also note that the data output by the first execution of Navigation are successfully sent to T2 at the end of the first period of T1, but will be overwritten by the second occurrence of Navigation, and are therefore not of interest when checking the satisfaction of reactivities. Therefore, the time from the production of these data (at $t = 1$) to their writing on the avionic bus (at $t = 6$) is 5, and therefore the reactivity is satisfied.

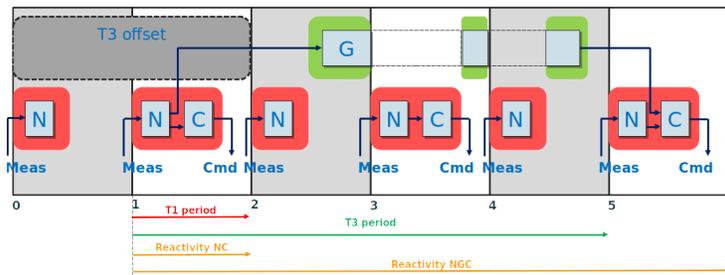


Figure 4: Determinism and reactivities

We want to solve the scheduling problem of periodic processings *under reactivity constraints*.

3.3 Processings and assignment into threads

A WCET (worst case execution time) is measured or computed for each processing. An example is given in Fig. 5.

An important problem is to find a proper assignment of the processings into threads, with their respective periods, while minimizing the number of threads. A solution to this problem consists of a set of cyclic threads on which the processings

1	processing wcet	Navigation	(1ms);
2	processing wcet	Guidance	(15ms);
3	processing wcet	Control	(3ms);
4	processing wcet	Monitoring	(5ms);

Figure 5: Example of Worst Case Execution Times

are deployed. In our setting, these threads are scheduled with a preemptive and fixed priority policy (FPS). A thread has a name and is defined by the following data:

1. a rational-valued period;
2. a rational-valued offset (with $0 \leq \text{offset} < \text{period}$), i. e., the time from the system start until the first periodic activation;
3. a rational-valued (relative) deadline (with $0 < \text{deadline} \leq \text{period}$), i. e., the time after each thread activation within which all processings of the current thread should be completed;
4. a rational-valued major frame (or “MAF”). A MAF defines the duration of a pattern of processing activation. The MAF in this case study is equal to 10.
5. a set of processings deployed on the thread. Different processings may be executed in an order which may change at each cycle. However, after a MAF duration, the same pattern of processings is repeated.

In order to simplify the scheduling problem, we have considered in this paper a pre-allocation of processings on threads, as specified in Fig. 7: that is, Navigation and Control are allocated on T1, while Monitoring and Guidance are allocated on T2 and T3, respectively. In addition, Navigation is executed at every period of T1, while Control is executed (after Navigation) on *odd* cycles only; this is denoted by the **when 1** syntax in Fig. 7.

3.4 A formal framework for real-time systems

A real-time system $\mathcal{S} = \{\mathcal{P}, \mathcal{T}, \mathcal{R}\}$ is viewed here as a set of *processings* $\mathcal{P} = \{\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_m\}$, a set of *threads* $\mathcal{T} = \{\mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_n\}$ and a set of *reactivities* $\mathcal{R} = \{\mathbf{r}_1, \mathbf{r}_2, \dots, \mathbf{r}_q\}$. A thread \mathbf{t}_i computes a usually infinite stream of processings instances.

In our setting, a thread \mathbf{t}_i is periodic, i. e., generates instances every fixed amount of time (the “period”), and is characterized by a 5-tuple $(PT_i, OT_i, DT_i, MAF_i, \mathcal{P}_i)$, where PT_i corresponds to the period, OT_i to the offset, DT_i to the deadline, MAF_i defines the duration of a pattern of processing activation \mathbf{p}_{i_k} (where \mathbf{p}_{i_k} denotes the k th processing computed in thread \mathbf{t}_i), and \mathcal{P}_i defines a subset of processings of \mathcal{P} allocated to \mathbf{t}_i .³

A processing \mathbf{p}_i is characterized by two values $WCET_i$ (Worst Case Execution Time) and PP_i (Processing Period): When a processing is activated, it is executed for at most time $WCET_i$ time units every PP_i time units.

³Note that the MAF is a per-thread property; it is quite similar to the ARINC 653 standard used in industrial civil airplane [GNC13] designs, except that the ARINC 653 is a per-CPU property.

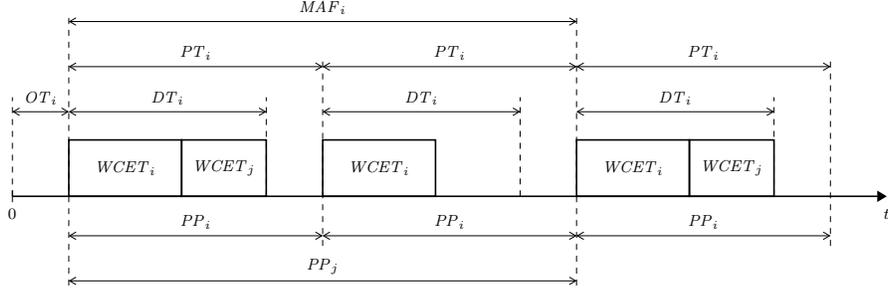


Figure 6: Real-time characteristics of the system

Example 1 Let us illustrate these definitions using Fig. 6. A single thread \mathbf{t}_i is considered, with offset OT_i , MAF MAF_i , period PT_i and deadline DT_i . This thread has two processings \mathbf{p}_i and \mathbf{p}_j , where \mathbf{p}_i is characterized by a WCET $WCET_i$ and a period PP_i and \mathbf{p}_j has WCET $WCET_j$ and period PP_j .

A reactivity is of the form $\mathbf{r}_i = ((\mathbf{p}_{i_1} \rightarrow \mathbf{p}_{i_2} \rightarrow \dots \rightarrow \mathbf{p}_{i_k}), DR_i)$ where $\mathbf{p}_{i_1}, \mathbf{p}_{i_2}, \dots, \mathbf{p}_{i_k}$ are k processings of \mathcal{P} , $(\mathbf{p}_{i_1} \rightarrow \mathbf{p}_{i_2} \rightarrow \dots \rightarrow \mathbf{p}_{i_k})$ denotes a precedence constraint, and DR_i is the maximum *time of reactivity* for \mathbf{r}_i : the end of the thread period containing the last processing \mathbf{p}_{i_k} of the precedence sequence has to be completed before the deadline DR_i . (If a reactivity is satisfied, its precedence constraint is obviously satisfied too.)

Definition 3.2 A system \mathcal{S} is schedulable if

1. $\forall \mathbf{t}_i \in \mathcal{T}$, the end of each instance of \mathbf{t}_i occurs before its relative deadline DT_i .
2. $\forall \mathbf{r}_i \in \mathcal{R}$, the end of each instance of the thread containing the last processing \mathbf{p}_{i_k} of \mathbf{r}_i occurs before DR_i .

3.5 Formalization of the case study

We formalize in the following the system, with the values given in Figs. 1 and 5 and the assignments onto threads given in Fig. 7.

3.5.1 Processings

Let \mathcal{P} denote the set of processings. This set can be defined as $\mathcal{P} = \{\mathbf{p}_{Navi}, \mathbf{p}_{Cont}, \mathbf{p}_{Moni}, \mathbf{p}_{Guid}\}$, where:

Control: $\mathbf{p}_{Cont} = (WCET_{Cont}, PP_{Cont}) = (3, 10)$.

Guidance: $\mathbf{p}_{Guid} = (WCET_{Guid}, PP_{Guid}) = (15, 60)$.

Monitoring: $\mathbf{p}_{Moni} = (WCET_{Moni}, PP_{Moni}) = (5, 20)$.

Navigation: $\mathbf{p}_{Navi} = (WCET_{Navi}, PP_{Navi}) = (1, 5)$.

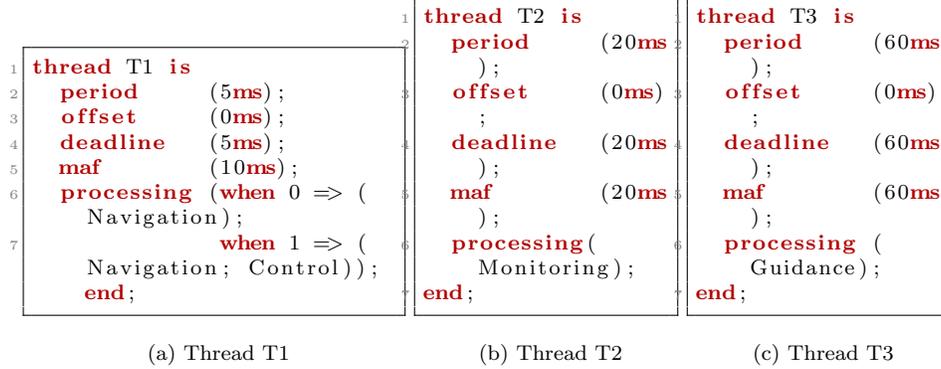


Figure 7: A typical solution of the flight control scheduling problem

3.5.2 Threads

Let $\mathcal{T} = \{\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3\}$ denote the set of threads, with:

- $\mathbf{t}_1 = (PT_1, OT_1, DT_1, MAF_1, \mathcal{P}_1) = (5, OT_1, DT_1, 10, \{\mathbf{p}_{Navi}, \mathbf{p}_{Cont}\})$.
- $\mathbf{t}_2 = (PT_2, OT_2, DT_2, MAF_2, \mathcal{P}_2) = (20, OT_2, DT_2, 20, \{\mathbf{p}_{Moni}\})$.
- $\mathbf{t}_3 = (PT_3, OT_3, DT_3, MAF_3, \mathcal{P}_3) = (60, OT_3, DT_3, 60, \{\mathbf{p}_{Guid}\})$.

3.5.3 Reactivities

Let $\mathcal{R} = \{\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3\}$ denote the set of reactivities, with:

- $\mathbf{r}_1 = ((\mathbf{p}_{Navi} \rightarrow \mathbf{p}_{Guid} \rightarrow \mathbf{p}_{Cont}), DR_1)$ with $DR_1 = 150$.
- $\mathbf{r}_2 = ((\mathbf{p}_{Navi} \rightarrow \mathbf{p}_{Cont}), DR_2)$ with $DR_2 = 15$.
- $\mathbf{r}_3 = ((\mathbf{p}_{Navi} \rightarrow \mathbf{p}_{Moni}), DR_3)$ with $DR_3 = 55$.

3.6 Objectives

Let us summarize the problems we address in this paper. Our problems take as input a real-time system, i.e.:

1. a list of processings with their WCET (for example Fig. 5) and period, and their input or output data (for example Fig. 1);
2. a set of reactivities (for example Fig. 3);
3. an allocation of processings on threads, with period, offset, deadline and MAF for each thread (for example Fig. 7).

Remark 1 Observe in Fig. 7 that the harmonic assumption on threads is respected, with threads ordered by increasing frequency as follows: T3, T2, T1.

The first problem is to formally *verify* the schedulability of the real-time system:

Scheduling verification problem:

INPUT: a real-time system

PROBLEM: formally verify that \mathcal{S} is schedulable.

Recall that schedulability also ensures that all reactivity constraints are met (from [Definition 3.2](#)).

The second problem assumes that some constants of the real-time system (deadlines, periods, offsets, WCET...) become unknown. The real-time system can then be seen as a *partially specified* or *abstract* system.

In this work, we assume that the offsets and deadlines of each thread are unknown; that is, some of the values in [Fig. 7](#) are not known anymore. The scheduling synthesis problem for our flight control system consists thus in computing the offsets and deadlines of each thread in order to fulfill the required reactivities.

Scheduling synthesis problem:

INPUT: a real-time system, a set of unknown constants

PROBLEM: exhibit valuations for the unknown constants such as \mathcal{S} is schedulable.

Recall that our synthesis problem still considers as input the periods; therefore offsets and deadlines are the main results of interest.

4 Parametric stopwatch automata

4.1 Clocks, parameters, constraints

We assume a set $\mathbb{X} = \{x_1, \dots, x_{|\mathbb{X}|}\}$ of *clocks*, i. e., real-valued variables that evolve at the same rate. A clock valuation is a function $w : \mathbb{X} \rightarrow \mathbb{R}_{\geq 0}$. We write $\vec{0}$ for the clock valuation assigning 0 to all clocks. Given $R \subseteq \mathbb{X}$, we define the *reset* of a valuation w , denoted by $[w]_R$, as follows: $[w]_R(x) = 0$ if $x \in R$, and $[w]_R(x) = w(x)$ otherwise. Given a valuation w , $d \in \mathbb{R}_+$ and $\mathbb{X}' \subseteq \mathbb{X}$, we define the *time-elapsing of w by d except for clocks in \mathbb{X}'* , denoted by $w_{\mathbb{X}'}^{\nearrow+d}$, as the clock valuation such that

$$w_{\mathbb{X}'}^{\nearrow+d}(x) = \begin{cases} w(x) & \text{if } x \in \mathbb{X}' \\ w(x) + d & \text{otherwise} \end{cases}$$

We assume a set $\mathbb{P} = \{p_1, \dots, p_{|\mathbb{P}|}\}$ of *parameters*, i. e., unknown constants. A parameter *valuation* v is a function $v : \mathbb{P} \rightarrow \mathbb{Q}_+$. We denote $\bowtie \in \{<, \leq, =, \geq, >\}$. A guard g is a constraint over $\mathbb{X} \cup \mathbb{P}$ defined by a conjunction of inequalities of the form $x \bowtie d$ or $x \bowtie p$, with $x \in \mathbb{X}$, $d \in \mathbb{N}$ and $p \in \mathbb{P}$. Given a guard g , we write $w \models v(g)$ if the expression obtained by replacing in g each $x \in \mathbb{X}$ by $w(x)$ and each $p \in \mathbb{P}$ by $v(p)$ evaluates to true.

