Scheduling with IMITATOR: Some Case Studies

Romain Soulat

March 2012

Research report LSV-12-05

Laboratoire Spécification & Vérification

École Normale Supérieure de Cachan
61, avenue du Président Wilson
94235 Cachan Cedex France

Scheduling with IMITATOR: Some Case Studies

Romain Soulat

Abstract

The tool IMITATOR implements the *Inverse Method* (*IM*) for Timed Automata (TAs). Given a TA $\mathcal{A}$ and a tuple π_0 of reference valuations for timings, *IM* synthesizes a constraint around π_0 where $\mathcal{A}$ behaves in the same discrete manner. This provides us with a quantitative measure of robustness of the behavior of $\mathcal{A}$ around π_0. The new version IMITATOR 2.5 integrates the new features of stopwatches (in addition to standard clocks) and updates (in addition to standard clock resets), as well as powerful algorithmic improvements for state space reduction. We illustrate on several case studies of preemptive scheduling problems how such features make the tool well-suited to analyze robustness.

1 Introduction

IMITATOR 2.5 (for *Inverse Method for Inferring Time AbstracT behaviOR*) is a tool for parameter synthesis in the framework of real-time systems based on the inverse method *IM* for Parametric Timed Automata (PTAs). Different from CEGAR-based methods, this algorithm for parameter synthesis makes use of a "good" parameter valuation π_0 instead of a set of "bad" states [2]. IMITATOR takes as input a network of PTAs with stopwatches and a reference valuation π_0; it synthesizes a constraint K on the parameters such that (1) $\pi_0 \models K$ and (2) for all parameter valuation π satisfying K, the trace set (i.e., the discrete behavior) of $\mathcal{A}$ under π is the same as for $\mathcal{A}$ under π_0. This provides the system with a criterion of *robustness* (see, e.g., [10]) around π_0.

Figure 1: Functional view of IMITATOR

We show in this report how to exploit solutions of scheduling problems for generating zones of robustness around these solutions. We represent the scheduling problems under the form of networks of Timed Automata (TAs). As we consider scheduling with preemption, TAs are enriched with stopwatches and arbitrary updates [1]. Each schedule (interleaving of tasks) corresponds to a

branch in the reachability tree of the TA. A typical solution (schedule) corresponds to the shortest path of this tree. Roughly speaking, using the given solution of a scheduling problem, we explain how to synthesize a robustness region around this solution inside which the interleaving of the given solution still corresponds to the shortest path. In order to do this, we use a parametric version of the original TA and apply IMITATOR.

The plan of this paper is as follows. In section 2, we present on an example the sketch of our methodology that we will apply on different classes of examples taken from the literature. We apply our method to problems of

- cyclic tasks with various scheduling policies (Section 3)
- sporadic tasks with variable execution times (Section 4)
- cyclic tasks with fixed priorities and given task deadlines (Section 5)
- job-shop scheduling (Section 6)

In section 7, we explain how to use the Behavioral Cartography (BC) method [3]. Experimental results are given in Section 8. Final remarks are given in Section 9.

2 Sketch of the Method

We will now illustrate our method on a preemptive jobshop example given in [1]. The jobshop scheduling problem is a generic resource allocation problem in which common resources ("machines") are required at various time points (and for given duration) by different tasks. For instance, one needs to use a machine m_1 for d_1 time units, machine m_2 for d_2 time units, and so on. The goal is to find a way ("schedule") to allocate the resources such that all tasks terminate as early as possible ("minimal makespan"). Let us consider the jobshop problem $\{J_1, J_2\}$ for 2 jobs and 3 machines with: $J_1 = (m_1, d_1), (m_2, d_2), (m_3, d_3)$ and $J_2 = (m_2, d'_2)$ with $d_1 = 3, d_2 = 2, d_3 = 4, d'_2 = 5$. There are many possible schedules (two of them are depicted in Figure 2). In [1], this problem is modeled as a product $\mathcal{A}$ of TAs with stopwatches, each TA modeling a job. Each schedule corresponds to a branch in the reachability tree of $\mathcal{A}$. The makespan value corresponds to the duration of the shortest branch, here 9.

