A Weak Simulation Relation for Real-Time Schedulability Analysis of Global Fixed Priority Scheduling Using Linear Hybrid Automata

Youcheng Sun, Giuseppe Lipari

April 2014

Research report LSV-14-06 **(Version 1)**

Laboratoire Spécification & Vérification

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

A Weak Simulation Relation for Real-Time Schedulability Analysis of Global Fixed Priority Scheduling Using Linear Hybrid Automata

Youcheng Sun[1] and Giuseppe Lipari[1,2]
{y.sun, g.lipari}@sssup.it

[1]Scuola Superiore Sant'Anna, Pisa, Italy
[2]Laboratoire Spécification et Verification – ENS de Cachan, France*

Abstract. In this paper we present an exact schedulability test for sporadic real-time tasks scheduled by the Global Fixed Priority Fully Preemptive Scheduler on a multiprocessor system. The analysis consists in modelling the system as a Linear Hybrid Automaton, and in performing a reachability analysis for states representing deadline miss conditions. To mitigate the problem of state space explosion, we propose a partial order relationship over the symbolic states of the model and we prove that this is a weak simulation relation. We then present an implementation of the analysis in a software tool, and we show that the use of the proposed model permits to analyse larger systems than other exact algorithms in the literature.

1 Introduction

A *real-time system* consists of a set of real-time tasks with timing constraints, executed on a single or multiprocessor platform. A real-time task is a piece of code that must be executed periodically or upon reception of an event. Each instance of the task is called a *job* and it is characterised by a *worst-case execution time* (i.e. an upper bound on the execution time of the corresponding piece of code), an *arrival time* (i.e. the instant at which the job is inserted in the ready queue of the operating system and could start executing) and a *deadline* (i.e. the instant in time within which it must completed). The *response time* of a job is the length of interval between its arrival and the time instant it finishes execution.

Typically, real-time tasks can be modelled as *periodic tasks*, if the arrival times of any two consecutive jobs are separated by a constant amount of time called *period*; or as *sporadic tasks* if this interval is unknown but we can establish a lower bound called *minimum interarrival time*. Real-time tasks are

* This research has been carried out while G. Lipari was a Marie Curie Fellow at the Laboratiore Spécification et Vérification del École Normale Supérieure de Cachan, France, funded by the European Union Seventh Framework Programme (project RBUCE-UP, FP7/2007-2013) under grant agreement No. 246556.

scheduled by a certain scheduling algorithm that decides which task executes at each instant on all processors.

One important problem for real-time systems is to assess the *schedulability* of a set of tasks on platform by a certain scheduling algorithm: a task set is said to be *schedulable* if all jobs complete before their deadlines.

One of the most popular scheduling algorithms in the programming practice is the *Fixed Priority Fully Preemptive* scheduler: each task is assigned a fixed priority, and jobs are ordered in the ready queue by decreasing priority; if a low priority task is executing and a higher priority task is activated, the latter can *preempt* the former and execute in its place.

Analysing the schedulability of a set of periodic or sporadic tasks under fixed priority scheduling is a fundamental problem for real-time system design and development. Since the seminal work of Liu and Layland [12], the fixed-priority scheduling problem has been extensively studied. The problem has been solved exactly for single processor systems, by using a well known property: the worst-case response time of a task happens when it is activated simultaneously with its higher priority tasks, and all jobs are activated at their maximum frequency. Therefore, it suffices to simulate the system starting from this *critical instant* activating all subsequent jobs as soon as possible, until the first idle time.

In this paper, we consider the problem of checking the schedulability of a set of independent real-time sporadic tasks on a multiprocessor platform when the scheduling algorithm is the *Global Fixed Priority* (G-FP) Fully Preemptive scheduler. According to this scheduling algorithm, on a m-processor platform all jobs are ordered in one single ready queue by decreasing priority, and the first m highest priority jobs are executed at every instant.

Unfortunately, checking the schedulability of a task set scheduled by G-FP is extremely harder than for the single processor case. The fundamental problem is that no single *critical instant* exists: the worst-case response time of a task can be found anywhere in the schedule. Also, it is not true that the worst-case response time happens when all jobs are activated as soon as possible. An example is presented in Section 3.

In order to find the exact combination of arrival times that leads to the worst-case response time of a task, it is then necessary to explore all possible legal combinations of arrivals, and this number is so large that a brute-force approach fails already for very small task sets.

Therefore, most of the research in the literature has been focused on finding approximate upper bounds to the response times. However, to assess the pessimism of such approximate analyses, it is necessary to solve the problem exactly, i.e. to obtain necessary and sufficient conditions for the schedulability of a task set.

Contributions. In this paper, we address the problem of deriving an *exact analysis* for the schedulability of a set of sporadic real-time tasks scheduled by G-FP on a multiprocessor platform. We model the problem using the formalism of Linear Hybrid Automata [2, 1] to represent the tasks and the scheduler. In particular, deadline miss conditions are modelled as error locations in the automata.

The analysis consists in performing a reachability analysis for such error states. Since the complexity explodes for very small task sets, we propose a *weak simulation relation* between symbolic states and prove its correctness. The relation allows us to eliminate those states that are not useful for our reachability analysis, thus reducing the size of the state space. We present the implementation of our model in a software tool called FORTS, and we show that we can handle more complex task sets with respect to state-of-the-art exact algorithms. Also, we evaluate the pessimism of RTA-LC, the current state-of-the-art approximate analysis.

The paper is organised as follows. In Section 2 we discuss previous work on the same problem. In Section 3 we formally introduce the problem. In Section 4 we describe the formalism of Linear Hybrid Automata, and in Section 5 we present our model. The core of the paper is Section 6, in which we present the simulation relationship and prove its correctness. In Section 7 we report a set of experiments. Finally, in Section 8 we present our conclusions.

2 Related Work

Given the complexity of the problem, most of the work for G-FP scheduling is focused on obtaining an approximate schedulability test. Baker [3] developed sophisticated schedulability analysis techniques consisting in selecting a *problem window*, and in computing an upper bound to the maximum amount of workload of each individual task in that window. Bertogna and Cirinei [6] later applied this technique to perform an iterative response time analysis of global scheduling.