4.2 Parametric stopwatch automata

Parametric timed automata (PTA) extend timed automata with parameters within guards and invariants in place of integer constants [[AHV93](#)]. For many real-time systems, especially when they are subject to preemptive scheduling, PTA are not sufficiently expressive. As a result, we will use here an extension of PTA with stopwatches [[CL00](#)], namely parametric stopwatch automata [[Sun+13](#)].

Definition 4.1 (PSA) A parametric stopwatch automaton (PSA) \mathcal{A} is a tuple $\mathcal{A} = (\Sigma, L, \ell_0, \mathbb{X}, \mathbb{P}, \mathcal{I}, \mathcal{S}, E)$, where:

1. Σ is a finite set of actions,
2. L is a finite set of locations,
3. $\ell_0 \in L$ is the initial location,
4. \mathbb{X} is a finite set of clocks,
5. \mathbb{P} is a finite set of parameters,
6. \mathcal{I} is the invariant, assigning to every $\ell \in L$ a guard $\mathcal{I}(\ell)$,
7. \mathcal{S} is the stop function $\mathcal{S} : \ell \rightarrow 2^{\mathbb{X}}$, assigning to every $\ell \in L$ a set of stopped clocks,
8. E is a finite set of edges $e = (\ell, g, a, R, \ell')$ where $\ell, \ell' \in L$ are the source and target locations, g is a guard, $a \in \Sigma$, and $R \subseteq \mathbb{X}$ is the set of clocks to be reset.

Stopwatch automata can be composed as usual using parallel composition on synchronized actions. Note that our clocks are *shared* by default, i.e., a same clock (i.e., with the same name) can be read, stopped or reset in several automata. The same applies to parameters.

Given a parameter valuation v and PSA \mathcal{A} , we denote by $v(\mathcal{A})$ the non-parametric structure where, for each parameter $p \in \mathbb{P}$, all occurrences of p have been replaced by $v(p)$. Any structure $v(\mathcal{A})$ is also a *stopwatch automaton* [CL00]. If $\mathcal{S}(\ell) = \emptyset$ for all $\ell \in L$, then by assuming a rescaling of the constants (multiplying all constants in $v(\mathcal{A})$ by the least common multiple of their denominators), we obtain an equivalent (integer-valued) TA, as defined in [AD94].

Let us now recall the concrete semantics of stopwatch automata.

Definition 4.2 Given a PSA $\mathcal{A} = (\Sigma, L, \ell_0, \mathbb{X}, \mathbb{P}, \mathcal{I}, \mathcal{S}, E)$, and a parameter valuation v , the semantics of $v(\mathcal{A})$ is given by the timed transition system (TTS) (S, s_0, \rightarrow) , with

1. $S = \{(\ell, w) \in L \times \mathbb{R}_{\geq 0}^{|\mathbb{X}|} \mid w \models v(\mathcal{I}(\ell))\}$,
2. $s_0 = (\ell_0, \vec{0})$,
3. \rightarrow consists of the discrete and (continuous) delay transition relations:
 - (a) discrete transitions: $(\ell, w) \xrightarrow{e} (\ell', w')$, if $(\ell, w), (\ell', w') \in S$, and there exists $e = (\ell, g, a, R, \ell') \in E$, such that $w' = [w]_R$, and $w \models v(g)$.
 - (b) delay transitions: $(\ell, w) \xrightarrow{d} (\ell, w_{\mathcal{S}(\ell)}^{\nearrow+d})$, with $d \in \mathbb{R}_{\geq 0}$, if $\forall d' \in [0, d], (\ell, w_{\mathcal{S}(\ell)}^{\nearrow+d'}) \in S$.

5 Specifying the system

Since the seminal work of Liu and Layland in [LL73], an abundant number of methods and tools have been designed to check the schedulability of real-time systems. However, while some aspects are reasonably easy (FPS, no mixed-criticality), the problem we address here is not typical for several reasons:

1. offsets may be non-null;

2. the executed processings may differ depending on the cycle;
3. the reactivities must always be met, and therefore define new, non-classical timing constraints; and, perhaps most importantly,
4. the admissible values for deadlines and offsets may not be known. Only the global end-to-end reactivity is specified.

As a consequence, we choose to follow a *model checking* based method. Model checking is known for being more expressive than analytical methods, at the cost of performance or even decidability. We show here that, although we use an undecidable formalism, we do get exact results for the instance of the problem we consider. We indeed rely on a procedure (“reachability synthesis”, formalized in e. g., [JLR15]) which is not guaranteed to terminate—but is correct whenever it does.

We present in the remainder of this section our modeling of the verification and the synthesis problem using PSA. This formalism has several advantages. First, it is helpful to model concurrent aspects of the system (different threads and processings running concurrently). Second, stopwatches can be used to model preemption. Third, parameters can be used to model the unknown constants, and solve the synthesis problem.

For now, we consider that there are no context switches in the system. We will discuss in [Section 7](#) how to introduce them.

5.1 Architecture of the solution

5.1.1 A modular solution

To model the system, we use the concurrent structure of parametric stopwatch automata so as to build a modular solution: that is, each element (thread, processing, scheduling policy) and each constraint (reactivity) is defined by a dedicated PSA. These automata are then composed by usual parallel composition on synchronization actions.

This makes our solution modular in the sense that, in the case of a modification in the system (e. g., the scheduling policy), we can safely replace one PSA with another (e. g., the FPS scheduler automaton with another scheduler PSA) without impacting the rest of the system.

5.1.2 Encoding elements and constraints as automata

We will model each processing activation as a PSA. These automata ensure that processings are activated periodically with their respective period and initial offset.

In addition, we will create one PSA for each thread: the purpose of these automata is to ensure that the processings associated with each thread are executed at the right time. In the case of our concrete problem, we assign both the Navigation and Control processings to thread T1, the monitoring process to T2 and the guidance processing to T3.

The reactivities also follow the concept of modularity. That is, each reactivity is *tested* using a single PSA. By testing (as in [Ace+03]), we mean that a reactivity fails iff a special location is reached. Therefore, ensuring the validity of the reactivities is equivalent to the unreachability of these special locations.

Finally, we will specify a scheduler automaton that encodes the scheduling policy between the different threads (in our problem, recall that the scheduling policy is fixed priority scheduling (FPS)).

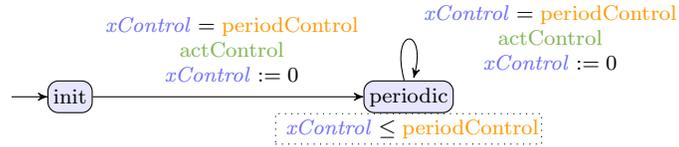


Figure 8: Automaton periodicControl

We give more details on each of these automata in the following.

5.2 Modeling periodic processing activations

To model the periodicity of the processings, we create one PSA for each processing activation. This PSA simply performs the activations in a periodic manner. Activations are modeled by a synchronization action that is used to communicate with other automata (typically the thread automaton). For example, the activation of the Control processing is denoted by `actControl`; this action will be used to synchronize between the Control activation automaton with other automata (e.g., the threads or the scheduler).

In addition, the period processing activation automaton detects whether a processing missed its (implicit) deadline (equal to its period); that is, we assume that a processing that has not finished by its next period is a situation corresponding to a deadline miss.

Each automaton features a single clock.

We present in Fig. 8 a simplified version of the `periodicControl` automaton, modeling the periodic activation of the Control processing.⁴ This automaton uses one clock `xControl` and one parameter `periodControl`. The clock `xControl` is used to measure the time between any two consecutive processing activations; it is never stopped. Note that the period `periodControl` is known beforehand, and is therefore not strictly speaking a parameter, but that makes our solution both more generic and more readable (in IMITATOR, a parameter can be statically instantiated to a constant before running the analysis).

The initial location is `init`: from then, the first occurrence of Control is immediately activated (action `actControl`), and the automaton enters the `periodic` location. Then, exactly every `periodControl` time units (guard `xControl = periodControl`), another instance of Control is activated.

5.3 Modeling threads

We create one PSA for each thread. Each of these automata contains one clock for the thread (used to measure the thread period and offset), as well as one clock per processings assigned to the thread. These processings clocks are used to measure the amount of time spent on executing these processings; these clocks can be stopped (they are therefore *stopwatches*, strictly speaking) when the processor was preempted for a higher priority task. For example in Fig. 9, the thread automaton `threadT1` contains `xT1` (the thread clock), as well as `xExecControl` and `xExecNavigation` (the

⁴Among the simplifications, we do not represent the check for the deadline miss.

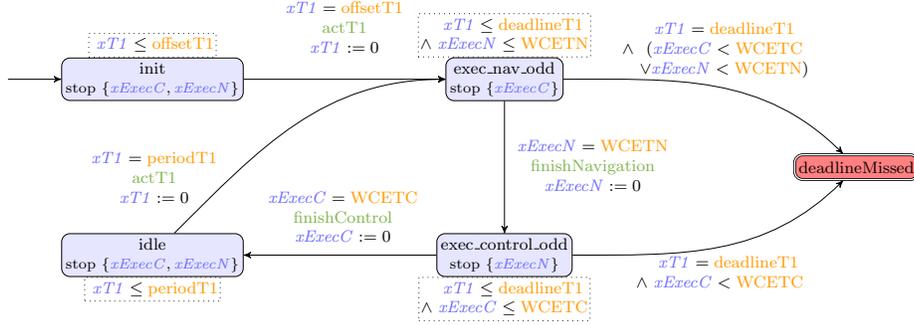


Figure 9: Fragment of automaton threadT1

clocks associated to the processings of T1). Parameters include the offset, period, and deadline of the thread, but also the WCETs of the processings assigned to this thread.

The thread automaton is responsible for:

1. encoding the initial thread offset, i. e., starting the periodic thread activation only after the offset;
2. performing the periodic thread activation;
3. executing the processings associated with the thread; and
4. detecting the deadline misses.

The clocks associated with the processings are used to measure the execution time of these processings: they are in fact stopped most of the time, except when the thread is actively executing the processing. This is in contrast with the clocks associated with the processing activation automaton, which are never stopped, as they measure a period. Then, a deadline miss occurs if the clock measuring the thread period reaches the deadline (recall that the deadline is less than or equal to the period, and therefore we can use the same clock), while the clock measuring a processing execution time is strictly less than its WCET.

We give in Fig. 9 a fragment of the automaton threadT1. We only give the odd cycle, as this is the most interesting; that is, we removed the fragment corresponding to the even cycle (only executing Navigation) between locations `init` and `exec.nav_odd` (and the transition from `idle` should go to the removed `exec.nav_even` location). The automaton uses several synchronization variables, notably the end of the processings (e. g., `finishControl`), but also the start and end of the concerned thread (e. g., `actT1` and `endT1`, not depicted in the simplified version in Fig. 9). We also abbreviate some variable names to save space (e. g., `xExecC` for `xExecControl` and `xExecN` for `xExecNavigation` or `WCETN` for `WCETNavigation`).

First, the automaton waits for the offset: that is, it stays in `init` exactly `offsetT1` time units. Then, it executes the first processing of the odd cycle, i. e., Navigation: it stays in `exec.nav_odd` until completion, i. e., for `WCETNavigation` time units.⁵ Note that this is the only location where `xExecNavigation` is elapsing, i. e., is not stopped, as

⁵In the full model, we can allow for a best case execution time, in which case the duration is nondeterministically chosen in the interval `[BCETNavigation, WCETNavigation]`.

it measures the execution time. Then, upon completion of the Navigation processing, the automaton moves to `exec_control_odd`, where Control is executed. Upon completion, it moves to idle, and waits until the clock $xT1$ reaches its period. Then, the cycle restarts and so on.

In addition, at any time, possible deadline misses are checked for. A deadline miss occurs on an odd cycle while execution Navigation whenever $xT1 = \text{deadlineT1}$ and either $xExecControl < WCETControl$ or $xExecNavigation < WCETNavigation$.⁶ When executing Control, only the execution time of Control needs to be checked.

Remark 2 *Our model is in fact more complicated as, for sake of modularity, we make no assumption in the thread automaton on how the other automata behave, notably the processings activation automata. Therefore, we allow for processings to be activated at any time, which must be taken care of in the thread automaton.*

5.4 Modeling the FPS scheduler

The FPS scheduler is modeled using an additional PSA. It reuses existing works from the literature (e.g., [Fer+07; Sun+13]), and does not represent a significant original contribution. We mainly reuse the scheduler encoding of [Sun+13], which consists of an automaton synchronizing with the rest of the system on the start and end task synchronization actions as well as the task activation actions. Whenever a new task is activated, the scheduler decides what to do depending on its current state and the respective priorities of the new and the executing tasks (if any).

Nevertheless, we had to modify this encoding due to the fact that existing scheduler automata simply schedule tasks: in our setting, the scheduler schedules both the threads and the threads’ processings. Among the various modifications, in case of preemption, our scheduler does not stop the clocks measuring the execution times of the preempted threads (because such clocks do not exist), but stop the clocks measuring the execution times of the *processings deployed on the preempted threads*.