Let us explain how to analyze the robustness of the valuation $\pi_0 : \{d_2 = 2, d'_2 = 5\}$ with respect to the makespan value 9. We first consider a parametric version of $\mathcal{A}$ where d_2 and d'_2 become parameters. In the same spirit as in [6], we add an observer $\mathcal{O}$, which is a TA synchronized with $\mathcal{A}$, that fires a transition labeled *DEADLINE* as soon as a schedule spends more than 9 time units. We then use IMITATOR (instead of a CEGAR-like method as in [6]) with $\mathcal{A} \parallel \mathcal{O}$ as a model input and π_0 as a valuation input. This yields the constraint K: $7 > d'_2 \wedge 3 > d_2 \wedge d'_2 + d_2 \geq 7$. See Figure 2 for a geometrical representation of K.

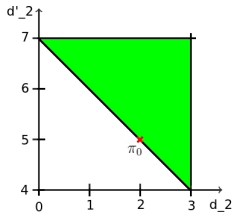

Figure 2: Geometrical representation of K around $\pi_0 : (2, 5)$.

By the *IM* principle, the set of traces (i.e., discrete runs) of $\mathcal{A} \parallel \mathcal{O}$ is always the same, for any point (d_2, d'_2) of K. This set of traces is depicted under the form of a tree in Figure 3. Since the makespan for π_0 is 9, we know that some

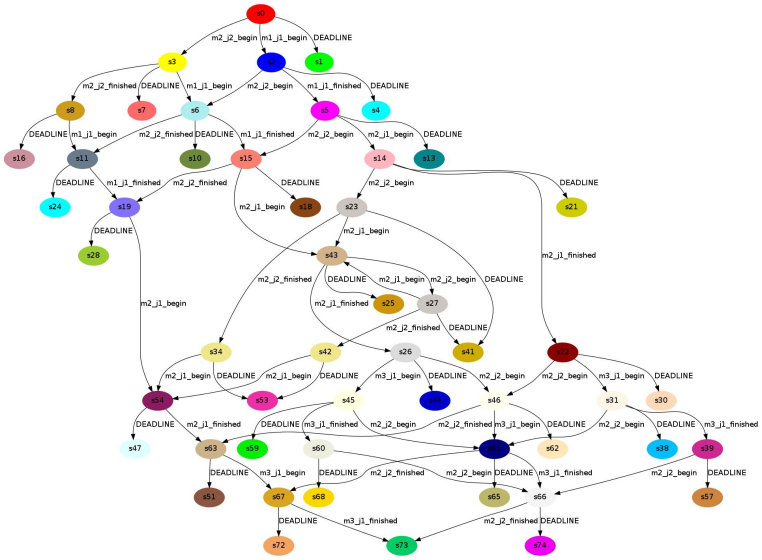

Figure 3: Graphics of the trace set output by IMITATOR on the jobshop example

branches of the tree do not contain any *DEADLINE* label (these branches end at node $s73$ in Figure 3). This holds for each point (d_2, d'_2) of K. The makespan of the system is thus always at most 9 in K. (In particular, we can increase d_2 from 2 to 3, or increase d'_2 from 5 to 7 while keeping the makespan less than or equal to 9.)

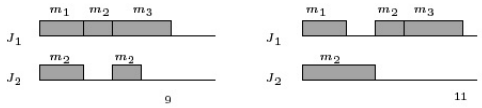

Figure 4: Schedules

3 Cyclic Tasks with Various Scheduling Policies

3.1 Description of the Problem

In this framework of case studies, we want to prove the schedulability of cyclic tasks on a single machine, we consider the preemptive framework, where a currently running task can be interrupted by another one when it becomes activated and has a higher priority. Each task τ_i is defined by an offset $O_i \in \mathbb{N}$ (the time before the first release), a period $T_i \in \mathbb{N}$ (the amount of time between two releases), a deadline $D_i \in \mathbb{N}$ (the maximum timed allowed to perform the task) and a Worst Case Execution Time $C_i \in \mathbb{N}$ (the maximum of work required to execute the task).