Guan et al [10] developed RTA-LC (Response Time Analysis with Limited Carry-in), the state-of-the-art approximate schedulability analysis for G-FP scheduling. RTA-LC integrates Bertogna and Cirinei's response time analysis and Baruah's technique [5] for Global Earliest Deadline First (G-EDF) scheduling of limiting the number of carry-in tasks. In this paper, we also evaluate how much pessimism lies in the RTA-LC test.

Regarding exact analysis, the first brute force approach to the problem was proposed Baker and Cirinei [4]: the test assumes discrete time parameters, and it consists in building a finite state machine that represents all possible combinations of arrival times and execution sequences for a task set scheduled by G-EDF. Unfortunately, the problem is so complex that the authors could analyse only tasks whose period was in the range $[3, 4, 5]$; the tool went in out-of-memory for values of $T = 6$.

Recently, Geeraerts et al. [9] improved over Baker and Cirinei's method by using an *antichain* technique. In particular, they proposed a *simulation relation* between states of the underlying finite automaton. An informal definition of simulation relation is the following:

> Given two states s_1 and s_2, we say that s_1 simulates s_2 (denoted as $s_1 \succeq s_2$) if and only if: 1) for every state s_2' successor of s_2, there exists a state s_1' successor of s_1 and $s_1' \succeq s_2'$; 2) if s_2 is an error state (i.e. it models a deadline miss), then also s_1 is an error state.

Thanks to this relation, when we find two states such that $s_1 \succeq s_2$, we can avoid analysing all paths starting from state s_2: in fact, if the error state is not reachable from s_1, then it is not reachable from s_2 either. This allows to significantly reduce the number of states to be analysed in the reachability analysis. The simulation relation proposed in [9] is valid for any fixed job-level scheduling algorithm, and this includes G-FP and G-EDF. However, the method is again based on discrete time; in their experiments the authors could analyse task sets with maximum period equal to $T = 8$ on 2 processors.

In this paper we take a different approach. We model the system as a Linear Hybrid Automaton (LHA) and then we perform our analysis on the corresponding symbolic state space. This allows us to analyse task set with much larger periods. As in [9], we define a weak simulation relation over the symbolic states, and prove its correctness for G-FP scheduling. This allowed us to considerably reduce the analysis time, and thus to analyse more complex task sets.

3 System Model

We consider the problem of checking the schedulability of a set of n sporadic tasks, scheduled on m identical processors, with $n > m$, by G-FP, so that all timing constraints are respected.

A sporadic task $\tau_i = (C_i, D_i, T_i)$ is characterised by a minimum interarrival time T_i, a relative deadline $D_i > 0$ and a worst-case execution time (WCET) C_i, all positive integer values. The task emits jobs whose activation time is separated by at least T_i units of time; each job executes for C_i units of time and must complete within D_i units of time from its activation. A task is said to have constrained-deadline if $D_i \leq T_i$; otherwise, it is called an unconstrained-deadline task. We assume that all jobs of the same task must be executed sequentially and cannot be parallelised. Each task is assigned a fixed and unique priority, and we assume lower task index correspond to higher priority.

In this paper we consider *global fixed-priority* (G-FP) fully-preemptive scheduling: the execution of a job can be suspended at any time to execute another higher priority job (preemption); the same job can later resume execution on a possibly different processor (migration).

As explained in Section 1, the main problem is that there is no critical instant, and that the worst-case response time of a task may not correspond to a situation in which all jobs arrive as soon as possible. To better understand the problem, consider the following example (from [5]): the system consists of 3 tasks $\tau_1 = (1, 1, 2)$, $\tau_2 = (1, 3, 3)$ and $\tau_3 = (5, 6, 6)$, to be scheduled by G-FP on a 2 processor platform. Assume that task τ_1 has the highest priority and task τ_3 the lowest. The schedule obtained when all tasks start at time 0 and arrive as soon as possible is shown in Figure 1a. However, if the second job of task τ_1 arrives at time instant 3 instead of 2, task τ_3 misses its deadline (Figure 1b).

Therefore, we cannot make any worst-case assumption on the arrival times of the jobs. In principle, we need to analyse all legal combinations of arrival instants.

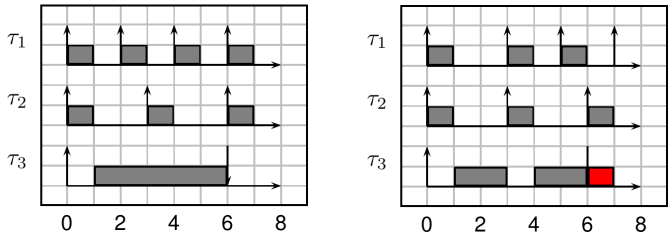


Fig. 1: Example of schedule of sporadic tasks (a) jobs arrive as soon as possible (b) second job of τ_1 is delayed.

4 Linear Hybrid Automata

A hybrid automaton [2][11] is a finite automaton associated with a finite set of variables continuously varying in dense time. In this section, we introduce the basic terminology and the definition of Linear Hybrid Automata.

Let $\mathsf{Var} = \{x_1, \ldots, x_n\}$ be a set of variables and $\dot{\mathsf{Var}} = \{\dot{x}_1, \ldots, \dot{x}_n\}$ be the set of variables' derivatives over time. A *linear constraint atom* over Var is of the form $\sum_{i=1}^{n} c_i x_i \sim b$, where c_i $(1 \leq i \leq n)$ and b are rational numbers and $\sim \in \{<, \leq, =, \geq, >\}$. A *linear constraint* $\mathcal{C}$ is the conjunction of a finite number of constraint atoms. A *valuation* ν over Var is a function that assigns a real value to each element in Var. The set of all possible valuations over Var is denoted as $V(\mathsf{Var})$. We write $\nu \models \mathcal{C}$ to represent that ν satisfies $\mathcal{C}$. The same notations can also be defined for $\dot{\mathsf{Var}}$.