We give in Fig. 10 an example of such a scheduler in a simplified version, with only two threads T1 and T2; the full scheduler is of course more complete. If any of the two threads get activated (`actT1` or `actT2`), the scheduler starts executing them. If a second thread gets activated, the highest priority thread (T1) is executed, while T2 is put on the waiting list (which is encoded in location `execT1waitT2`). This is the location responsible for stopping the clock of the (only) processing of T2, i.e., Monitoring (clock $xexecM$). Only after T1 has completed (`endT1`), T2 can execute. Our real scheduler is in fact significantly more complex as it has to cope with three threads, but also with special cases such as the activation of a new thread activation of t_i while executing a previous instance of t_i , etc.

5.5 Reachability synthesis

Finally, the system is schedulable if none of the “bad” locations (corresponding to deadline misses, e.g., in the thread automata) is reachable. If all parameters are

⁶This encoding is not necessarily optimal. In fact, on odd cycles, as Navigation is executed first, and followed by Control, a deadline miss can be detected earlier, i.e., if Navigation is still executed, but there is not enough time to finish the execution of Navigation and that of Control: that is, an optimized deadline miss condition could be $xT1 + WCETControl = \text{deadlineT1}$ and $xExecNavigation < WCETNavigation$. This optimization has not been implemented, so as to leave the model (relatively) simple and maintainable, but could be tested in the future.

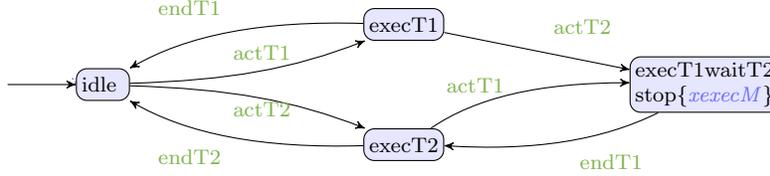


Figure 10: Encoding the FPS scheduler (simplified version)

valuated, the system is a TA, and schedulability reduces to reachability checking. If some parameters are free (i. e., the analysis is parametric), the set of valuations for which the system is schedulable exactly corresponds to the valuations for which these bad locations are unreachable, i. e., the complement of the valuations set result of reachability synthesis. This guarantees our method correctness.

6 Compositional verification of reactivities

An originality of our work—which among other reasons, notably the timing parameters, justifies our choice to use model checking—is the encoding of *reactivities*. Indeed, our goal is to verify a system, or synthesize valuations, for which all reactivities are met.

How to properly encode reactivities turned out rather subtle. Let us first exemplify the complexity of the definition of reactivities.

Example 2 Consider the third reactivity in Fig. 3 (abbreviated by NM in the following) that requires that any data transmission Meas \rightarrow Navigation \rightarrow Monitoring \rightarrow Safeguard must always be less than 55 ms. Recall that data are transmitted upon the end of a thread period.

We can see this reactivity as the start of a timer at the beginning of the last thread period of an execution of Navigation that completed before the end of an execution of task T1, where T1 is such that it is the last execution of T1 the period of which ends before the start of an execution of Monitoring; then, the timer stops following the end of the period of an execution of T1 immediately following the end of the period of T3 corresponding to the end of the aforementioned execution of Monitoring. At the end, the timer must be less than 55 ms.

In other words, this reactivity requires that any following sequence of actions should take less than 55 ms: actT1, startNavigation followed by endNavigation (without any occurrence of startNavigation in between) followed by endT1, followed by actT3 (without any occurrence of endT1 in between), startMonitoring followed by endMonitoring (without any occurrence of startMonitoring in between), followed by endT3.

Encoding reactivities is arguably the most technical part of our solution, and we tried multiple methods (either incorrect or that represented a too large overhead) before converging to this solution. Nevertheless, the solution we chose still represents a large overhead, as we will see in Section 8.

In our solution, each reactivity is encoded as a sort of *observer* automaton [Ace+03; And13]; an observer automaton observes the system behavior without interfering with it. That is, it can read clocks, and synchronize on synchronization actions, but without

impacting the rest of the systems; in particular, it must be non-blocking (except potentially once the property verified by the observer is violated). In addition, an observer often reduces to *reachability* analysis: the property encoded by the observer is violated iff a special location of the observer is reachable.

Each reactivity automaton uses a single (local) clock used to check the reactivity constraint, and synchronizes with the rest of the system on (some) synchronization labels encoding the start and end of processings and tasks.

In fact, we deviate from the principle of observer automaton by allowing it to block in some cases. Indeed, a key point in the definition of reactivities in our problem is the communication between threads as exemplified in [Example 2](#). In order to allow a generic solution for reactivities, and due to the fact that some timing parameters are unknown, we cannot make assumptions on the respective ordering of processings w.r.t. each other. Therefore, when a given processing is faster than another one (e.g., Navigation is faster than Guidance), it is not possible to know *a priori* which instance of the fast processing (e.g., Navigation) will effectively transmit its data to the following slower processing (e.g., Monitoring). As a consequence, our observer will nondeterministically “guess” from which instance of the slower processing to start its timer: this is achieved by a nondeterministic choice in the initial location of the automaton. If the guess is wrong, the observer “blocks” the system (impossibility to fire a transition or let time elapse). Note that, while blocking is usually not an admissible feature of observer automata, this is harmless in this case as, due to the nondeterministic guess and the fact that model checking explores all choices, all possible behaviors of the system are still explored by our solution.

Example 3 *Consider again reactivity NM from [Example 2](#). Consider a given instance of Navigation. If a second full instance of Navigation (including the end of thread T1) is observed before the start of T2, our observer made a wrong guess, and the observer clock is not measuring a proper reactivity, as the instance of Navigation on which the clock should be started must be the last completed instance before the start of T2. In that case, the observer simply blocks.*

6.1 Observer construction

Our solution consists in translating the sequence of starting and ending actions of threads and processings following the definition of the reactivities, while forbidding some actions in some locations to ensure the proper encoding of the definition of thread communication and reactivities. In addition, a clock measuring the reactivity is started upon the (nondeterministic) activation of the first thread, and is checked against the reactivity nominal maximum time upon completion of the last thread. If this maximum time constraint is violated, the observer enters a special “bad” location. This observer violation location is added to the list of “bad” locations in [Section 5.5](#) when performing reachability synthesis.

Example 4 *We give the observer automaton corresponding to reactivity NM in [Fig. 11](#). We abbreviate in [Fig. 11](#) the names of processings (N and M stand for Navigation and Monitoring respectively). The only clock is xNM while $reacNM$ denotes the maximum nominal reactivity for NM (55ms here). Σ_{NM} stands for this automaton alphabet; given $a \in \Sigma_{NM}$, \bar{a} denotes $\Sigma_{NM} \setminus \{a\}$ (we extend this notation to sets of actions). In addition, whenever $xNM > reacNM$ occurs in any location (except the initial location), a transition leads to the special “bad” location (these transitions are not depicted in [Fig. 11](#) for sake of clarity).*

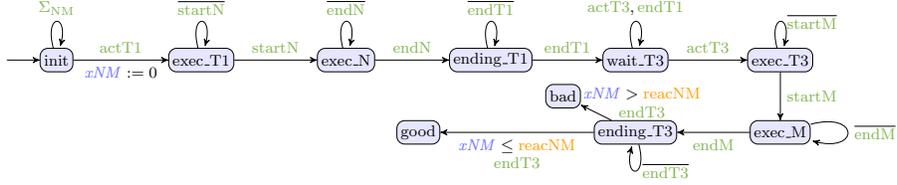


Figure 11: Encoding reactivity Navigation \rightarrow Monitoring

The nondeterministic choice is encoded in the initial location where, upon action actT1 , the automaton can either self-loop in init , or go to actT1 to try to measure the reactivity from this instance of $T1$. The blocking is encoded by the absence of transition labeled with endT1 in location wait_T3 (an alternative is to synchronize on endT1 to a sink location that also blocks time elapsing).

Both remaining reactivities in Fig. 3 follow easily from this scheme: the first reactivity (Navigation \rightarrow Guidance \rightarrow Control) follows the same principle for Navigation and Guidance, and is immediately followed by a third check for Control, while the second reactivity (Navigation \rightarrow Control) is simpler as both Navigation and Control are on the same thread.

6.2 Compositional verification and synthesis

Due to the nondeterministic choice, the verification of the reactivities entails a clear overhead to the verification (see Section 8). Verifying all three reactivities can be naturally done by adding the three observer automata to the same system, and performing synthesis on the composition of all these automata.

However, we claim that this can be done in a *compositional* fashion. Indeed, checking reactivities is checking that a constraint is met for all executions; this can be seen as a global invariant of the property “all reactivities are satisfied”, and we will verify it using observers. Observers simply *observe* the system and do not interact with it as long as the property they are verifying is not violated; therefore, independent properties can be observed by different observers using different executions. Therefore, checking that these three invariants are valid can be done separately. In the non-parametric case, we will perform three different verifications of the system, with only one reactivity automaton at a time. Then, if the “bad” locations are unreachable for the three different verifications, then the system is schedulable and the reactivities are met. In the case of synthesis, we will *intersect* the result of the synthesis applied to the three parametric models.

This compositional analysis comes in contrast with many works on scheduling, where compositionality is hard to achieve (see, e.g., [SL03; Ric05; LB05; SEL08; CPV13]). Note that our compositional verification is not necessarily specific to a *parametric* approach, and using our approach in a non-parametric setting (e.g., using UPPAAL) could also benefit from a similar compositionality.

1	processing wct	Navigation	(1ms);
2	processing wct	Guidance	(10.5ms);
3	processing wct	Control	(3ms);
4	processing wct	Monitoring	(3ms);

Figure 12: Example of Worst Case Execution Times for a system with switch time

7 Enhancing the analysis with context switches

7.1 Problem

When switching between two threads, the CPU needs to store the state of a thread, so that it can be restored later, and consequently the execution can be resumed from the same point later. Threads usually do not switch instantaneously: a certain amount of time is required for copying data. For each change in thread execution, the system must copy data before running the next thread. The time to save this state and restore another is known as *thread context-switch time*. This context-switch time between threads is small, but can be important to consider for schedulability.

Example 5 *For the threads assignment given in Fig. 7 together with the processings values in Figs. 1 and 5, we can show using the Cheddar analyzer (see Section 9.1.1) that the schedule is tight, i. e., the occupancy of the processor is 100 %. For this reason, any non-zero context-switch time implies that the system becomes non-schedulable.*

Because of the tight schedule mentioned in Example 5, in order to study the system using non-zero context-switch times, we consider the second set of (fictitious) values, given in Fig. 12. This set reduces the WCETs of the processings, and therefore allows for some non-zero context-switch time.

Following the new data from Fig. 12, let us redefine the set of processings as $\mathcal{P}' = \{\mathbf{p}_{Navi}, \mathbf{p}_{Cont}, \mathbf{p}_{Moni}, \mathbf{p}_{Guid}\}$, where:

Control: $\mathbf{p}_{Navi} = (WCET_{Navi}, PP_{Navi}) = (1, 5)$.

Guidance: $\mathbf{p}_{Cont} = (WCET_{Cont}, PP_{Cont}) = (3, 10)$.

Monitoring: $\mathbf{p}_{Moni} = (WCET_{Moni}, PP_{Moni}) = (3, 20)$.

Navigation: $\mathbf{p}_{Guid} = (WCET_{Guid}, PP_{Guid}) = (10.5, 60)$.

From now on, we consider that the switch from a thread to another one requires a (constant) switch time equal to $500 \mu s$ ⁷. Also note that, during the change of processing within the same thread (e. g., from Navigation to Control in the odd period of Thread T1 in Fig. 7), the switch remains 0, as these processings are part of the *same* thread.

7.2 Modeling the context switch

The switch time between threads is modeled as part of the scheduler automaton. For each change of execution from one thread to another, we go through an intermediate

⁷All values are confidential and therefore the given values in this paper are not the genuine ones.

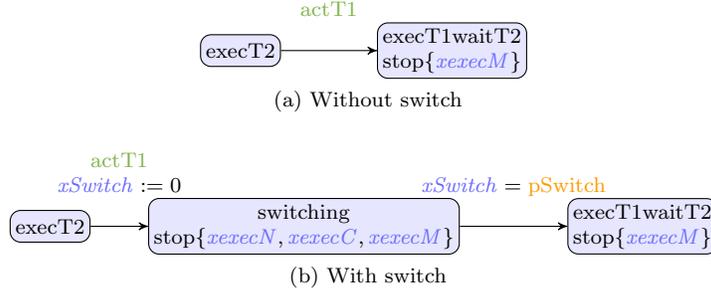


Figure 13: Encoding the FPS scheduler without and with switch (fragment)

location: upon activation of a thread implying a thread switch (recall that some thread activations may not imply an immediate thread change, if the newly activated thread has lesser priority than the currently activated thread, e.g., activating Thread T2 in location `execT1` in Fig. 10), then a clock `xSwitch` is set to 0, and counts until it reaches the context switch time. In our case, this timing is parameter `pSwitch` (this parameter is in practice assigned to its nominal value $500 \mu s$ and is therefore not truly parametric).

Example 6 We give in Fig. 13a a fragment of the original FPS scheduler from Fig. 10, corresponding to the execution of thread T2, followed by the activation of thread T1, which has higher priority and therefore requires a context-switch time. In the original version in Fig. 13a, the processor immediately starts executing T1 in location `execT1waitT2`. In contrast, in the transformed version in Fig. 13b, the processor first transits through an intermediate location `switching`, that it can only leave `pSwitch` time units later; only from there, T1 starts being executed.

Also note that, in the intermediate location `switching`, all clocks measuring the execution times of the processings associated to any of the threads of the processor (here Navigation and Control for T1 and Monitoring for T2) are stopped, as the processor is not executing any thread, but is performing the context switch.