Added to these automata, we have an a priori scheduling policy either Fixed Priority (FP) or Earliest Deadline First (EDF). The first policy gives to each task τ_i a priority $\mathcal{P}_i$ and when two tasks are activated at the same time, the computational time is given to the highest priority. The EDF policy gives the computational time to the task that has its deadline coming the sooner.

In this context, given a scheduler, the system is said to be *schedulable* if each task τ_i is completed before D_i time units after the beginning of its period. Actually because of the periodicity of the problem, we only have to ensure that it is schedulable within $lcm_{i\in\mathcal{I}}(T_i)$.

This problem is:
Given a list of tasks $\{\tau_i\}_{i\in\mathcal{I}}$

$$\tau_i = (O_i, T_i, D_i, C_i),$$

is the system schedulable?
Formally, we have:

- Model M: A TA per task and a TA for the scheduler
- Input I: $\{O_i, T_i, D_i, C_i\}_{i\in I}$
- Output: Yes/No

3.2 Robustness Analysis

The problem is parameterized typically by letting deadlines D_is instantiated and parameterizing some of the C_is, O_is and T_is. We have:

- Model M': a set of PTA deduced from M by parameterizing the values of interest

- Input: I and an identified set of parameters
- Output found by IMITATOR: a constraint K such that for every $\pi \models K$, the π-instantiated model $M'[\pi]$ is schedulable.

4 Scheduling with Variable Execution Times [11]

4.1 Description of the Problem

The problem addressed here is how to determine what is the minimum time such that every job in n independent job chains, denoted $J_1, J_2, \cdots, J_n$ can complete in time when the jobs are scheduled on a processor according to a priority-driven algorithm. Roughly speaking, we let $J_{i,j}$ denote the jth job of the chain J_i. Each job J_{i_j} has a fixed priority $\Phi_{i,j}$ and is preemptable. The execution time of $J_{i,j}$ is in the range $[e^-_{i,j}, e^+_{i,j}]$ with $e^-_{i,j}, e^+_{i,j}$ in $\mathbb{N}$. The release time $r_{i,j}$ of job $J_{i,j}$ is set to an integer value. $J_{i,1}$ is ready for execution at its release time $r_{i,1}$; for each $j > 1$, $J_{i,j}$ cannot execute until its immediate predecessor $J_{i,j-1}$ completes. The problem and two schedules are depicted in Figure 5. For details, see [11].

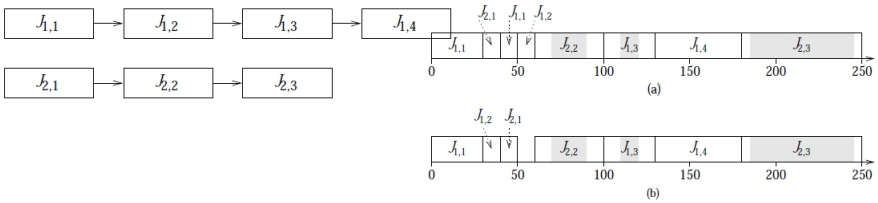

Figure 5: Two task chains (left) and two acceptable schedules (right)

The problem is as follows:
Given a list of chain of tasks $\{J_i\}_{i\in\mathcal{I}}$ where $J_i = J_{i,j\,j\in\mathcal{J}(i)}$ and

$$J_{i,j} = (r_{i,j}, e^-_{i,j}, e^+_{i,j}, \Phi_{i,j}),$$

what is the maximum time needed to complete all the tasks?
Formally, we have:

- Model: A product of TAs, with a TA per task and a TA for the FP scheduler
- Input: $I = (r_{i,j}, e^-_{i,j}, e^+_{i,j}, \Phi_{i,j})_{i\in\mathcal{I}}$
- Output: Δ = Worst case completion time of the last task (of the last chain).