Definition 1. *A Linear Hybrid Automaton* $\mathcal{A} = (\mathsf{Var}, \mathsf{Loc}, \mathsf{Init}, \mathsf{Lab}, \mathsf{Trans}, D, \mathsf{Inv})$ *consists of seven components: a finite set* Var *of variables; a finite set* Loc *of locations; a labeling function* Init *that specifies the initial linear constraint over variables; a finite set* Lab *of synchronisation labels including a* stutter *label* ϵ*; a finite set* Trans *of transitions; a labeling function* D *which assigns to each location* l *a linear constraint over variables' derivatives; and a labelling function* Inv *which assigns each location* l *a constraint, called* invariant*, over variables.*

The automaton can be in a location l as long as the current valuations of the variables satisfy $\mathsf{Inv}(l)$. A *transition* is a tuple $(l, \gamma, a, \alpha, l')$ consisting of a source location l, a target location l', a guard γ that is a linear constraint over Var, a synchronisation label $a \in \mathsf{Lab}$, and the transition relation α that updates values for variables in Var. We require that on each location, there is a *stutter transition* $(l, true, \epsilon, Id, l)$ where $Id = \{(\nu, \nu) | \nu \in V(\mathsf{Var})\}$ is the identical transition relation.

Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two LHA over a set of variables Var. Their parallel composition $\mathcal{A}_1 \times \mathcal{A}_2$ is the LHA $(\mathsf{Var}, \mathsf{Loc}_1 \times \mathsf{Loc}_2, \mathsf{Init}, \mathsf{Lab}_1 \cup \mathsf{Lab}_2, \mathsf{Trans}, D, \mathsf{Inv})$ such that:

- $\mathsf{Init}(l_1, l_2) = \mathsf{Init}_1(l_1) \wedge \mathsf{Init}_2(l_2)$.
- $((l_1, l_2), \gamma, a, \alpha, (l_1', l_2')) \in \mathsf{Trans}$ iff

1. $(l_1, \gamma_1, a_1, \alpha_1, l_1') \in \mathsf{Trans}_1$ and $(l_2, \gamma_2, a_2, \alpha_2, l_2') \in \mathsf{Trans}_2$;
2. $\gamma = \gamma_1 \wedge \gamma_2$.
3. either $a_1 = a_2 = a$, or either $a_1 = a \notin (\mathsf{Lab}_1 \cap \mathsf{Lab}_2)$ and $a_2 = \epsilon$ or $a_1 = \epsilon$ and $a_2 = a \notin (\mathsf{Lab}_1 \cap \mathsf{Lab}_2)$;
4. $\alpha = \alpha_1 \wedge \alpha_2$.

– $D(l_1, l_2) = D_1(l_1) \wedge D_2(l_2)$.
– $\mathsf{Inv}(l_1, l_2) = \mathsf{Inv}_1(l_1) \wedge \mathsf{Inv}_2(l_2)$.

A *concrete state* s of the LHA is in the form of (l, ν), where l is a location and $\nu \in V(\mathsf{Var})$. A state can change in two ways:

– A discrete step: $(l, \nu) \xrightarrow{\mathrm{a}} (l', \nu')$ which means there exists a transition $(l, \gamma, a, \alpha, l')$ and
$$\nu \models \gamma \wedge \nu' = \alpha(\nu) \wedge \nu' \models \mathsf{Inv}(l')$$
– A time step: $(l, \nu) \xrightarrow{\mathrm{t}} (l, \nu')$ where t is a real-value represents time elapse. And
$$\nu \models \mathsf{Inv}(l) \wedge \nu' \in \nu \uparrow^t_{D(l)} \wedge \nu' \models \mathsf{Inv}(l) \wedge t \geq 0$$
$\nu \uparrow^t_{D(l)}$ represents the set of valuations that can be reached by letting variables continuously evolve for t time units, according to derivatives constrained by D, and starting from the valuation ν.

We use $\rightarrow$ to represent a *step*, which could be either a discrete step or time step. We also define $\Rightarrow$ to denote a sequence of steps. And $\stackrel{t}{\Rightarrow}$ means that the accumulated time during the sequence of steps is t.

A *symbolic state* S of the LHA is a pair $(l, \mathcal{C})$, where l is a location and $\mathcal{C}$ is a linear constraint over variables. We can define a step and a sequence of steps for symbolic states by lifting the definitions of step and sequence of steps for concrete states. When it comes to symbolic states, the corresponding operations are performed on convex regions instead of concrete valuations on variables.

For a concrete state s and a symbolic state S, we say $s \in S$ if $s.l = S.l$ and $s.\nu \models S.\mathcal{C}$. The concrete state space and symbolic state space of a LHA $\mathcal{A}$ are represented by $\mathsf{space}(\mathcal{A})$ and $\mathsf{Space}(\mathcal{A})$ respectively.

5 Multiprocessor Schedulabilty in LHA

In this section we describe the automata used for modelling our scheduling problem. In particular, we use two different types of automata that synchronise each other: the task automata and the scheduler automaton.

5.1 The task automata

We start by presenting the LHA that models one single sporadic task. A concrete task automaton $\mathsf{TA} = (C, D, T)$ is depicted in Figure 2. It has two variables, p, c, and four locations.

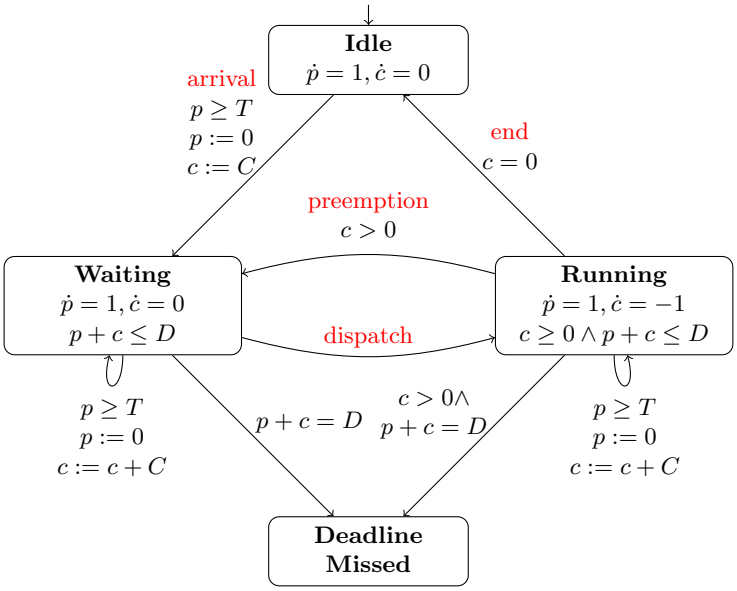


Fig. 2: Task Automaton

Variable p represents the *time passed* since the last activation of the task, and its rate is always 1. Every time a new job arrives, p is reset to 0. Variable c represents the *remaining computation time* of a task. Its rate can be 0, when the task does not execute, or -1 when the task executes.