We give in Fig. 14 the Gantt chart of the case study of interest with switch time equal to $500 \mu s$. (This Gantt chart was generated by Cheddar [Sin].)

We give in Fig. 17 in Appendix A the full version of the scheduler with three threads and the switch time between threads.

8 Experiments

8.1 Experimental environment

We modeled our network of PSA in the IMITATOR input language [And21]. IMITATOR is a parametric model checker taking as input networks of PSA extended with useful features such as synchronization actions and discrete variables. Synthesis can be performed using various properties. We use here reachability synthesis (formalized in, e.g., [JLR15]). When IMITATOR terminates (which is not guaranteed in theory), the result is always sound (but not necessarily complete), but the tool is often able to

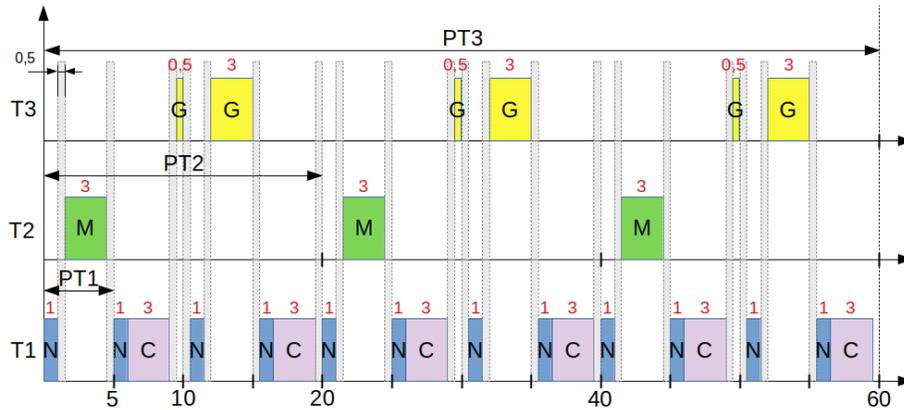


Figure 14: Gantt chart of the system GNC with switch time = $500 \mu s$

infer whether the result is *exact* (sound and complete). All analyses mentioned in this manuscript terminate with an exact result.

The translation effort was manual due to the specificity of our solution (with the exception of the scheduler, for which we started from an automated generator). However, we tried to keep our translation as systematic as possible to allow for a future automated generation from the problem input data. We made intensive use of clock resets and stopwatches for clocks not necessary at some points, in order to let IMITATOR apply inactive clock reductions.

All experiments were conducted using IMITATOR 2.10.4 “Butter Jellyfish” on an ASUS X411UN Intel Core™ i7-8550U 1.80 GHz with 8 GiB memory running Linux Mint 19 64 bits.⁸

In Sections 8.2 and 8.3, we first study the system without the extra context switch time introduced in Section 7; then, we study in Section 8.4 the overhead incurred by the context switch time.

8.2 Verification and synthesis without reactivities

In order to evaluate the overhead of the satisfaction of the reactivities, we first run analyses *without* reactivities.

8.2.1 Non-parametric model

First, a non-parametric analysis shows that the bad locations are unreachable, and therefore the system is schedulable under the nominal values given in Figs. 1 and 5.

The computation time of this non-parametric analysis, together with other parametric analyses (all without reactivities) are given in Fig. 16a.

We give in Fig. 15 the Gantt chart (obtained with Cheddar [Sin]) of this entirely instantiated model.

⁸Sources, binaries, models and results are available at imitator.fr/static/FI2021/ and at <https://www.doi.org/10.5281/zenodo.5042059>.

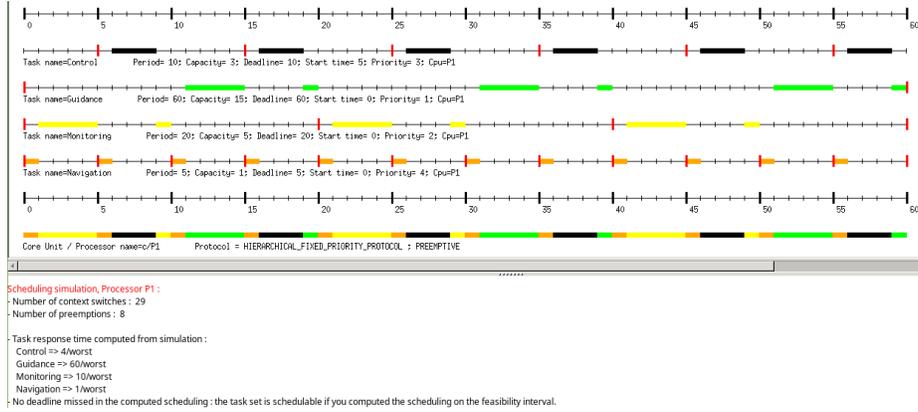


Figure 15: scheduling GNC without reactivities using Cheddar

Analysis	Time (s)
No parameter	3.1
Parametric offsets	95.8
Parametric deadlines	17.7

(a) Without switch time

Analysis	Time (s)
No parameter	17.9
Parametric offsets	5,396.3
Parametric deadlines	38.7

(b) With switch time

Figure 16: Computation times without reactivities

8.2.2 Parameterized offsets

We then parameterize offsets, i.e., we seek admissible offsets for which the system is schedulable. The constraint synthesized by IMITATOR is given in Fig. 18 in Appendix B.1. We can see that, while several conditions for schedulability are given, at least one offset must be 0 to ensure schedulability.

In order to exemplify admissible values, we exhibit some valuations satisfying this constraint in Table 11 in Appendix B.1; we also give some valuations *not* satisfying this constraint. These valuations were derived manually from the constraint, but an automatization thanks to an SMT solver would be possible.

8.2.3 Parameterized deadlines

We then parameterize deadlines, i.e., we seek admissible deadlines for which the system is schedulable. The constraint is: $\text{deadlineT2} \in [11, 20] \wedge \text{deadlineT1} \in [4, 5] \wedge \text{deadlineT3} = 60$. That is, the deadline of T3 is strict, while T1 and T2 can be relaxed while preserving schedulability.

Again, we exhibit some valuations satisfying this constraint in Table 12 in Appendix B.2.

8.3 Compositional verification of reactivities

We then solve the scheduling verification and scheduling synthesis problems with reactivities, using two methods:

Table 1: Computation times with reactivities (s)

Analysis	Monolithic	NGC	NC	NM	Compositional
No parameter	109.4	21.4	3.4	15.2	40.1
Parametric offsets	2304.0	1111.9	210.8	955.7	2278.4
Parametric deadlines	637.2	173.0	28.5	129.8	331.3

Table 2: Computation times with reactivities and switch time (s)

Analysis	Monolithic	NGC	NC	NM	Compositional
No parameter	476.5	47.5	6.5	34.5	88.5
Parametric offsets	TO	33,449.6	2915.4	TO	TO
Parametric deadlines	1,919.7	342.3	62.7	278.0	683.0

1. monolithic verification: all three reactivity automata are included in the model; and
2. compositional verification: we verify sequentially three different models, each of them including all automata modeling the system, but only one reactivity at a time.

We give the various computation times, including the overhead incurred by each reactivity, in Table 1. Table 1 shows the interest of the compositional verification over monolithic verification, as the computation time is divided by a factor 2, except in the case of parametric offsets, where the compositional verification is just a little more efficient. Also, without surprise, the most complicated reactivity (NGC) takes the longest computation time.

8.4 Switch time

We now give the computation times in the case of switch time of $500 \mu s$ in Fig. 16b and Table 2. In Table 2, “TO” denotes non-termination after 12 hours.

The constraints for offsets and deadlines synthesized by IMITATOR are given respectively in Fig. 22 in Appendix D.1 and in Fig. 23 in Appendix D.2.

We give in Tables 3 and 4 some examples of the values of parameters for which the system with switch time is schedulable (“ $\models K$ ”) or not.

Let us briefly compare the computation times of the system without switch time on the one hand, and of the system with the switch time of $500 \mu s$ on the other hand. The execution time of IMITATOR for the system with the switch time is nearly three times higher than the system without switch time, in the non-parametric case and the

Table 3: Some valuations for which the system is schedulable (without reactivities)

Valuation	offsetT1	offsetT2	offsetT3	$\models K$
v_1	0	11.5	1.5	✓
v_2	3	0	10	✓
v_3	0	5	1	✓
v_4	12	0	3	✗
v_5	3	15	0	✗

Table 4: Some valuations for which the system is schedulable (with reactivities)

Valuation	deadlineT1	deadlineT2	deadlineT3	$\models K$
v_1	4.5	20	60	✓
v_2	5	4.5	60	✓
v_3	4.5	4.5	60	✓
v_4	4	5	60	✗
v_5	4.5	4	65	✗

case of parametric deadlines. For the case of parametric offsets, it is nearly ten times higher.

9 Comparison with other tools

9.1 Comparison of our results with non-parametric tools

We perform a comparison with two other well-known tools, one from the real-time system community, namely Cheddar [Sin], and one from the timed automata community, namely UPPAAL [LPY97]. Both tools cannot handle parameters nor consider partially specified problems, and therefore can only solve the scheduling *verification* problem. Therefore, in this section, we consider the instantiated version of the system according to the nominal values given in Figs. 1 and 5. In addition, to the best of our knowledge, Cheddar cannot test the reactivities.

9.1.1 Non-parametric comparison with Cheddar

Cheddar is a real-time scheduling tool distributed under the GPL license. Cheddar is used to model software architectures of real-time systems and to check whether the system is schedulable.

We checked the system’s schedulability using Cheddar when the system is instantiated (i.e., all offsets are initialized to 0 and the deadline of each thread equal to the period). We have indicated the period, the execution time and deadline of each processings.

As result, Cheddar proves that the system in Fig. 5 without switch time between threads is schedulable and there is no deadline missed in the computed scheduling. We give in Fig. 15 the Gantt chart of this system using Cheddar. The computation time of this analysis is given in Table 7. In this solution, the number of context switches per period of T3 is 29 and the number of preemptions is 8.

Cheddar cannot give a solution to the scheduling synthesis problem since it only works with instantiated systems, so we cannot determine offsets and deadlines, and also it does not deal with reactivities.

9.1.2 Non-parametric comparison with Uppaal

We also compare the obtained results using IMITATOR with UPPAAL results (for the model without switch time). UPPAAL is a timed model checker taking as input networks of timed automata, extended with some useful features such as synchronization, integer-valued global variables, data structures and C-style functions. We wrote a

Table 5: Possible offset valuations with switch time using UPPAAL

Valuation	offsetT1	offsetT2	offsetT3	Uppaal (with reactivities)
v_1	0	11.5	1.5	✓
v_2	3	0	10	✓
v_3	0	5	1	✓
v_4	12	0	3	×
v_5	3	15	0	×

Table 6: Possible deadline valuations with switch time using UPPAAL

Valuation	deadlineT1	deadlineT2	deadlineT3	Uppaal (with reactivities)
v_1	4.5	20	60	✓
v_2	5	4.5	60	✓
v_3	4.5	4.5	60	✓
v_4	4	5	60	×
v_5	4.5	4	65	×

UPPAAL model identical to the IMITATOR model—with instantiated parameters as UPPAAL does not support parametric analyses.

As result, UPPAAL proves that the instantiated system is schedulable, both without and with reactivities. We give in Table 5 obtained results using UPPAAL when offsets are parameterized and in Table 6 when deadlines are parameterized.

9.1.3 Summary of comparisons

We give the computation times without reactivities in Table 7. Clearly, from our experiments, if the model features no parameters, Cheddar (if no reactivities are specified) or UPPAAL (if some reactivities are specified) should be used. However, none of these tools cope with uncertain constants. Therefore, despite the complexity overhead, IMITATOR should be used if some timing constants are unspecified.

9.2 “Testing” the parametric analysis

Finally, we tried to obtain additional guarantees on our *model’s* correctness. Indeed, while we can reasonably suppose that our methodology is correct and that the tools are exempt from bugs for the algorithms used here (which remains to be done formally though), a major issue is that of the manual coding of our model into the input language of IMITATOR. In order to have further guarantees, we compared several aspects of the results with other results, or with other tools, whenever applicable.

Table 7: Computation times without parameters

Analysis	Without reactivities (s)	With reactivities (s)
Cheddar	< 0.1	N/A
IMITATOR	3.086	109.404
UPPAAL	0.002	0.003

Table 8: Possible offset valuations (with reactivities) checked using Cheddar and UPPAAL

Valuation	offsetT1	offsetT2	offsetT3	$\models K$	Cheddar (without reactivities)	Uppaal (with reactivities)
v_1	0	2	1	✓	✓	✓
v_2	4	0	10	✓	✓	✓
v_3	2	10	0	✓	✓	✓
v_4	0	9	0	✓	✓	✓
v_5	2	12	1	✗	✗	✗
v_6	5	9	0	✗	✗	✗

Table 9: Possible deadline valuations (with reactivities) checked using Cheddar and UPPAAL

Valuation	deadlineT1	deadlineT2	deadlineT3	$\models K$	Cheddar (without reactivities)	Uppaal (with reactivities)
v_1	5	20	60	✓	✓	✓
v_2	4	11	60	✓	✓	✓
v_3	5	15	60	✓	✓	✓
v_4	4	20	60	✓	✓	✓
v_5	3	11	60	✗	✗	✗
v_6	4	9	55	✗	✗	✗

9.2.1 Using non-parametric model checking

In order to increase our confidence in the results obtained with IMITATOR in [Section 8.2](#), we will first *test* that sampled valuations from the parametric constraint synthesized by IMITATOR are indeed proved schedulable (resp. non-schedulable) by non-parametric tools whenever they belong (resp. do not belong) to the constraint synthesized by IMITATOR. Once more, we do so using both a popular tool in the real-time systems community (Cheddar) and a non-parametric timed model checker (UPPAAL).