4.2 Robustness Analysis

The problem is parameterized typically by letting the priorities $\Phi_{i,j}$ and the $[e^-_{i,j}, e^+_{i,j}]$ instantiated as well as the computed value Δ, and by parameterizing some of the r_is. We have:

- Model M': a PTA deduced from M by parameterizing the values of interest and composed with a TA that tests wether or not the last task is completed within Δ (using an error state)
- Input: I together with Δ and an identified set of parameters
- Output found by IMITATOR: a constraint K such that for every $\pi \models K$, the last task of the π-instantiated model $M'[\pi]$ is completed within Δ.

5 Cyclic Tasks with Given Tasks Deadlines [6, 8]

5.1 Description of the Problem

We are given a set of tasks $\{\tau_1, \ldots, \tau_n\}$. Each task τ_i is periodic of period T_i (a fixed duration of time between two activation events) , and an offset O_i for its first activation time. Once a task τ_i has been activated, it executes for at most time C_i and has to terminate within the deadline D_i.

We say that the system is *schedulable* if each task τ_i is completed before its relative deadline D_i. Actually, because of the periodicity of the system, we only have to be sure that it is schedulable within $lcm_{i\in\mathcal{I}}(T_i)$.

The problem is as follows:
Given a list of tasks $\{\tau_i\}_{i\in\mathcal{I}}$ with

$$\tau_i = (O_i, T_i, D_i, C_i),$$

is the system schedulable?
Formally, we have:

- Model M: A TA per task and a TA for the scheduler
- Input I: $\{O_i, T_i, D_i, C_i\}_{i\in\mathcal{I}}$
- Output: Yes or No

5.2 Robustness Analysis

The problem is parameterized typically by letting the deadlines D_is instantiated and by parameterizing some of the C_is, O_is and T_is. Formally, we have:

- Model M': a PTA deduced from M by parameterizing the values of interest
- Input: I and an identified set of parameters
- Output found by IMITATOR: a constraint K such that for every $\pi \models K$, the π-instantiated model $M'[\pi]$ is schedulable.

5.3 Comparison with [6]

We consider the case of two periodic tasks $\{\tau_1, \tau_2\}$ with $D_1 = 7, T_1 = 10, O_1 = 0, C_1 = 3, D_2 = 6, T_2 = 10, O_2 = 3, C_2 = 5$. We parameterize C_1, C_2 and O_2. Applying IMITATOR, we found the following constraint K:

$$
\begin{aligned}
& 6 \geq C_2 \\
\wedge\ & 3 \geq C_1 \\
\wedge\ & 6 \times C_1 > 17 \\
\wedge\ & 2 \times C_1 + C_2 > 6 + O_2 \\
\wedge\ & O_2 \geq C_1 \\
\wedge\ & 10 \geq O_2 + C_2
\end{aligned}
$$

In [6], the authors uses a CEGAR-based method to synthesize a constraint on the parameters that will guarantee that the system is schedulable.

The constraint found by their method is:

$$
\begin{aligned}
& C_1 + C_2 < 6 + O_2 \\
\wedge\ & C_1 + C_2 < 10 \\
\wedge\ & C_1 + C_2 > 6 \\
\wedge\ & C_2 < 10 - O_2 \\
\wedge\ & C_1 < 7 \\
\wedge\ & C_2 < 6
\end{aligned}
$$

We notice that this constraint is uncomparable with the constraint K found by IMITATOR.

6 Job-Shop Scheduling [1]

6.1 Description of the Problem

The Job-shop scheduling problem is a generic resource allocation problem in which common resources ("machines") are required at various time points (and for given duration) by different tasks. The goal is to find a way to allocate the resources such that all the tasks terminate as soon as possible (or "minimal makespan" in the scheduling jargon). We consider a fixed set M of resources. A *step* is a pair (m, d) where $m \in M$ and $d \in \mathbb{N}$, indicating the required utilization of resource m for time duration d. A *job specification* is a finite sequence

$$J = (m_1, d_1), (m_2, d_2), \cdots, (m_k, d_k)$$

of steps stating that in order to accomplish job J, one needs to use a machine m_1 for d_1 time, then use machine m_2 for d_2 time etc. For details, see [1].The problem is:
Given a list of job specifications $\{J_i\}_{i\in\mathcal{I}}$ where

$$J_i = \{(m_{i,j}, d_{i,j})\}_{j\in J(i)},$$

what is the minimal makespan to complete all the tasks?
Formally, we have:

- Model M: A product of TAs, with a TA per job
- Input I: $\{(m_{i,j}, d_{i,j})_{i\in I(j)}\}_{j\in J}$
- Output: μ = minimal makespan

6.2 Robustness Analysis

The problem is parameterized typically by parameterizing some of the d_is. Formally, we have:

- Model M': a PTA deduced from M by parameterizing the values of interest and composed with a TA that tests wether or not the last task is completed within μ (using an ok and an error state)
- Input: I together with an identified set of parameters and the value of the computed makespan μ
- Output found by IMITATOR: a constraint K such that for every $\pi \models K$, there is a schedule which allows to complete the last task of the π-instantiated model $M'[\pi]$ within μ.

7 Schedulability with Behavioral Cartography

When one is only interested in finding all the parameters valuation such that the system is schedulable, without considering a particular schedule, it is interesting to consider the Behavioral Cartography Method included in IMITATOR. By iterating IM over all the integers points inside a finite but dense rectangle V_0 in the parametric space, one is able to decompose (most of) the parametric space included into V_0 into behavioral tiles[1] [3].

For a given tile, we ensure that for every valuation of the parameters in this tile, the behavior of $\mathcal{A}$ is the same; therefore, we only have to test the schedulability of a single point for each tile to ensure the schedulability on the whole tile.

[1]Actually, one can take a finer discretization step than integers to ensure a better coverability of V_0

Let us apply this method on a "Rate monotonic" example of [5, Section III]. There are three periodic tasks τ_1, τ_2 and τ_3 with periods of $T_1 = 3$, $T_2 = 8$ and $T_3 = 20$ and deadlines of $D_1 = 3$, $D_2 = 8$ and $D_3 = 20$. We are interested in finding the set of computation times of each task such that the system is schedulable. (The interested reader can find the full details in [5].)

We set V_0 as $C_1 \in [0, 3]$, $C_2 \in [0, 8]$ and $C_3 \in [0, 20]$, where C_i is the computational time of τ_i. The system is unschedulable if there exists a task τ_i such that $C_i > T_i$. Algorithm *BC* outputs the set of tiles and we check for one point of each tile whether the system is schedulable or not. The result for this example is given in Figures 6, 7, 8 for this V_0 with a discretization step of 0.2. This corresponds exactly to the schedulability region found in [5] (Figure 1 (a)).

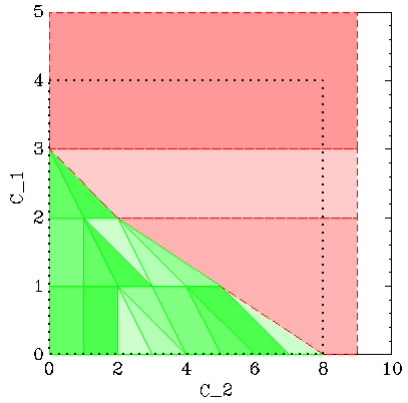

Figure 6: Cartography output by IMITATOR 2.5 (in green the system is schedulable; in red, unschedulable)

8 Experimental Results

All case studies and experiments have been performed on Ubuntu 11.10 equipped with an Intel Core 2 2.93 GiHz processor with 2 GiB RAM). More data (including the output constraints) are given in `http://www.lsv.ens-cachan.fr/Software/imitator/case-studies.php`. For case studies FPi and $EDFi$, see the values of the parameters in Appendix A.

In Figure 9, we give from left to right the name of the case study, the number $|\mathcal{A}|$ of automata in parallel, the number $|X|$ of clocks, the number $|P|$ of parameters, the number $|S|$ of states computed, the number $|T|$ of transitions computed, the number n of iterations of the inverse method, the number $|K|$ of inequalities within the resulting constraint K, the computation time t in seconds.