The automaton works as follows. Initially, it is in state Idle. From there, when the guard constraint $p \geq T$ is satisfied, it can move at any time non-deterministically to location Waiting. Along this transition, p variable is reset to 0 and C is assigned to c variable. Also, it synchronises with the scheduler (see next section) on the arrival label.

While in Waiting, the rate of c will remain equal to 0. The automaton moves to location Running (where the rate of c is set to -1) after synchronising with the scheduler on label dispatch; and it can move back to location Waiting after synchronising with the scheduler on label preemption.

We say a task is *active* if it is in locations Waiting or Running. For an active task, its *slack* is the difference between the time left before the task deadline, and the remaining computation time: slack $= D - p - c$. When slack becomes less than 0, a task misses its deadline. Hence the invariant $p + c \leq D$ in Waiting and Running. In location Running, the invariant $c \geq 0$ forces the task to move to location Idle when the current instance has completed its execution.

We allow unconstrained-deadline tasks, thus there could be a new job arrival for an active task. This is modelled as a non-deterministic transition from Waiting or Running to itself. Since the new instance must wait for its precedence completes, variable c incremented by C with the transition.

In the following we will denote as TA_i the automaton corresponding to the i-th task in the system, with q_i, c_i, p_i its location, left computation time and passed time, respectively, and with $\mathsf{arrival}_i, \mathsf{end}_i, \mathsf{dispatch}_i, \mathsf{preemption}_i$ the corresponding synchronisation labels.

5.2 Scheduling automaton

Given a set of tasks $\mathcal{T} = \{\mathsf{TA}_1, \ldots, \mathsf{TA}_n\}$, set A is defined as the set of active tasks that are in locations Waiting or Running. and set R denotes the set of tasks that are in location Running.

Let $\mathsf{Scheduler} : 2^{\mathcal{T}} \to 2^{\mathcal{T}}$ be a *scheduling function* that, given a set of active tasks, returns the set of executing tasks: $\mathsf{R} = \mathsf{Scheduler}(\mathsf{A})$. The function must respect the following properties:

- $\forall \mathsf{A} \subseteq \mathcal{T}\ \ \mathsf{Scheduler}(\mathsf{A}) \subseteq \mathsf{A}$
- $\forall \mathsf{A} \subseteq \mathcal{T}\ \ |\mathsf{Scheduler}(\mathsf{A})| = \min(m, |\mathsf{A}|)$

The first property tells us that the scheduler only selects for execution tasks that are active. The second property tells us that at most m tasks can execute in the system, and the scheduler is not allowed to not execute an active task if there is some free resource. A scheduling function that respects the latter property is also called a *work conserving* scheduler. In this paper, we consider a G-FP Scheduler, which chooses $\min(m, |\mathsf{A}|)$ highest priority tasks to run.

The scheduling function can be modelled as a finite automaton synchronised with the task automata the system is composed of. More formally, the scheduling automaton $\mathsf{Sched} = \{m, \mathsf{Loc}, \mathsf{Lab}\}$ is characterised by:

- m is the number of identical processors in the system;
- Loc is the set of locations of the scheduler;
- $\mathsf{Lab} = \bigcup_i Lab_i, \quad Lab_i = \{\mathsf{arrival}_i, \mathsf{end}_i, \mathsf{dispatch}_i, \mathsf{preemption}_i\}$ is the set of synchronisation labels.

When set A changes to A', the automaton computes the new set R' according to the scheduling function $\mathsf{R}' = \mathsf{Scheduler}(\mathsf{A}')$. Then, it informs every task automaton TA_i of the fact that it is executing or not; this is done by using labels $\mathsf{dispatch}_i$ and $\mathsf{preemption}_i$. Therefore, all tasks that where in R and are not in R' synchronise on the $\mathsf{preemption}_i$ labels; and all tasks that were not in R but are now in R' synchronise on the $\mathsf{dispatch}_i$ labels.

An example of task automaton for $n = 3$ tasks on $m = 2$ processors in shown in Figure 3. The automaton consists of 19 states. The responsibility of a scheduling automaton is to synchronise with the task automata. In the figure, nodes depict locations, and the name of the location encodes the state of the system queue, and in some cases the event that just happened. For example, location E1E2W3 corresponds to the execution of task τ_1 and τ_2 on the two processors, and the task τ_3 waiting to be executed; location E1_arr2 represents that fact that, while task τ_1 is executing on one processor, task τ_2 has just arrived. Also, please note that all locations with names containing arr are assumed to be

committed locations, where time cannot elapse. Finally, on the edges we show the synchronisation labels in short form for graphical reasons, hence arr1 stands for $\mathsf{arrival}_1$, etc.