Model with reactivities First, we fix the deadlines, and we vary the offsets according to the constraint synthesized in [Section 8.3](#). We sample four valuations of this constraint, and give them in [Table 8](#) (v_1 to v_4); we also add two valuations (v_5 and v_6) not belonging to the constraint synthesized by IMITATOR. For each of these valuations, we test using Cheddar and UPPAAL whether the system is schedulable.

We give in [Table 8](#) obtained results using Cheddar and UPPAAL when deadlines are instantiated (and offsets remain parameterized) and in [Table 9](#) when offsets are instantiated (and deadlines remain parameterized). As one can see from [Tables 8](#) and [9](#), all results are consistent. Recall that this does not formally *prove* the correctness of our method, but increases our confidence by *testing sample points*. Still, if one considers that UPPAAL or Cheddar are reliable tools and that our model is entirely correct, once a given valuation is chosen from the constraint output by IMITATOR, checking again its correctness using one of the aforementioned tools is a good way to assess the validity of the whole process.

9.2.2 Using constraints comparisons

We now perform additional tests on the results of IMITATOR.

Table 10: Possible offset valuations in the difference of constraints without and with reactivities

Valuation	offsetT1	offsetT2	offsetT3	Cheddar	Uppaal
v_1	0	2	0	✓	×
v_2	0	4	1	✓	×
v_3	0	3	1	✓	×

Model without switch time We consider here the constraints for the full system including reactivities but excluding the switch time [Section 8.3](#) (the case of the switch time gave similar results).

Constraints comparisons First, we verified using PolyOp⁹ that the constraint obtained by monolithic verification is equal to the intersection of the 3 constraints (reactivity NC, reactivity NM and reactivity NGC) obtained by separate verifications, on the one hand when offsets are parameterized, and on the other hand when deadlines are parameterized.

Second, we checked that the results with reactivities are included in the constraint without reactivities in all three cases (no parameter, parametric offsets, parametric deadlines). Indeed, the model with reactivities is more constrained, and therefore its admissible valuations set shall be included in or equal to the valuations set without reactivity constraints.

Constraints difference We give below the difference of constraints without reactivities and constraints with reactivities when offsets are parameterized:

$$\text{offsetT3} + 5 > \text{offsetT2} \wedge \text{offsetT1} = 0 \wedge \text{offsetT2} \in (1, 5] \wedge \text{offsetT3} \in [0, 1]$$

This shows that the two constraints are not equal: some valuations ensure schedulability when reactivities are not considered, but do not ensure schedulability under reactivity constraints. *This is a major outcome of our experiments, as it justifies for the analysis under reactivity constraints.* That is, tools that are not able to test schedulability under reactivity constraints (such as Cheddar) will give incorrect results for this case study.

We present in [Table 10](#) some examples of values of offsets in the difference of constraints without and with reactivities: as expected, Cheddar mistakenly guarantees the system is correct while UPPAAL shows it is not, due to some violated reactivities.

Model with switch time We finally perform additional verifications on the results of [Section 8.4](#). We verify using PolyOp that the constraint with switch time obtained by monolithic verification is equal to the intersection of the 3 constraints (reactivity NC, reactivity NM and reactivity NGC) obtained by separate verifications when deadlines are parameterized, just as we did in [Section 6](#). We also checked that the result with reactivity NC is included in the constraint without reactivities when offsets are parameterized; and similarly for the result with all 3 reactivities. Not all situations could be considered, as some analyses reach time out (see [Table 2](#)).

⁹A simple interface over the Parma Polyhedra Library [BHZ08], available at github.com/etienneandre/PolyOp, and that allows for polyhedral computations such as intersection or difference, as well as polyhedral checks such as equality or (strict) inclusion.

We also rechecked the sample results obtained by IMITATOR in [Tables 3 and 4](#) using UPPAAL. We did not use Cheddar in this example because, to the best of our knowledge, Cheddar cannot apply switch time between threads.

10 Conclusion and perspectives

We proposed an approach to synthesize timing valuations ensuring schedulability of the flight control of a space launcher. A key issue is to ensure that the system reactivities are met—for which we proposed a compositional solution.

Our implementation of the flight control system as an extension of the parametric timed automata formalism using IMITATOR allows to determine offsets and deadlines of each thread taking into account that all reactivities are satisfied, and allows to ensure formally that the FPS type scheduling can be a solution for our problem. We build a modular solution, i.e., we specified an automaton for each element of our system (thread, processing, scheduling policy) and for each constraint (reactivity). The interest of using IMITATOR as the main tool of our approach is that it allows to analyze a system with parameters in order to determine the possible values of those parameters, unlike other existing tools (e.g., Cheddar and UPPAAL) which treat only initialized systems. In addition, we showed that the reactivity constraints are important as, without them, wrong valuations can be derived.

Future works Due to the efficiency gap of an order of magnitude, combining some non-parametric analyses (e.g., with UPPAAL or Cheddar) with parametric analyses (IMITATOR) would be an interesting future work.

The harmonic periods are a strong assumption of the problem. Tuning our solution to benefit from this assumption is on our agenda. This may indeed allow us to reuse some clocks, and therefore reduce the number of clocks; it is well-known that the model-checking problem is exponential in the number of clocks.

In addition, *synthesizing* the admissible values for the context switch time, i.e., making this time a parameter, seems interesting as it derives admissible values of the processor context switch speed for which the system can be scheduled.

So far, our method allows us to prove formally properties for systems that contain only a limited number of threads and processings. In order to scale up the method, it would be interesting to combine it with Bini-Buttazzo’s uniprocessor taskset generation method [[BB05](#); [ESD10](#)].

We envisage two tracks for longer-term future works:

1. Generalizing the flight control scheduling problem by automatically synthesizing the allocations of processings on threads. This generalization raises first the issue of modeling such a problem (how to model an allocation with a parameter) and second the classical combinatorial explosion of states.
2. Applying this approach to the automatic synthesis of the launcher sequential, i.e., of the scheduling of all the system events necessary to fulfill a mission: ignition and shut-down of stages, release of firing, release of payloads, etc.

Acknowledgements

We would like to thank the anonymous reviewers for useful comments.

References

- [AAM06] Yasmina Abdeddaïm, Eugene Asarin, and Oded Maler. “Scheduling with timed automata”. In: *Theoretical Computer Science* 354.2 (Mar. 2006), pp. 272–300. DOI: [10.1016/j.tcs.2005.11.018](https://doi.org/10.1016/j.tcs.2005.11.018) (cit. on p. 3).
- [Ace+03] Luca Aceto, Patricia Bouyer, Augusto Burgueño, and Kim Guldstrand Larsen. “The power of reachability testing for timed automata”. In: *Theoretical Computer Science* 300.1-3 (2003), pp. 411–475. DOI: [10.1016/S0304-3975\(02\)00334-1](https://doi.org/10.1016/S0304-3975(02)00334-1) (cit. on pp. 13, 17).
- [AD94] Rajeev Alur and David L. Dill. “A theory of timed automata”. In: *Theoretical Computer Science* 126.2 (Apr. 1994), pp. 183–235. ISSN: 0304-3975. DOI: [10.1016/0304-3975\(94\)90010-8](https://doi.org/10.1016/0304-3975(94)90010-8) (cit. on pp. 2, 3, 12).
- [AHV93] Rajeev Alur, Thomas A. Henzinger, and Moshe Y. Vardi. “Parametric real-time reasoning”. In: *STOC* (May 16–18, 1993). Ed. by S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal. San Diego, California, United States: ACM, 1993, pp. 592–601. DOI: [10.1145/167088.167242](https://doi.org/10.1145/167088.167242) (cit. on pp. 2, 3, 11).
- [AHW18] Étienne André, Ichiro Hasuo, and Masaki Waga. “Offline timed pattern matching under uncertainty”. In: *ICECCS* (Dec. 12–14, 2018). Ed. by Anthony Widjaja Lin and Jun Sun. Melbourne, Australia: IEEE Computer Society, 2018, pp. 10–20. DOI: [10.1109/ICECCS2018.2018.00010](https://doi.org/10.1109/ICECCS2018.2018.00010) (cit. on p. 2).
- [AL17] Étienne André and Didier Lime. “Liveness in L/U-Parametric Timed Automata”. In: *ACSD* (June 25–30, 2017). Ed. by Alex Legay and Klaus Schneider. Zaragoza, Spain: IEEE, 2017, pp. 9–18. DOI: [10.1109/ACSD.2017.19](https://doi.org/10.1109/ACSD.2017.19) (cit. on p. 2).
- [ALR18] Étienne André, Didier Lime, and Mathias Ramparison. “TCTL model checking lower/upper-bound parametric timed automata without invariants”. In: *FORMATS* (Sept. 4–6, 2018). Ed. by David N. Jansen and Pavithra Prabhakar. Vol. 11022. Lecture Notes in Computer Science. Beijing, China: Springer, 2018, pp. 1–17. DOI: [10.1007/978-3-030-00151-3_3](https://doi.org/10.1007/978-3-030-00151-3_3) (cit. on p. 2).
- [AM01] Yasmina Abdeddaïm and Oded Maler. “Job-Shop Scheduling Using Timed Automata”. In: *CAV* (July 18–22, 2001). Ed. by Gérard Berry, Hubert Comon, and Alain Finkel. Vol. 2102. Lecture Notes in Computer Science. Paris, France: Springer, 2001, pp. 478–492. ISBN: 3-540-42345-1. DOI: [10.1007/3-540-44585-4_46](https://doi.org/10.1007/3-540-44585-4_46) (cit. on p. 3).
- [AM02] Yasmina Abdeddaïm and Oded Maler. “Preemptive Job-Shop Scheduling using Stopwatch Automata”. In: *TACAS* (Apr. 8–12, 2002). Ed. by Joost-Pieter Katoen and Perdita Stevens. Vol. 2280. Lecture Notes in Computer Science. Grenoble, France: Springer-Verlag, Apr. 2002, pp. 113–126. DOI: [10.1007/3-540-46002-0_9](https://doi.org/10.1007/3-540-46002-0_9) (cit. on p. 3).
- [And+09] Étienne André, Thomas Chatain, Emmanuelle Encrenaz, and Laurent Fribourg. “An Inverse Method for Parametric Timed Automata”. In: *International Journal of Foundations of Computer Science* 20.5 (Oct. 2009), pp. 819–836. DOI: [10.1142/S0129054109006905](https://doi.org/10.1142/S0129054109006905) (cit. on p. 4).

- [And+19] Étienne André, Laurent Fribourg, Jean-Marc Mota, and Romain Soulat. “Verification of an industrial asynchronous leader election algorithm using abstractions and parametric model checking”. In: *VMCAI* (Jan. 13–15, 2019). Ed. by Constantin Enea and Ruzica Piskac. Vol. 11388. Lecture Notes in Computer Science. Lisbon, Portugal: Springer, 2019, pp. 409–424. DOI: [10.1007/978-3-030-11245-5_19](https://doi.org/10.1007/978-3-030-11245-5_19) (cit. on p. 4).
- [And13] Étienne André. “Observer Patterns for Real-Time Systems”. In: *ICECCS* (July 17–19, 2013). Ed. by Yang Liu and Andrew Martin. Singapore: IEEE Computer Society, July 2013, pp. 125–134. DOI: [10.1109/ICECCS.2013.26](https://doi.org/10.1109/ICECCS.2013.26) (cit. on p. 17).
- [And17] Étienne André. “A unified formalism for monoprocessor schedulability analysis under uncertainty”. In: *FMICS-AVoCS* (Sept. 18–20, 2017). Ed. by Ana Cavalcanti, Laure Petrucci, and Cristina Seceleanu. Vol. 10471. Lecture Notes in Computer Science. Torino, Italy: Springer, 2017, pp. 100–115. DOI: [10.1007/978-3-319-67113-0_7](https://doi.org/10.1007/978-3-319-67113-0_7) (cit. on pp. 2, 4).
- [And19] Étienne André. “What’s decidable about parametric timed automata?” In: *International Journal on Software Tools for Technology Transfer* 21.2 (Apr. 2019), pp. 203–219. DOI: [10.1007/s10009-017-0467-0](https://doi.org/10.1007/s10009-017-0467-0) (cit. on p. 2).
- [And21] Étienne André. “IMITATOR 3: Synthesis of timing parameters beyond decidability”. In: *CAV* (July 18–23, 2021). Ed. by Rustan Leino and Alexandra Silva. Vol. 12759. Lecture Notes in Computer Science. virtual: Springer, 2021, pp. 1–14. DOI: [10.1007/978-3-030-81685-8_26](https://doi.org/10.1007/978-3-030-81685-8_26) (cit. on pp. 2, 21).
- [AS13] Étienne André and Romain Soulat. *The Inverse Method*. FOCUS Series in Computer Engineering and Information Technology. ISTE Ltd and John Wiley & Sons Inc., 2013. ISBN: 9781848214477 (cit. on p. 4).
- [BB04] Enrico Bini and Giorgio C. Buttazzo. “Schedulability Analysis of Periodic Fixed Priority Systems”. In: *IEEE Transactions on Computers* 53.11 (2004), pp. 1462–1473. DOI: [10.1109/TC.2004.103](https://doi.org/10.1109/TC.2004.103) (cit. on p. 3).
- [BB05] Enrico Bini and Giorgio C Buttazzo. “Measuring the Performance of Schedulability Tests”. In: *Real-Time Systems* 30.1-2 (2005), pp. 129–154. DOI: [10.1007/s11241-005-0507-9](https://doi.org/10.1007/s11241-005-0507-9) (cit. on p. 29).
- [BB97] Iain Bate and Alan Burns. “Schedulability analysis of fixed priority real-time systems with offsets”. In: *RTS* (June 11–13, 1997). Toledo, Spain: IEEE Computer Society, 1997, pp. 153–160. DOI: [10.1109/EMWRTS.1997.613776](https://doi.org/10.1109/EMWRTS.1997.613776) (cit. on p. 3).
- [Ben+15] Nikola Beneš, Peter Bezděk, Kim Gulstrand Larsen, and Jiří Srba. “Language Emptiness of Continuous-Time Parametric Timed Automata”. In: *ICALP, Part II* (July 6–10, 2015). Ed. by Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann. Vol. 9135. Lecture Notes in Computer Science. Kyoto, Japan: Springer, July 2015, pp. 69–81. DOI: [10.1007/978-3-662-47666-6_6](https://doi.org/10.1007/978-3-662-47666-6_6) (cit. on p. 2).
- [Bér+16] Béatrice Bérard, Serge Haddad, Aleksandra Jovanovic, and Didier Lime. “Interrupt Timed Automata with Auxiliary Clocks and Parameters”. In: *Fundamenta Informatica* 143.3-4 (2016), pp. 235–259. DOI: [10.3233/FI-2016-1313](https://doi.org/10.3233/FI-2016-1313) (cit. on p. 4).