The examples LA02 2×5 and 3×5 are taken from the LA02 example of `http://bach.istc.kobe-u.ac.jp/csp2sat/jss/`, where we only consider the

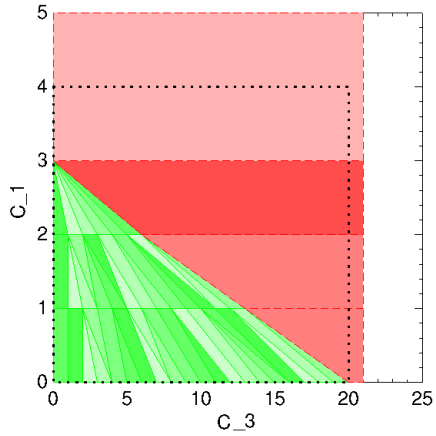

Figure 7: Cartography output by IMITATOR 2.5 (in green the system is schedulable; in red, unschedulable)

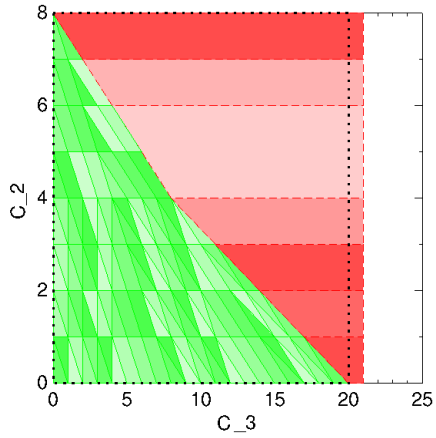

Figure 8: Cartography output by IMITATOR 2.5 (in green the system is schedulable; in red, unschedulable)

2 (resp. 3) first jobs on 5 machines.

9 Final Remarks

The use of models such as PTAs and parametric Time Petri Nets (TPNs) for solving scheduling problems has received attention in the past few years. For example, Roméo [9] performs model checking for parametric TPNs with stopwatches, and synthesizes parameter valuations satisfying TCTL formulæ. An extension of Uppaal allows parametric model checking [4], although the model

Case study	$\|\mathcal{A}\|$	$\|X\|$	$\|P\|$	$\|S\|$	$\|T\|$	n	$\|K\|$	t (s)
Examples from section 3								
FP 0	5	7	10	63	62	32	12	1.10
FP 1	5	7	10	205	225	40	11	15.6
FP 2	5	7	10	208	228	42	12	10.7
FP 3	5	7	10	49	51	15	12	1.03
EDF 0	5	10	13	63	62	32	13	2.42
EDF 1	5	10	13	76	91	31	20	66.1
EDF 2	5	10	13	254	326	47	12	9.9
EDF 3	5	10	13	31	33	9	13	1.09
Examples from section 4								
from [11]	8	15	18	215	264	15	17	85.3
Examples from section 5								
from [6]	4	6	8	676	886	15	15	288.3
from [8]	3	4	9	60	103	10	7	2.1
Examples from section 6								
from [1]	3	3	4	53	70	10	5	0.45
LA02 2×5	3	3	11	371	528	21	10	63.4
LA02 3×5	4	3	16	4903	9043	30	5	160.6
Examples from section 7								
from [5]	5	7	10	_	_	_	_	66

Figure 9: Results for the schedulability problem

itself remains non-parametric. The approach most related to IMITATOR 2.5 is [6, 8], where the authors infer parametric constraints guaranteeing the feasibility of a schedule, using PTAs with stopwatches. The main difference between [6, 8] and IMITATOR relies in our choice of the inverse method, rather than a CEGAR-based method. First results obtained on the same case studies are incomparable (although similar in form), which seems to indicate that the two methods are complementary. The problem of finding the schedulability region was attacked in analytic terms in [5]; the size of our examples is rather modest compared to those treated using such analytic methods. However, in many schedulability problems, no analytic solution exists (see, e.g., [11]), and exhaustive simulation is exponential in the number of jobs. In such cases, symbolic methods as ours and those of [6, 8] are useful to treat critical real-life examples of small size. We are thus involved in a project [7] with an industrial partner with first interesting results.

References

[1] Y. Abdeddaïm and O. Maler. Preemptive job-shop scheduling using stopwatch automata. In *TACAS*, pages 113–126, 2002.