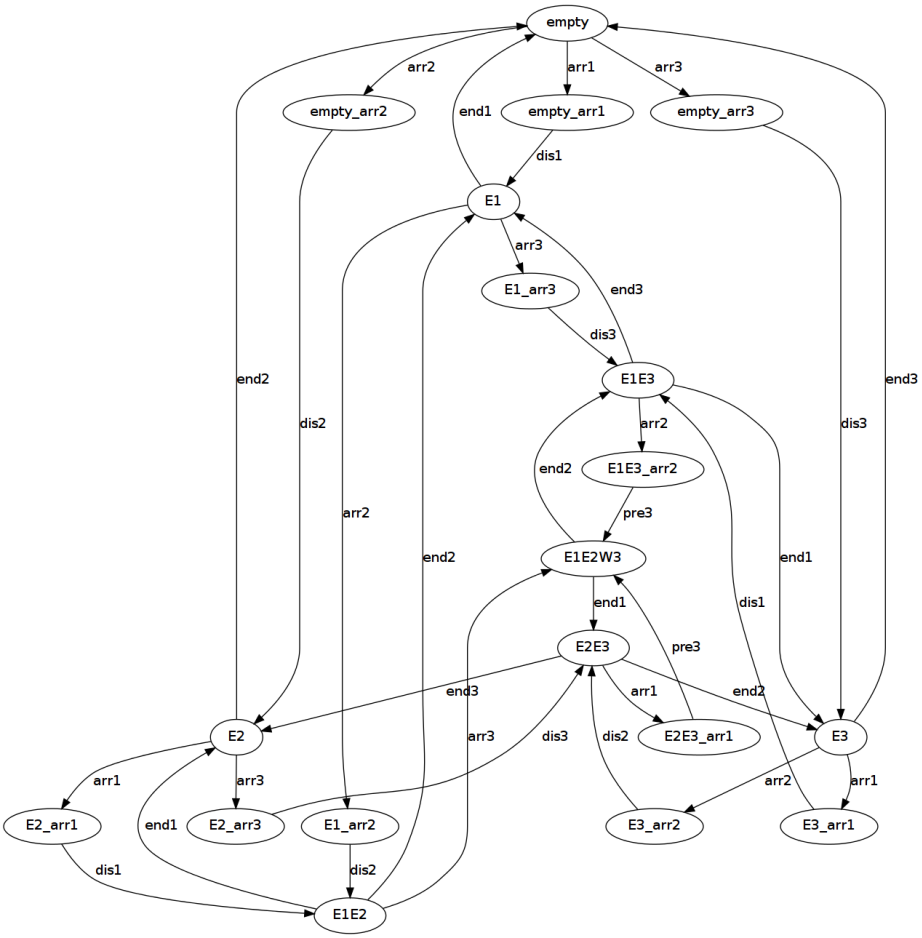


Fig. 3: Scheduler for 3 tasks on 2 processors

The number of locations needed for representing the scheduler automaton is exponential in the number of tasks. However, such locations can be automatically generated by using function Scheduler() for computing which task to execute and which task to suspend or preempt. Notice also that the location encodes the same information that is contained in the task automata presented above; in particular, executing tasks will be in location Running, whereas suspended tasks will be in location Waiting. Therefore, the scheduler automaton does not add any additional complexity to the problem; on the contrary, it restricts the

number of possible combinations of task locations: for example a lower priority task cannot be in the Running location if there are m higher priority tasks that are active.

Finally, a system automaton $\mathsf{SA} = (\mathcal{T}, \mathsf{Sched})$, is the parallel composition of the n task automata and one scheduler automaton, where

- $\mathcal{T} = \{\mathsf{TA}_1, \ldots, \mathsf{TA}_n\}$ is a set of n task automata;
- Sched is the scheduler automaton.

Given a state s in SA, $\mathsf{A}(s)$ and $\mathsf{R}(s)$ represent the set of active and running tasks in the system respectively. And we denote with q_0 current location of the scheduler automaton, and with q_i $(1 \leq i \leq n)$ the location of the i-th task automaton. The same notions are also applied to a symbolic state S.

6 Weak Simulation Relation in SA

Analysing the schedulability of a task set is equivalent to performing a reachability analysis of DeadlineMissed locations in SA. Due to the complexity of the multiprocessor schedule problem, the exploration of SA's (symbolic) state space will easily produce a "state explosion". To reduce the number of generated states, we propose a weak simulation relation for SA such that, given two states S_1 and S_2, if S_1 simulates S_2 then S_2 can be eliminated without interfering with the schedulability analysis.

6.1 Weak Simulation in concrete state space

Definition 2. *A weak simulation relation in the concrete state space of* SA *is a pre-order* $\succeq \subseteq \mathsf{space} \times \mathsf{space}$ *such that :*

1. $\forall s_1, s_2, s_4$ *s.t.* $s_1 \succeq s_2, s_2 \rightarrow s_4$ *there exists* s_3 *s.t.* $s_1 \Rightarrow s_3$ *and* $s_3 \succeq s_4$.
2. $\forall s_1, s_2$ *s.t.* $s_1 \succeq s_2 : \forall i$ $s_2.q_i$ $=$ DeadlineMissed *implies* $s_1.q_i =$ DeadlineMissed

If $s_1 \succeq s_2$*, we say that* s_1 *simulates* s_2.

Definition 3 (Slack-time simulation). *For automaton* SA*, the* slack-time *simulation relation* $\succeq_{st} \subseteq \mathsf{space} \times \mathsf{space}$ *is defined as follows:* $\forall s_1, s_2$, $s_1 \succeq_{st} s_2$ *if and only if*

$$\forall \tau_i : \quad s_1.p_i \geq s_2.p_i \quad \wedge \quad s_1.c_i \geq s_2.c_i$$

Proof. To prove that $\succeq_{st}$ is indeed a weak simulation relation, we must demonstrate that it satisfies the two properties stated in Definition 2, where the second point trivially holds for $\succeq_{st}$. Therefore, in this proof we address the first point: i.e. given $s_1 \succeq_{st} s_2$ and $s_2 \rightarrow s_4$, we prove that there exists s_3 such that $s_1 \Rightarrow s_3$ and $s_3 \succeq_{st} s_4$.

$s_2 \rightarrow s_4$ in SA can be a time step or a discrete step. The latter could be further differentiated, depending on whether it is caused by a task arrival or by a task completion. In the following we will analyse these cases one by one.

$s_2 \to s_4$ *is a time step with elapsed time* t*:* $s_2 \xrightarrow{t} s_4$. Let us consider a timed step sequence $s_1 \overset{t}{\Rightarrow} s_3$ with the same t as accumulated time; and we assume there is no new task arrival during this time interval. For any task τ_i, $s_3.p_i = s_1.p_i + t \geq s_2.p_i + t = s_4.p_i$. If a task τ_i is not in $\mathsf{A}(s_2)$, then $s_2.c_i = s_4.c_i = 0$; certainly, there will be $s_3.c_i \geq s_4.c_i$. Otherwise, task τ_i in $\mathsf{A}(s_2)$ also belongs to $\mathsf{A}(s_1)$. Suppose from s_1 to s_3 (s_2 to s_4), the time that τ_i stays in location Running is t_1 (t_2). Since the scheduler chooses tasks to run according to their fixed priority and $\mathsf{A}(s_2) \subseteq \mathsf{A}(s_1)$, t_1 will be no larger than t_2 and $s_3.c_i = s_1.c_i - t_1 \geq s_2.c_i - t_2 = s_4.c_i$. So, we proved that $s_3 \succeq_{st} s_4$.