- [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”. In: *Science of Computer Programming* 72.1–2 (2008), pp. 3–21. DOI: [10.1016/j.scico.2007.08.001](https://doi.org/10.1016/j.scico.2007.08.001) (cit. on p. 28).
- [BL09] Laura Bozzelli and Salvatore La Torre. “Decision problems for lower/upper bound parametric timed automata”. In: *Formal Methods in System Design* 35.2 (2009), pp. 121–151. DOI: [10.1007/s10703-009-0074-0](https://doi.org/10.1007/s10703-009-0074-0) (cit. on p. 2).
- [CC99] Sérgio Vale Aguiar Campos and Edmund M. Clarke. “Analysis and Verification of Real-Time Systems Using Quantitative Symbolic Algorithms”. In: *International Journal on Software Tools for Technology Transfer* 2.3 (1999), pp. 260–269. DOI: [10.1007/s100090050033](https://doi.org/10.1007/s100090050033) (cit. on p. 3).
- [Che+09] Rémy Chevallier, Emmanuelle Encrenaz-Tiphène, Laurent Fribourg, and Weiwen Xu. “Timed Verification of the Generic Architecture of a Memory Circuit Using Parametric Timed Automata”. In: *Formal Methods in System Design* 34.1 (Feb. 2009), pp. 59–81. DOI: [10.1007/s10703-008-0061-x](https://doi.org/10.1007/s10703-008-0061-x) (cit. on p. 2).
- [CL00] Franck Cassez and Kim Guldstrand Larsen. “The Impressive Power of Stopwatches”. In: *CONCUR* (Aug. 22–25, 2000). Ed. by Catuscia Palamidessi. Vol. 1877. Lecture Notes in Computer Science. University Park, PA, USA: Springer, 2000, pp. 138–152. DOI: [10.1007/3-540-44618-4_12](https://doi.org/10.1007/3-540-44618-4_12) (cit. on pp. 2, 11, 12).
- [CPR08] Alessandro Cimatti, Luigi Palopoli, and Yusi Ramadian. “Symbolic Computation of Schedulability Regions Using Parametric Timed Automata”. In: *RTSS* (Nov. 30–Dec. 3, 2008). Barcelona, Spain: IEEE Computer Society, 2008, pp. 80–89. ISBN: 978-0-7695-3477-0. DOI: [10.1109/RTSS.2008.36](https://doi.org/10.1109/RTSS.2008.36) (cit. on pp. 2, 4).
- [CPV13] Laura Carnevali, Alessandro Pinzuti, and Enrico Vicario. “Compositional Verification for Hierarchical Scheduling of Real-Time Systems”. In: *IEEE Transactions on Software Engineering* 39.5 (May 2013), pp. 638–657. ISSN: 1939-3520. DOI: [10.1109/TSE.2012.54](https://doi.org/10.1109/TSE.2012.54) (cit. on p. 19).
- [ESD10] Paul Emberson, Roger Stafford, and Robert I. Davis. “Techniques For The Synthesis Of Multiprocessor Tasksets”. In: *WATERS* (July 6, 2010). Brussels, Belgium, July 2010, pp. 6–11 (cit. on p. 29).
- [Fan+16] Bingbing Fang, Guoqiang Li, Daniel Sun, and Hongming Cai. “Schedulability Analysis of Timed Regular Tasks by Under-Approximation on WCET”. In: *SETTA*. Ed. by Martin Fränzle, Deepak Kapur, and Naijun Zhan. Vol. 9984. Lecture Notes in Computer Science. Beijing, China, 2016, pp. 147–162. DOI: [10.1007/978-3-319-47677-3_10](https://doi.org/10.1007/978-3-319-47677-3_10) (cit. on p. 4).
- [Fer+07] Elena Fersman, Pavel Krcál, Paul Pettersson, and Wang Yi. “Task automata: Schedulability, decidability and undecidability”. In: *Information and Computation* 205.8 (2007), pp. 1149–1172. DOI: [10.1016/j.ic.2007.01.009](https://doi.org/10.1016/j.ic.2007.01.009) (cit. on pp. 3, 16).
- [FJ13] Léa Fanchon and Florent Jacquemard. “Formal Timing Analysis Of Mixed Music Scores”. In: *ICMC* (Aug. 12–16, 2013). Perth, Australia: Michigan Publishing, Aug. 2013 (cit. on p. 2).

- [For+10] Julien Forget, Frédéric Boniol, Emmanuel Grolleau, David Lesens, and Claire Pagetti. “Scheduling Dependent Periodic Tasks without Synchronization Mechanisms”. In: *RTAS* (Apr. 12–15, 2010). Ed. by Marco Caccamo. Stockholm, Sweden: IEEE Computer Society, 2010, pp. 301–310. DOI: [10.1109/RTAS.2010.26](https://doi.org/10.1109/RTAS.2010.26) (cit. on pp. 4, 6).
- [Fri+12] Laurent Fribourg, David Lesens, Pierre Moro, and Romain Soulat. “Robustness Analysis for Scheduling Problems using the Inverse Method”. In: *TIME* (Sept. 12–14, 2012). Ed. by Mark Reynolds, Paolo Terenziani, and Ben Moszkowski. Leicester, UK: IEEE Computer Society Press, Sept. 2012, pp. 73–80. DOI: [10.1109/TIME.2012.10](https://doi.org/10.1109/TIME.2012.10) (cit. on p. 4).
- [GNC13] Mainak Ghoshhajra, Sabitha Nair, and Ananda CM Cm. “ARINC 653 API and its application – An insight into Avionics System Case Study”. In: *Defence science journal* 63 (Apr. 2013). DOI: [10.14429/dsj.63.4268](https://doi.org/10.14429/dsj.63.4268) (cit. on p. 8).
- [Hun+02] Thomas Hune, Judi Romijn, Mariëlle Stoelinga, and Frits W. Vaandrager. “Linear parametric model checking of timed automata”. In: *Journal of Logic and Algebraic Programming* 52-53 (2002), pp. 183–220. DOI: [10.1016/S1567-8326\(02\)00037-1](https://doi.org/10.1016/S1567-8326(02)00037-1) (cit. on p. 2).
- [JLR15] Aleksandra Jovanović, Didier Lime, and Olivier H. Roux. “Integer Parameter Synthesis for Real-Time Systems”. In: *IEEE Transactions on Software Engineering* 41.5 (2015), pp. 445–461. DOI: [10.1109/TSE.2014.2357445](https://doi.org/10.1109/TSE.2014.2357445) (cit. on pp. 13, 21).
- [LB05] Giuseppe Lipari and Enrico Bini. “A methodology for designing hierarchical scheduling systems”. In: *Journal of Embedded Computing* 1.2 (Apr. 2005), pp. 257–269. ISSN: 1875-9025 (cit. on p. 19).
- [Le+13] Thi Thieu Hoa Le, Luigi Palopoli, Roberto Passerone, and Yusi Ramadian. “Timed-automata based schedulability analysis for distributed firm real-time systems: a case study”. In: *International Journal on Software Tools for Technology Transfer* 15.3 (2013), pp. 211–228. DOI: [10.1007/s10009-012-0245-y](https://doi.org/10.1007/s10009-012-0245-y) (cit. on p. 4).
- [Lim+09] Didier Lime, Olivier H. Roux, Charlotte Seidner, and Louis-Marie Traonouez. “Romeo: A Parametric Model-Checker for Petri Nets with Stopwatches”. In: *TACAS* (Mar. 22–29, 2009). Ed. by Stefan Kowalewski and Anna Philippou. Vol. 5505. Lecture Notes in Computer Science. York, United Kingdom: Springer, Mar. 2009, pp. 54–57. DOI: [10.1007/978-3-642-00768-2_6](https://doi.org/10.1007/978-3-642-00768-2_6) (cit. on p. 4).
- [LL73] C. L. Liu and James W. Layland. “Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment”. In: *Journal of the ACM* 20.1 (1973), pp. 46–61. ISSN: 0004-5411. DOI: [10.1145/321738.321743](https://doi.org/10.1145/321738.321743) (cit. on p. 12).
- [LPY97] Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. “UPPAAL in a Nutshell”. In: *International Journal on Software Tools for Technology Transfer* 1.1-2 (1997), pp. 134–152. DOI: [10.1007/s100090050010](https://doi.org/10.1007/s100090050010) (cit. on p. 25).

- [Lut+19] Lars Luthmann, Timo Gerecht, Andreas Stephan, Johannes Bürdek, and Malte Lochau. “Minimum/maximum delay testing of product lines with unbounded parametric real-time constraints”. In: *Journal of Systems and Software* 149 (2019), pp. 535–553. DOI: [10.1016/j.jss.2018.12.028](https://doi.org/10.1016/j.jss.2018.12.028) (cit. on p. 2).
- [Mik+10] Marius Mikucionis, Kim Guldstrand Larsen, Jacob Illum Rasmussen, Brian Nielsen, Arne Skou, Steen Ulrik Palm, Jan Storbak Pedersen, and Poul Hougaard. “Schedulability Analysis Using Uppaal: Herschel-Planck Case Study”. In: *ISoLA, Part II* (Oct. 18–21, 2010). Ed. by Tiziana Margaria and Bernhard Steffen. Vol. 6416. Lecture Notes in Computer Science. Heraklion, Crete, Greece: Springer, 2010, pp. 175–190. DOI: [10.1007/978-3-642-16561-0_21](https://doi.org/10.1007/978-3-642-16561-0_21) (cit. on p. 4).
- [Mil00] Joseph S. Miller. “Decidability and Complexity Results for Timed Automata and Semi-linear Hybrid Automata”. In: *HSCC* (Mar. 23–25, 2000). Ed. by Nancy A. Lynch and Bruce H. Krogh. Vol. 1790. Lecture Notes in Computer Science. Pittsburgh, PA, USA: Springer, 2000, pp. 296–309. ISBN: 3-540-67259-1. DOI: [10.1007/3-540-46430-1_26](https://doi.org/10.1007/3-540-46430-1_26) (cit. on p. 2).
- [NWY99] Christer Norström, Anders Wall, and Wang Yi. “Timed Automata as Task Models for Event-Driven Systems”. In: *RTCSA* (Dec. 13–16, 1999). Hong Kong, China: IEEE Computer Society, 1999, pp. 182–189. DOI: [10.1109/RTCSA.1999.811218](https://doi.org/10.1109/RTCSA.1999.811218) (cit. on p. 3).
- [OGL06] Iulian Ober, Susanne Graf, and David Lesens. “Modeling and Validation of a Software Architecture for the Ariane-5 Launcher”. In: *FMOODS* (June 14–16, 2006). Ed. by Roberto Gorrieri and Heike Wehrheim. Vol. 4037. Lecture Notes in Computer Science. Bologna, Italy: Springer, 2006, pp. 48–62. DOI: [10.1007/11768869_6](https://doi.org/10.1007/11768869_6) (cit. on p. 5).
- [Par+16] Baptiste Parquier, Laurent Rioux, Rafik Henia, Romain Soulat, Olivier H. Roux, Didier Lime, and Étienne André. “Applying parametric model-checking techniques for reusing real-time critical systems”. In: *FTSCS* (Nov. 14, 2016). Ed. by Cyrille Artho and Peter Csaba Ölveczky. Vol. 694. Communications in Computer and Information Science. Tokyo, Japan: Springer, Nov. 2016, pp. 129–144. DOI: [10.1007/978-3-319-53946-1_8](https://doi.org/10.1007/978-3-319-53946-1_8) (cit. on p. 4).
- [Ric05] Kai Richter. “Compositional scheduling analysis using standard event models: The SymTA/S approach”. PhD thesis. University of Braunschweig - Institute of Technology, 2005. DOI: [10.24355/dbbs.084-200511080100-362](https://doi.org/10.24355/dbbs.084-200511080100-362) (cit. on p. 19).
- [SAL15] Youcheng Sun, Étienne André, and Giuseppe Lipari. “Verification of Two Real-Time Systems Using Parametric Timed Automata”. In: *WATERS* (July 7, 2015). Ed. by Sophie Quinon and Tullio Vardanega. Lund, Sweden, July 2015 (cit. on p. 4).
- [SEL08] Insik Shin, Arvind Easwaran, and Insup Lee. “Hierarchical Scheduling Framework for Virtual Clustering of Multiprocessors”. In: *ECRTS* (July 2–4, 2008). Prague, Czech Republic: IEEE Computer Society, 2008, pp. 181–190. DOI: [10.1109/ECRTS.2008.28](https://doi.org/10.1109/ECRTS.2008.28) (cit. on p. 19).