[2] É. André, T. Chatain, E. Encrenaz, and L. Fribourg. An inverse method for parametric timed automata. *International Journal of Foundations of Computer Science*, 20(5):819–836, 2009.

[3] É. André and L. Fribourg. Behavioral cartography of timed automata. In *RP*, volume 6227 of *LNCS*, pages 76–90. Springer, 2010.

[4] G. Behrmann, K.G. Larsen, and J.I.Rasmussen. Beyond liveness: Efficient parameter synthesis for time bounded liveness. In *FORMATS*, pages 81–94, 2005.

[5] E. Bini and G.C. Buttazzo. Schedulability analysis of periodic fixed priority systems. *IEEE Trans. Computers*, 53(11):1462–1473, 2004.

[6] A. Cimatti, L. Palopoli, and Y. Ramadian. Symbolic computation of schedulability regions using parametric timed automata. In *RTSS*, pages 80–89, Washington, DC, USA, 2008. IEEE Computer Society.

[7] Laurent Fribourg and David Lesens. Projet ROSCOV: Robuste ordonnancement de systèmes de contrôle de vol. Project report (in French), December 2011. Available at `www.farman.ens-cachan.fr/ROSCOV.pdf`.

[8] T.T.H. Le, L. Palopoli, R. Passerone, Y. Ramadian, and A. Cimatti. Parametric analysis of distributed firm real-time systems: A case study. In *ETFA*, pages 1–8, 2010.

[9] D. Lime, O. H. Roux, C. Seidner, and L.-M. Traonouez. Romeo: A parametric model-checker for Petri nets with stopwatches. In *TACAS*, volume 5505 of *LNCS*, pages 54–57. Springer, 2009.

[10] Nicolas Markey. Robustness in real-time systems. In *SIES*, pages 28–34. IEEE, 2011.

[11] J. Sun, M.K. Gardner, and J.W.S. Liu. Bounding completion times of jobs with arbitrary release times, variable execution times, and resource sharing. *IEEE Trans. Softw. Eng.*, 23:603–615, 1997.

A Parameter values for the FPs and EDFs examples

A.1 FP 0 and EDF 0

Three cyclic tasks of the form $\tau_i = (O_i, T_i, D_i, C_i)$ with:

- $\tau_1 = (0, 10, 10, 4)$
- $\tau_2 = (94, 100, 100, 20)$
- $\tau_3 = (0, 1000, 1000, 120)$

and the priorities for FP 0 are $\mathcal{P}_1 = 3$, $\mathcal{P}_2 = 2$ and $\mathcal{P}_3 = 1$.

A.2 FP 1 and EDF 1

Three cyclic tasks of the form $\tau_i = (O_i, T_i, D_i, C_i)$ with:

- $\tau_1 = (0, 100, 100, 20)$
- $\tau_2 = (0, 150, 150, 40)$
- $\tau_3 = (0, 350, 350, 100)$

and the priorities for the FP 1 are $\mathcal{P}_1 = 3$, $\mathcal{P}_2 = 2$ and $\mathcal{P}_3 = 1$.

A.3 FP 2 and EDF 2

Three cyclic tasks of the form $\tau_i = (O_i, T_i, D_i, C_i)$ with:

- $\tau_1 = (0, 3, 3, 1)$
- $\tau_2 = (0, 5, 5, 1)$
- $\tau_3 = (0, 10, 10, 1)$

and the priorities for the FP 2 are $\mathcal{P}_1 = 3$, $\mathcal{P}_2 = 2$ and $\mathcal{P}_3 = 1$.

A.4 FP 3 and EDF 3

Three cyclic tasks of the form $\tau_i = (O_i, T_i, D_i, C_i)$ with:

- $\tau_1 = (0, 100, 100, 20)$
- $\tau_2 = (0, 150, 150, 40)$
- $\tau_3 = (0, 160, 160, 20)$

and the priorities for the FP 3 are $\mathcal{P}_1 = 3$, $\mathcal{P}_2 = 2$ and $\mathcal{P}_3 = 1$.