$s_2 \to s_4$ *is a discrete step caused by the arrival of a task* τ_i. For such a step, only variables of the arriving task will change. Because that $s_1.p_i \geq s_2.p_i$, there exists also a discrete step from s_1 to s_3 triggered by τ_i's new arrival job. We have $s_3.p_i = s_4.p_i = 0$ and $s_3.c_i = s_1.c_i + C_i \geq s_2.c_i + C_i = s_4.c_i$. So, we proved $s_3 \succeq_{st} s_4$.

$s_2 \to s_4$ *is a discrete step caused by the completion of a task* τ_i. For such a step, only variables of the finishing task will change. And $s_4.p_i = s_2.p_i \leq s_1.p_i$ and $s_4.c_i = 0 \leq s_1.c_i$. Remember that in the definition of LHA, there is always a stutter transition from a location to itself. So, there is $s_1 \to s_1$ and $s_1 \succeq_{st} s_4$.

In conclusion, the pre-order $\succeq_{st}$ satisfies point one in Definition 2 also. Thus $\succeq_{st}$ is a weak simulation relation in SA.

6.2 Weak Simulation in symbolic state space

In this section, we extend the weak simulation relation $\succeq_{st}$ to symbolic states. Without restricting to our specific simulation relation, for two symbolic states S_1 and S_2, we say S_1 simulates S_2 if

$$\forall s_2 \in S_2 \ , \ \exists s_1 \in S_1 \quad s.t. \quad s_1 \succeq s_2$$

Remember that a symbolic state is a pair $(l, \mathcal{C})$ with a location l and a linear constraint $\mathcal{C}$. The linear constraint $\mathcal{C}$ can be represented by a convex region. In the following we use $\mathcal{C}$ to denote both a linear constraint and its convex region. In the context of $\succeq_{st}$ for concrete state space, there is no need to consider location names. Clearly, given two states $S_1 = (l_1, \mathcal{C}_1)$ and $S_2 = (l_2, \mathcal{C}_2)$, if $\mathcal{C}_1$ includes $\mathcal{C}_2$ (denoted as $\mathcal{C}_1 \supseteq \mathcal{C}_2$), then S_1 simulates S_2.

Assume we are in a N-dimensional space. Given two valuations $\nu = (x_1, x_2, \ldots, x_N)$ and $\nu' = (y_1, y_2, \ldots, y_N)$, we say ν *dominates* ν', denoted by $\nu \geq \nu'$, if for all i it holds $x_i \geq y_i$. We say a valuation ν is dominated by a convex region $\mathcal{C}$ if there exists some valuation $\nu' \models \mathcal{C}$ and $\nu' \geq \nu$. Given two convex regions $\mathcal{C}_1$ and $\mathcal{C}_2$, $\mathcal{C}_1$ is said to dominate $\mathcal{C}_2$, denoted as $\mathcal{C}_1 \geq \mathcal{C}_2$ if all $\nu \models \mathcal{C}_2$, ν is dominated by $\mathcal{C}_1$. We can see that the domination relation is transitive. With the concept of domination in mind we can define the following weak simulation relation.

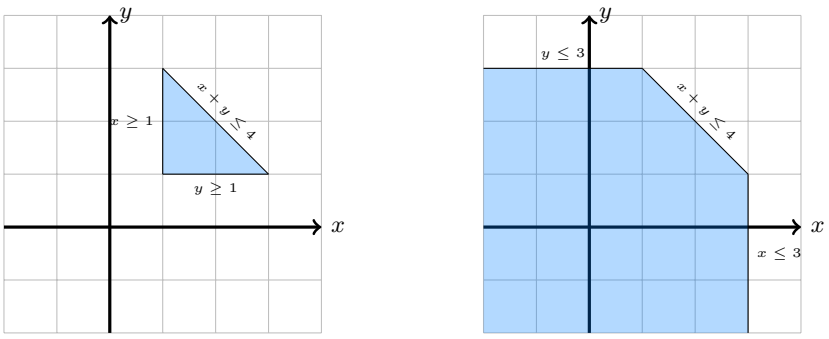
Definition 4. *For the* SA *automaton, the* slack-time *simulation relation* $\succeq_{st} \subseteq \mathsf{Space} \times \mathsf{Space}$ *is defined such that* $\forall S_1, S_2$, $S_1 \succeq_{st} S_2$ *if and only if* $S_1.\mathcal{C}$ *dominates* $S_2.\mathcal{C}$.

Proof. From the definition of domination.

We now need an efficient method for checking if two convex regions are in a relationship of domination. To do this, we first define a widening operator ∇.

Given a convex region $\mathcal{C}$, its widening $\nabla(\mathcal{C})$ is the convex region that can be obtained as follows:

- Construct linear constraints $\mathcal{C}'$ in $2\times N$ dimensional space $(x_1, \ldots, x_N, y_1, \ldots, y_N)$ such that
$$(y_1, \ldots, y_N) \models \mathcal{C} \quad \wedge \quad \forall i,\ x_i \leq y_i$$
- Remove the space dimensions higher than N in $\mathcal{C}'$.



(a) the original convex region (b) after windening

Fig. 4: A convex region $\mathcal{C}$ and its windening $\nabla(\mathcal{C})$

$\nabla(\mathcal{C})$ represents the largest region that is dominated by $\mathcal{C}$. $\forall \nu \in \nabla(\mathcal{C})$, there exists a $\nu' \in \mathcal{C}$ such that $\nu' \geq \nu$ and vice versa; this means $\mathcal{C} \geq \nabla(\mathcal{C})$ and $\nabla(\mathcal{C}) \geq \mathcal{C}$. To demonstrate how this widening operator works, we show a simple example in Figure 4.

Lemma 1. *Given two convex regions $\mathcal{C}_1$ and $\mathcal{C}_2$, $\mathcal{C}_1 \geq \mathcal{C}_2$ if and only if $\nabla(\mathcal{C}_1)$ includes $\nabla(\mathcal{C}_2)$.*

Proof. We first prove that $\mathcal{C}_1 \geq \mathcal{C}_2 \Rightarrow \nabla(\mathcal{C}_1) \supseteq \nabla(\mathcal{C}_2)$. Since $\mathcal{C}_1 \geq \mathcal{C}_2 \geq \nabla(\mathcal{C}_2)$ and $\nabla(\mathcal{C}_1)$ is the largest region dominated by $\mathcal{C}_1$, we get $\nabla(\mathcal{C}_1) \supseteq \nabla(\mathcal{C}_2)$.