- [Sin] Frank Singhoff. *The Cheddar project: a GPL real-time scheduling analyzer*. <http://beru.univ-brest.fr/singhoff/cheddar/> (cit. on pp. 21, 22, 25).
- [SL03] Insik Shin and Insup Lee. “Periodic Resource Model for Compositional Real-Time Guarantees”. In: *RTSS* (Dec. 3–5, 2003). Cancún, Mexico: IEEE Computer Society, 2003, pp. 2–13. DOI: [10.1109/REAL.2003.1253249](https://doi.org/10.1109/REAL.2003.1253249) (cit. on p. 19).
- [SL14] Youcheng Sun and Giuseppe Lipari. “A Weak Simulation Relation for Real-Time Schedulability Analysis of Global Fixed Priority Scheduling Using Linear Hybrid Automata”. In: *RTNS* (Oct. 8–10, 2014). Ed. by Mathieu Jan, Belgacem Ben Hedia, Joël Goossens, and Claire Maiza. Versailles, France: ACM, 2014, p. 35. DOI: [10.1145/2659787.2659814](https://doi.org/10.1145/2659787.2659814) (cit. on p. 3).
- [Sun+13] Youcheng Sun, Romain Soulat, Giuseppe Lipari, Étienne André, and Laurent Fribourg. “Parametric Schedulability Analysis of Fixed Priority Real-Time Distributed Systems”. In: *FSTCS* (Oct. 29–30, 2013). Ed. by Cyrille Artho and Peter Őlveczky. Vol. 419. Communications in Computer and Information Science. Auckland, New Zealand: Springer, Oct. 2013, pp. 212–228. DOI: [10.1007/978-3-319-05416-2_14](https://doi.org/10.1007/978-3-319-05416-2_14) (cit. on pp. 2, 4, 11, 16).
- [Sun+14] Youcheng Sun, Giuseppe Lipari, Romain Soulat, Laurent Fribourg, and Nicolas Markey. “Component-based analysis of hierarchical scheduling using linear hybrid automata”. In: *ERTCS* (Aug. 20–22, 2014). Chongqing, China: IEEE Computer Society, 2014, pp. 1–10. DOI: [10.1109/RTCSA.2014.6910502](https://doi.org/10.1109/RTCSA.2014.6910502) (cit. on p. 3).
- [TLR09] Louis-Marie Traonouez, Didier Lime, and Olivier H. Roux. “Parametric Model-Checking of Stopwatch Petri Nets”. In: *Journal of Universal Computer Science* 15.17 (2009), pp. 3273–3304. DOI: [10.3217/jucs-015-17-3273](https://doi.org/10.3217/jucs-015-17-3273) (cit. on p. 4).
- [WME92] Farn Wang, Aloysius K. Mok, and E. Allen Emerson. “Formal Specification of Synchronous Distributed Real-Time Systems by APTL”. In: *ICSE* (May 11–15, 1992). Ed. by Tony Montgomery, Lori A. Clarke, and Carlo Ghezzi. Melbourne, Australia: ACM Press, 1992, pp. 188–198. DOI: [10.1145/143062.143113](https://doi.org/10.1145/143062.143113) (cit. on p. 3).
- [YMW97] Jin Yang, Aloysius K. Mok, and Farn Wang. “Symbolic Model Checking for Event-Driven Real-Time Systems”. In: *ACM Transactions on Programming Languages and Systems* 19.2 (1997), pp. 386–412. DOI: [10.1145/244795.244803](https://doi.org/10.1145/244795.244803) (cit. on p. 3).

A Full scheduler

We give in Fig. 17 the full version of the scheduler with three threads and the switch time between threads.

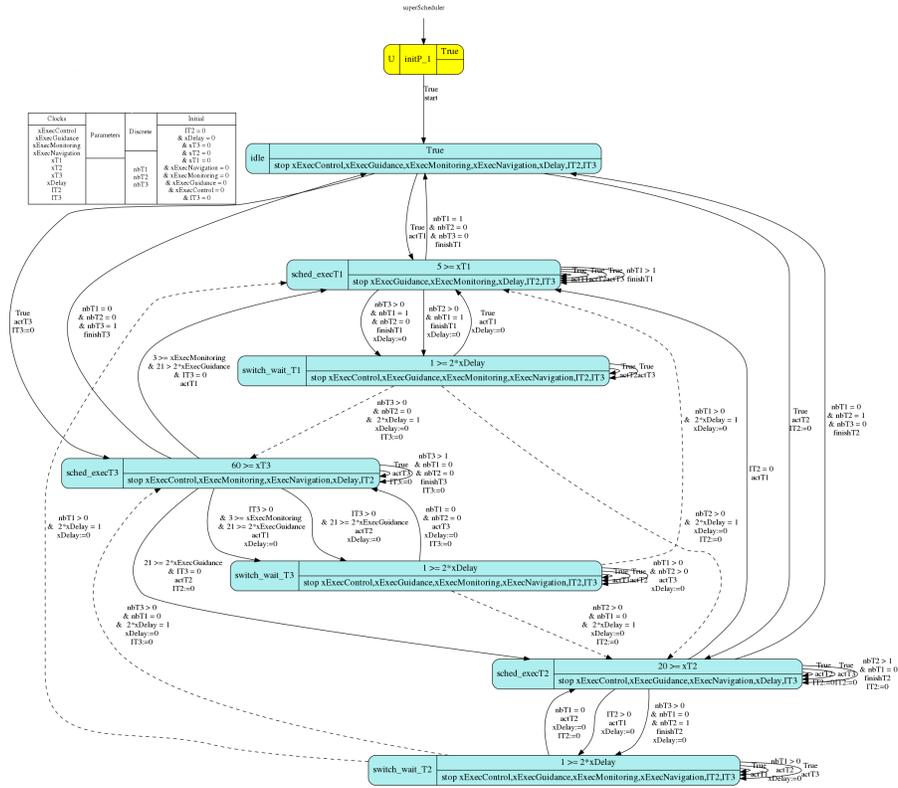


Figure 17: Encoding the FPS scheduler with switches (full version)

B Parametric analyses without reactivities

B.1 Parametric offsets

The constraint synthesized by IMITATOR for the model with parametric offsets is given in Fig. 18.

In order to exemplify admissible values, we exhibit in Table 11 some valuations satisfying the constraint in Fig. 18.

B.2 Parametric deadlines

We exhibit in Table 12 some valuations satisfying the constraint given in Section 8.2.3.

$5 \geq \text{offsetT2}$ $\wedge \text{offsetT3} + 5 > \text{offsetT2}$ $\wedge \text{offsetT3} \geq 0$ $\wedge \text{offsetT2} \geq 0$ $\wedge 1 \geq \text{offsetT3}$ $\wedge \text{offsetT1} = 0$ OR $\text{offsetT1} \geq 0$ $\wedge 11 \geq \text{offsetT3}$ $\wedge \text{offsetT3} > 1 + \text{offsetT1}$ $\wedge 4 \geq \text{offsetT1}$ $\wedge \text{offsetT2} = 0$ OR $\text{offsetT3} > 1$ $\wedge 11 \geq \text{offsetT3}$ $\wedge \text{offsetT2} > 0$ $\wedge 1 \geq \text{offsetT2}$ $\wedge \text{offsetT1} = 0$	OR $\text{offsetT1} > 0$ $\wedge \text{offsetT2} \geq 0$ $\wedge 11 \geq \text{offsetT2}$ $\wedge 4 \geq \text{offsetT1}$ $\wedge \text{offsetT3} = 0$ OR $11 \geq \text{offsetT2}$ $\wedge \text{offsetT3} \geq 0$ $\wedge \text{offsetT2} > 9$ $\wedge 1 \geq \text{offsetT3}$ $\wedge \text{offsetT1} = 0$ OR $\text{offsetT1} + 1 \geq \text{offsetT3}$ $\wedge \text{offsetT1} > 0$ $\wedge \text{offsetT3} > 0$ $\wedge 4 \geq \text{offsetT1}$ $\wedge \text{offsetT2} = 0$	OR $\text{offsetT2} > 5$ $\wedge 9 \geq \text{offsetT2}$ $\wedge \text{offsetT3} > 0$ $\wedge 1 \geq \text{offsetT3}$ $\wedge \text{offsetT1} = 0$ OR $\text{offsetT2} \geq 5$ $\wedge 9 \geq \text{offsetT2}$ $\wedge \text{offsetT1} = 0$ $\wedge \text{offsetT3} = 0$
--	--	--

Figure 18: Parametric offsets

Table 11: Possible offset valuations (without reactivities)

Valuation	offsetT1	offsetT2	offsetT3	$\models K$
v_1	0	2	1	✓
v_2	4	0	11	✓
v_3	2	11	0	✓
v_4	0	9	0	✓
v_5	2	12	1	✗
v_6	5	9	0	✗

B.3 Parametric offsets and deadlines

The constraint synthesized by IMITATOR for the model with both parametric offsets and parametric deadlines is given in Fig. 19.

C Parametric analyses with reactivities

C.1 Parametric offsets

The constraint synthesized by IMITATOR for the model with the 3 reactivities and parametric offsets is given in Fig. 20.

Table 12: Possible deadline valuations (without reactivities)

Valuation	deadlineT1	deadlineT2	deadlineT3	$\models K$
v_1	5	20	60	✓
v_2	4	11	60	✓
v_3	5	15	60	✓
v_4	4	20	60	✓
v_5	3	11	60	✗
v_6	4	9	55	✗

<pre> deadlineT2 > 11 ^ 11 ≥ offsetT3 ^ deadlineT1 ≥ 4 ^ offsetT3 > offsetT2 ^ 20 ≥ deadlineT2 ^ offsetT2 > 0 ^ 5 > deadlineT1 ^ 1 ≥ offsetT2 ^ offsetT1 = 0 ^ deadlineT3 = 60 OR offsetT1 > 0 ^ deadlineT1 ≥ 4 ^ 20 ≥ deadlineT2 ^ offsetT3 > 5 ^ deadlineT2 > 15 ^ 11 ≥ offsetT3 ^ 5 ≥ deadlineT1 ^ deadlineT1 ≥ 1 + offsetT1 ^ offsetT2 = 0 ^ deadlineT3 = 60 OR offsetT1 > 0 ^ offsetT1 + 1 ≥ offsetT3 ^ deadlineT2 ≥ 15 ^ 20 ≥ deadlineT2 ^ 4 ≥ offsetT1 ^ offsetT3 > 0 ^ deadlineT1 = 5 ^ offsetT2 = 0 ^ deadlineT3 = 60 OR deadlineT2 > 11 ^ 11 ≥ offsetT3 ^ deadlineT1 ≥ 4 ^ offsetT3 > deadlineT1 ^ 20 ≥ deadlineT2 ^ 5 ≥ deadlineT1 ^ offsetT2 = 0 ^ offsetT1 = 0 ^ deadlineT3 = 60 </pre>	<pre> OR deadlineT2 > 11 ^ offsetT3 > 0 ^ 20 ≥ deadlineT2 ^ 5 ≥ offsetT3 ^ offsetT2 = 0 ^ deadlineT1 = 5 ^ offsetT1 = 0 ^ deadlineT3 = 60 OR 20 ≥ deadlineT2 ^ deadlineT1 ≥ 4 ^ offsetT1 > 0 ^ deadlineT1 ≥ 1 + offsetT1 ^ deadlineT2 > 11 ^ offsetT3 > 1 + offsetT1 ^ 5 ≥ deadlineT1 ^ 5 ≥ offsetT3 ^ offsetT2 = 0 ^ deadlineT3 = 60 OR offsetT1 >= 0 ^ deadlineT2 > 11 ^ offsetT1 + 1 ≥ offsetT3 ^ deadlineT1 ≥ 4 ^ deadlineT2 ≥ 11 + offsetT1 ^ 20 ≥ deadlineT2 ^ offsetT3 ≥ 0 ^ 5 > deadlineT1 ^ deadlineT1 ≥ 1 + offsetT1 ^ offsetT2 = 0 ^ deadlineT3 = 60 OR offsetT2 > 5 ^ 20 ≥ deadlineT2 ^ offsetT1 > 0 ^ deadlineT1 > 4 ^ deadlineT2 ≥ 10 + offsetT2 ^ 5 ≥ deadlineT1 ^ deadlineT1 ≥ 1 + offsetT1 ^ deadlineT2 ≥ 19 ^ offsetT3 = 0 ^ deadlineT3 = 60 </pre>	<pre> OR deadlineT2 > 11 ^ 1 ≥ offsetT3 ^ deadlineT2 ≥ 10 + offsetT2 ^ offsetT2 ≥ 1 ^ 20 ≥ deadlineT2 ^ deadlineT1 ≥ offsetT2 ^ 5 > deadlineT1 ^ offsetT3 ≥ 0 ^ deadlineT1 ≥ 4 ^ offsetT1 = 0 ^ deadlineT3 = 60 OR deadlineT2 > 11 ^ 19 > deadlineT2 ^ offsetT2 ≥ 1 ^ 5 > offsetT2 ^ offsetT3 > 0 ^ deadlineT2 ≥ 10 + offsetT2 ^ 1 ≥ offsetT3 ^ offsetT1 = 0 ^ deadlineT1 = 5 ^ deadlineT3 = 60 OR deadlineT2 > 11 ^ deadlineT1 ≥ 4 ^ 5 ≥ deadlineT1 ^ offsetT3 > 0 ^ 1 > offsetT2 ^ 20 ≥ deadlineT2 ^ offsetT2 ≥ offsetT3 ^ offsetT1 = 0 ^ deadlineT3 = 60 OR 19 > deadlineT2 ^ offsetT3 ≥ 0 ^ offsetT2 > deadlineT1 ^ deadlineT2 ≥ 10 + offsetT2 ^ 5 > offsetT2 ^ 1 > offsetT3 ^ deadlineT1 ≥ 4 ^ offsetT1 = 0 ^ deadlineT3 = 60 </pre>
--	--	--