Then we prove $\nabla(\mathcal{C}_1) \supseteq \nabla(\mathcal{C}_2) \Rightarrow \mathcal{C}_1 \geq \mathcal{C}_2$. From $\nabla(\mathcal{C}_1) \supseteq \nabla(\mathcal{C}_2)$, we have $\mathcal{C}_1 \geq \nabla(\mathcal{C}_1) \geq \nabla(\mathcal{C}_2) \geq \mathcal{C}_2$. So, $\mathcal{C}_1 \geq \mathcal{C}_2 \Leftrightarrow \nabla(\mathcal{C}_1) \supseteq \nabla(\mathcal{C}_2)$ and the lemma is proved.

6.3 Schedulability Analysis in SA

In this section, we formulate the algorithm to explore the state space of SA for schedulability analysis. The pseudo-code of the Schedulability Analysis algorithm in SA (SA-SA) is shown in Algorithm 1. S_0 is the initial state of SA, R

denotes the set of reachable states in SA and F is the set of states representing deadline miss. The Post operation returns the set of states that can be reached in a single transition by states in R. If some state in F is reachable, then the task set encoded in SA is deemed not-schedulable.

$\mathsf{Max}^{\succeq}(R')$ is defined as $\{S \in R' | \nexists S' \in R' \ s.t. \ S' \succeq_{st} S\}$. At line 8 of the algorithm, $\mathsf{Max}^{\succeq}$ operation eliminates all simulated states from R'. From the definition of $\succeq_{st}$, if some state in R' can reach a DeadlineMissed location, so can some state in $\mathsf{Max}^{\succeq}(R')$. When there is no new reachable states (line 9), the algorithm terminates and the task set is deemed schedulable.

We can also define $\mathsf{Max}^{\supseteq}(R') = \{S \in R' | \nexists S' \in R' \ s.t. \ S'.l = S.l \wedge S'.C \supseteq S.C\}$. This is a common strategy to eliminate redundant states. If we replace $\mathsf{Max}^{\succeq}(R')$ with $\mathsf{Max}^{\supseteq}(R')$, we obtain a version of SA-SA that does not use the simulation relation. In Section 7.2, we will compare the efficiency of these two versions of SA-SA, with and without simulation relation.

Different from previous exact analysis techniques in discrete time domain, SA-SA works in continuous time domain, which makes it less sensitive to the values of task parameters. For example, given a task set $\mathcal{T}_1 = \{(C_1, D_1, T_1), \ldots, (C_n, D_n, T_n)\}$, we enlarge every task parameter by multiplying 10 and obtain $\mathcal{T}_2 = \{(C_1 \cdot 10, D_1 \cdot 10, T_1 \cdot 10), \ldots, (C_n \cdot 10, D_n \cdot 10, T_n \cdot 10)\}$. When we apply SA-SA on $\mathcal{T}_1$ and $\mathcal{T}_2$, the the number of states generated at each step will be exactly the same for the two cases.

We have implemented SA-SA in the software FOrmal Real-Time Scheduler (FORTS) [15].

Algorithm 1: Schedulability Analysis in SA (SA-SA)

```
1:  R ← {S_0}
2:  while true do
3:      P ← Post(R)
4:      if P ∩ F ≠ ∅ then
5:          return NOT schedulable
6:      end if
7:      R' ← R ∪ P
8:      R' ← Max^≽(R')
9:      if R' = R then
10:         return schedulable
11:     else
12:         R ← R'
13:     end if
14: end while
```

7 Experiments

In this section we report the results of applying the SA-SA algorithm to randomly generated schedulability problems. Each task set in the experiment is characterised by a tuple (m, n, U), where m is the number of processors, n is the number of tasks in the task set and U is the total utilisation of the task set (i.e. $U = \sum_{i=1}^{n} \frac{C_i}{T_i}$). Given a tuple of (m, n, U), we randomly generated a task set according to the Randfixedsum algorithm [8]. Task periods were selected from the set $\{20, 30, 40, 50, 60, 100\}$.

In case of constrained-deadline tasks, we required the ratio between relative deadline and period $\frac{D_i}{T_i} \in [0.8, 1]$; for unconstrained-deadline tasks, $\frac{D_i}{T_i} \in [0.8, 2]$. Priorities were assigned by the Deadline Monotonic strategy; that is, a task with shorter deadline is assigned higher priority.

7.1 Comparison with RTA-LC

Although being state-of-the-art analytic schedulability test for G-FP scheduling problem, RTA-LC is still pessimistic. It is interesting and meaningful to see how much gap there is between the approximate result of RTA-LC and the exact test SA-SA. To the best of our knowledge, this is the first comparison in literature between RTA-LC with an exact test for G-FP scheduling.

We first set $m \in \{2, 3\}, n = 5$ and U was chosen from $[\frac{1}{2}m, m]$. For each utilisation level, we randomly generated 100 constrained-deadline task sets. We then applied RTA-LC and SA-SA on these task sets, respectively, and recorded the number of schedulable task sets for each method.

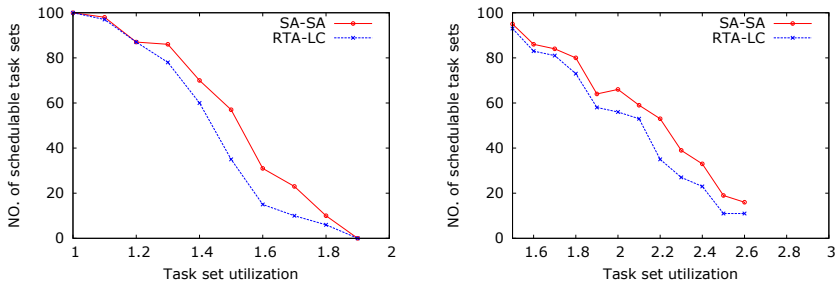
Results are plotted in Figure 5. As can be seen, there is a considerable number of schedulable task sets that RTA-LC failed to find. For example, when $m = 2$ and $U = 1.5$, RTA-LC found 35 schedulable task sets, whereas there were actually 57 schedulable task sets. Similar graphs hold for the case of $n = 5$ and $m = 3$[1].

7.2 Complexity of SA-SA

In this section, we are interested in evaluating algorithm SA-SA regarding the time for performing the analysis and its state space size. All tests are performed on a MacBook with 2.5 GHz Intel Core i5 and 8 GB memory.

First, we generated 100 constrained-deadline task sets with $n = 5$ tasks on $m = 2$ processors, with utilisation randomly chosen in $[1, 1.6]$. We applied SA-SA and SA-SA-WOS (WithOut Simulation) to each task set, and we recorded the time spent and final state space size for the two. Results are reported in

[1] Experiments with larger number of tasks take much more time to execute, and we were not able to complete a full round of experiments for $n = 6$ before the submission deadline. In case of acceptance, we will report the results of these experiments in the final paper.

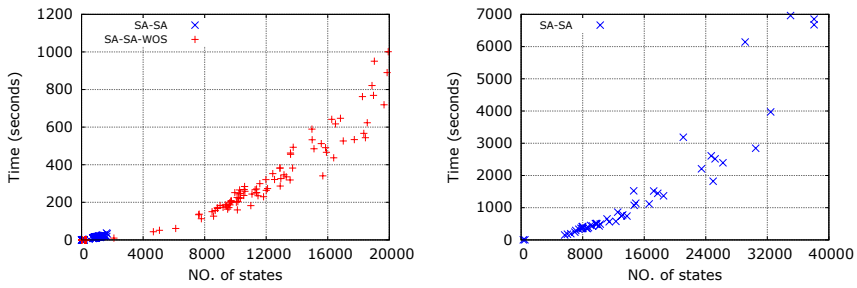


(a) $m = 2$ (b) $m = 3$

Fig. 5: RTA-LC vs. SA-SA

Figure 6a. To avoid the danger of going out of memory, we restricted the experiment to task sets such that SA-SA-WOS generated less than 20000 states for schedulability check.

Figure 6a shows the number of states generated and the corresponding analysis time for checking the schedulability of each task set. By employing $\succeq_{st}$, the number of generated states (and hence the execution time) for schedulability check is reduced considerably. Similarly, we applied SA-SA on 50 randomly generated unconstrained-deadline task sets with $m = 2$, $n = 6$, and $U_i \in [1, 2]$. We recorded the time cost and number of states produced. Results are in Figure 6b.



(a) $n = 5$ (b) $n = 6$

Fig. 6: Complexity tests (2 processors): state space size and analysis time

8 Conclusions and future work

In this paper, we propose a weak simulation relation for reducing the complexity of the exact analysis of Global Fixed Priority on a multi-processor platform. Compared to previous work on exact analysis, our methodology allows more complex task sets: we are able to analyse sets of 5 and 6 tasks on 2 and 3 processors with arbitrary values of the periods.

Unfortunately, even with our approach, the complexity remains too high for using our method on practical problems with large task sets. We are currently investigating other ways of reducing the complexity: first, we would like to use a different representation for the symbolic states (e.g. Octagons [14] or Difference Bound Matrices [13]), which requires an approximate analysis similar the ones used for Stopwatch Timed Automata and Time Petri Nets [7]. Second, we are currently investigating the possibility to enhance and extend our simulation relation, so to further reduce the state space.

References

1. Rajeev Alur, Costas Courcoubetis, Nicolas Halbwachs, Thomas A Henzinger, P-H Ho, Xavier Nicollin, Alfredo Olivero, Joseph Sifakis, and Sergio Yovine. The algorithmic analysis of hybrid systems. *Theoretical computer science*, 138(1):3–34, 1995.
2. Rajeev Alur, Costas Courcoubetis, Thomas A Henzinger, and Pei-Hsin Ho. *Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems*. Springer, 1993.
3. T. Baker. Multiprocessor edf and deadline monotonic schedulability analysis. *IEEE Real-Time Systems Symposium (RTSS)*, 2003.
4. TheodoreP. Baker and Michele Cirinei. Brute-Force Determination of Multiprocessor Schedulability for Sets of Sporadic Hard-Deadline Tasks. In Eduardo Tovar, Philippas Tsigas, and Hacène Fouchal, editors, *Principles of Distributed Systems*, volume 4878 of *Lecture Notes in Computer Science*, page 62–75. Springer Berlin Heidelberg.
5. S. Baruah. Techniques for Multiprocessor Global Schedulability Analysis. In *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, pages 119–128, 2007.
6. Marko Bertogna and Michele Cirinei. Response-time analysis for globally scheduled symmetric multiprocessor platforms. In *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, pages 149–160. IEEE, 2007.
7. Giacomo Bucci, Andrea Fedeli, Luigi Sassoli, and Enrico Vicario. Timed state space analysis of real-time preemptive systems. *Software Engineering, IEEE Transactions on*, 30(2):97–111, 2004.
8. Paul Emberson, Roger Stafford, and Robert I Davis. Techniques for the synthesis of multiprocessor tasksets. In *1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, pages 6–11, 2010.
9. Gilles Geeraerts, Joël Goossens, and Markus Lindstrõm. Multiprocessor schedulability of arbitrary-deadline sporadic tasks: complexity and antichain algorithm. *Real-Time Systems*, 49(2):171–218, 2013.

10. Nan Guan, Martin Stigge, Wang Yi, and Ge Yu. New response time bounds for fixed priority multiprocessor scheduling. In *Real-Time Systems Symposium, 2009, RTSS 2009. 30th IEEE*, pages 387–397. IEEE, 2009.
11. Nicolas Halbwachs, Yann-Erick Proy, and Patrick Roumanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, 11(2):157–185, 1997.
12. C.L. Liu and J.W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the Association for Computing Machinery*, 20(1):46–61, January 1973.
13. Antoine Miné. A New Numerical Abstract Domain Based on Difference-Bound Matrices. In *Proceedings of the Second Symposium on Programs as Data Objects*, PADO '01, pages 155–172, London, UK, UK, 2001. Springer-Verlag.
14. Antoine Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 19(1):31–100, 2006.
15. Youcheng Sun and Giuseppe Lipari. FOrmal Real-Time Scheduler (FORTS). Web page: https://github.com/glipari/forts, January 2014.