Figure 19: Parametric offsets and deadlines

<pre> 11 >= offsetT2 ^ offsetT3 >= 0 ^ offsetT2 >= offsetT3 ^ 1 >= offsetT3 ^ offsetT1 = 0 OR offsetT3 > offsetT2 ^ 1 >= offsetT2 ^ offsetT2 >= 0 ^ 11 >= offsetT3 ^ offsetT1 = 0 </pre>	<pre> OR offsetT2 >= 0 ^ offsetT1 > 0 ^ 11 >= offsetT2 ^ 4 >= offsetT1 ^ offsetT3 = 0 OR offsetT1 > 0 ^ 11 >= offsetT3 ^ offsetT3 > 0 ^ 4 >= offsetT1 ^ offsetT2 = 0 </pre>
---	---

Figure 20: Parametric offsets for model with the 3 reactivities

C.2 Parametric deadlines

The constraint synthesized by IMITATOR for the model with the 3 reactivities and parametric deadlines in Fig. 21.

$$\begin{aligned}
 & \text{deadlineT2} \geq 11 \\
 & \wedge \text{deadlineT1} \geq 4 \\
 & \wedge 5 \geq \text{deadlineT1} \\
 & \wedge 20 \geq \text{deadlineT2} \\
 & \wedge \text{deadlineT3} = 60
 \end{aligned}$$

Figure 21: Parametric deadlines for model with the 3 reactivities

D Parametric analyses with switch time without reactivities

D.1 Parametric offsets

The constraint synthesized by IMITATOR for the model with parametric offsets and the context switch time is given in Fig. 22.

$$\begin{aligned}
 & \text{offsetT3} \geq 10 \\
 & \wedge 7 \geq 2 * \text{offsetT1} \\
 & \wedge 2 * \text{offsetT1} > 5 + 2 * \text{offsetT2} \\
 & \wedge 23 \geq 2 * \text{offsetT3} \\
 & \wedge \text{offsetT1} \geq 3 \\
 & \wedge 2 * \text{offsetT2} + 7 > 2 * \text{offsetT1} \\
 & \wedge \text{offsetT2} \geq 0 \\
 & \wedge \text{offsetT1} > 2 + \text{offsetT2} \\
 & \text{OR} \\
 & \text{offsetT2} + 2 * \text{offsetT3} + 3 > 3 * \text{offsetT1} \\
 & \wedge \text{offsetT2} \geq 0 \\
 & \wedge 2 * \text{offsetT3} > 1 + 2 * \text{offsetT1} \\
 & \wedge 19 \geq 2 * \text{offsetT3} \\
 & \wedge 2 * \text{offsetT2} + 7 > 2 * \text{offsetT1} \\
 & \wedge \text{offsetT1} \geq 3 + \text{offsetT2} \\
 & \wedge 7 \geq 2 * \text{offsetT1} \\
 & \text{OR} \\
 & \text{offsetT3} > 3 \\
 & \wedge \text{offsetT3} \geq 3 + \text{offsetT2} \\
 & \wedge 2 * \text{offsetT3} > 5 + 2 * \text{offsetT2} \\
 & \wedge \text{offsetT1} + \text{offsetT3} > 6 + 2 * \text{offsetT2} \\
 & \wedge \text{offsetT1} \geq 3 + \text{offsetT2} \\
 & \wedge \text{offsetT2} \geq 0 \\
 & \wedge 2 * \text{offsetT1} + 1 \geq 2 * \text{offsetT3} \\
 & \wedge 7 \geq 2 * \text{offsetT1}
 \end{aligned}$$

$$\begin{aligned}
 & \text{OR} \\
 & 19 \geq 2 * \text{offsetT3} \\
 & \wedge \text{offsetT1} \geq 0 \\
 & \wedge \text{offsetT3} > 5 \\
 & \wedge \text{offsetT2} > 1 + \text{offsetT1} \\
 & \wedge 2 > \text{offsetT2} \\
 & \text{OR} \\
 & \text{offsetT1} \geq 0 \\
 & \wedge \text{offsetT2} > 0 \\
 & \wedge \text{offsetT3} > 5 + \text{offsetT2} \\
 & \wedge \text{offsetT2} \geq \text{offsetT1} \\
 & \wedge \text{offsetT1} + 1 \geq \text{offsetT2} \\
 & \wedge 1 > 2 * \text{offsetT1} \\
 & \wedge 19 \geq 2 * \text{offsetT3} \\
 & \text{OR} \\
 & \text{offsetT2} \geq 5 \\
 & \wedge 23 \geq 2 * \text{offsetT2} \\
 & \wedge \text{offsetT1} \geq 0 \\
 & \wedge \text{offsetT3} > 1 + \text{offsetT1} \\
 & \wedge 3 \geq 2 * \text{offsetT3}
 \end{aligned}$$

$$\begin{aligned}
 & \text{OR} \\
 & 23 \geq 2 * \text{offsetT2} \\
 & \wedge \text{offsetT3} \geq 0 \\
 & \wedge \text{offsetT2} \geq 5 \\
 & \wedge 1 \geq \text{offsetT3} \\
 & \wedge \text{offsetT1} = 0 \\
 & \text{OR} \\
 & 2 * \text{offsetT2} > 7 \\
 & \wedge 3 * \text{offsetT2} > 4 \\
 & \wedge \text{offsetT2} + 8 > 0 \\
 & \wedge 10 > \text{offsetT2} \\
 & \wedge 2 * \text{offsetT1} = 7 \\
 & \wedge \text{offsetT3} = 0
 \end{aligned}$$

Figure 22: Parametric offsets for model with switch time

D.2 Parametric deadlines

The constraint synthesized by IMITATOR for the model with parametric deadlines and the context switch time is given in Fig. 23.

$$\begin{array}{l} 2 * \text{deadlineT2} \geq 9 \\ \wedge 2 * \text{deadlineT1} \geq 9 \\ \wedge 5 \geq \text{deadlineT1} \\ \wedge 20 \geq \text{deadlineT2} \\ \wedge \text{deadlineT3} = 60 \end{array}$$

Figure 23: Parametric deadlines for model with switch time

E Parametric analyses with reactivities and with switch time

E.1 Parametric offsets

The constraint synthesized by IMITATOR for the model with parametric offsets, reactivities constraints the context switch time is given in [Fig. 24](#).

E.2 Parametric deadlines

The constraint synthesized by IMITATOR for the model with parametric deadlines, reactivities constraints and the context switch time is given in [Fig. 25](#).

<pre> offsetT3 >= 10 ^ 7 >= 2 * offsetT1 ^ 2 * offsetT1 > 5 + 2 * offsetT2 ^ 23 >= 2 * offsetT3 ^ offsetT1 >= 3 ^ 2 * offsetT2 + 7 > 2 * offsetT1 ^ offsetT2 >= 0 ^ offsetT1 > 2 + offsetT2 OR offsetT2 + 2 * offsetT3 + 3 > 3 * offsetT1 ^ offsetT2 >= 0 ^ 2 * offsetT3 > 1 + 2 * offsetT1 ^ 19 >= 2 * offsetT3 ^ 2 * offsetT2 + 7 > 2 * offsetT1 ^ offsetT1 >= 3 + offsetT2 ^ 7 >= 2 * offsetT1 OR offsetT2 >= 5 ^ offsetT3 >= 0 ^ 23 >= 2 * offsetT2 ^ 1 >= offsetT3 ^ offsetT1 = 0 OR offsetT1 >= 0 ^ offsetT2 > 0 ^ offsetT3 > 5 + offsetT2 ^ offsetT2 >= offsetT1 ^ offsetT1 + 1 >= offsetT2 ^ 1 > 2 * offsetT1 ^ 19 >= 2 * offsetT3 OR 19 >= 2 * offsetT3 ^ offsetT1 >= 0 ^ offsetT3 > 5 ^ offsetT2 > 1 + offsetT1 ^ 2 > offsetT2 OR 23 >= 2 * offsetT2 ^ offsetT2 >= 5 ^ offsetT1 >= 0 ^ offsetT3 > 1 + offsetT1 ^ 3 >= 2 * offsetT3 OR 2 * offsetT3 + 1 > 2 * offsetT1 + 2 * offsetT2 ^ offsetT3 >= 3 + offsetT2 ^ 2 * offsetT3 > 5 + 2 * offsetT2 ^ offsetT1 + offsetT3 > 6 + 2 * offsetT2 ^ offsetT1 >= 3 + offsetT2 ^ 2 * offsetT1 + 1 >= 2 * offsetT3 ^ 7 >= 2 * offsetT1 ^ offsetT2 >= 0 OR 2 * offsetT1 > 7 ^ offsetT2 >= 0 ^ offsetT3 > 5 ^ offsetT1 > 3 + offsetT2 ^ 19 >= 2 * offsetT3 ^ 4 >= offsetT1 OR offsetT2 >= 0 ^ offsetT1 >= 3 + offsetT2 ^ offsetT3 >= offsetT2 ^ 4 * offsetT2 + 7 > 2 * offsetT1 ^ offsetT2 + 3 > offsetT3 ^ 2 * offsetT2 + 7 >= 2 * offsetT1 ^ 1 >= 2 * offsetT2 OR 2 * offsetT2 > 7 ^ 68 * offsetT2 > 13 ^ 6 * offsetT2 + 43 > 0 ^ 5 * offsetT2 + 52 > 0 ^ 10 > offsetT2 ^ 2 * offsetT1 = 7 ^ offsetT3 = 0 </pre>	<pre> OR offsetT3 > 11 ^ offsetT1 + 1 >= offsetT2 ^ offsetT1 >= 0 ^ 1 > 2 * offsetT1 ^ offsetT2 > 0 ^ offsetT2 >= offsetT1 ^ 23 >= 2 * offsetT3 OR 1 > 2 * offsetT1 ^ offsetT3 > 5 ^ offsetT1 + 1 >= offsetT2 ^ offsetT2 > offsetT1 ^ 2 * offsetT2 > 1 + 2 * offsetT1 ^ offsetT2 + 5 >= offsetT3 ^ offsetT1 >= 0 OR offsetT1 >= 0 ^ 2 * offsetT2 + 19 >= 2 * offsetT3 ^ 6 * offsetT2 + 35 > 4 * offsetT3 ^ offsetT3 >= 10 ^ 2 * offsetT2 >= 3 ^ 2 > offsetT2 ^ offsetT2 > 1 + offsetT1 OR 5 >= offsetT3 ^ 9 > offsetT1 + offsetT3 ^ 5 > offsetT1 ^ offsetT3 > offsetT1 ^ 2 * offsetT3 > 1 + 2 * offsetT1 ^ 4 >= offsetT1 ^ 2 * offsetT1 >= 7 + 2 * offsetT2 ^ offsetT2 >= 0 OR 3 >= 2 * offsetT2 ^ offsetT3 >= 3 + offsetT2 ^ offsetT2 > 0 ^ 5 > offsetT3 ^ offsetT1 = 0 OR offsetT2 + 3 >= offsetT3 ^ 2 * offsetT1 + offsetT2 + 3 > offsetT3 ^ offsetT3 >= offsetT2 ^ offsetT1 >= 0 ^ offsetT2 > 1 + offsetT1 ^ 3 >= 2 * offsetT2 OR 4 * offsetT2 + 5 > 2 * offsetT1 + 2 * offsetT3 ^ 2 * offsetT1 + 2 * offsetT2 + 2 > offsetT3 ^ 2 * offsetT2 + 2 > offsetT3 ^ 4 * offsetT1 + 2 * offsetT2 > 3 ^ 2 * offsetT2 >= 3 ^ offsetT1 >= 0 ^ offsetT3 > 3 + offsetT2 ^ offsetT2 > 1 + offsetT1 ^ 5 >= offsetT3 OR offsetT3 >= 10 ^ 2 * offsetT1 >= 7 + 2 * offsetT2 ^ 23 >= 2 * offsetT3 ^ 4 >= offsetT1 ^ offsetT2 >= 0 OR offsetT3 >= 10 ^ offsetT2 >= 1 + offsetT1 ^ 2 > offsetT1 + offsetT2 ^ 3 > 2 * offsetT2 ^ 2 * offsetT2 + 19 >= 2 * offsetT3 ^ offsetT1 >= 0 </pre>
--	---

Figure 24: Parametric offsets for model with reactivities and with switch time

$$\begin{aligned} & 2 * \text{deadlineT2} \geq 9 \\ & \wedge 2 * \text{deadlineT1} \geq 9 \\ & \wedge 5 \geq \text{deadlineT1} \\ & \wedge 20 \geq \text{deadlineT2} \\ & \wedge \text{deadlineT3} = 60 \end{aligned}$$

Figure 25: Parametric deadlines for model with reactivities and with switch time