A pre-order relation for exact schedulability test of sporadic tasks on multiprocessor Global Fixed-Priority scheduling

Youcheng Sun, Giuseppe Lipari

A Pre-Order Relation for Exact Schedulability Test of Sporadic Tasks on Multiprocessor Global Fixed-Priority Scheduling*

Youcheng Sun
Scuola Superiore Sant'Anna
y.sun@sssup.it

Giuseppe Lipari
Univ. Lille, CNRS, Centrale Lille, UMR 9189 - CRIStAL - F-59000 Lille, France
giuseppe.lipari@univ-lille1.fr

December 9, 2015

Abstract

In this paper we present an exact schedulability test for sporadic real-time tasks scheduled by the Global Fixed Priority (G-FP) Fully Preemptive Scheduler on a multiprocessor system. The analysis consists in modeling the system as a Linear Hybrid Automaton (LHA), and in performing a reachability analysis for states representing deadline miss conditions. To mitigate the problem of state space explosion, we propose a pre-order relationship over the symbolic states of the model: states that are simulated by others can be safely eliminated from the state space.

We also formulate the concept of decidability interval with respect to a set of constrained-deadline sporadic tasks on multiprocessor. The decidability interval is a bounded time interval such that, if a deadline miss occurs in the schedule, then it is possible to find a configuration of arrival times for the tasks such that the deadline miss happens within the bounded interval. Vice versa, if no configuration of arrival times produces a deadline miss in the bounded interval, then no deadline miss is ever possible in the schedule. Hence we prove that the schedulability analysis problem is decidable, and we provide a formula for computing the decidability interval. To our knowledge, this is the first time such a time interval is proposed for sporadic tasks running on multiprocessor.

The proposed schedulability analysis has been implemented in a software tool. For the first time we assess the pessimism of the state-of-the-art approximate schedulability test through experiments. Moreover, we show

*This paper has been accepted by the Real Time Systems Journal. This is the submitted version, the final accepted version is avaliable at the journal's web site.

that the use of the proposed model permits to analyse tasks with more general parameter values than other exact algorithms in the literature. Nevertheless, even with our approach the complexity remains too high for analysing practical task sets with more than 7 tasks.

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.

One fundamental problem for real-time systems is to assess the *schedulability* of a set of tasks on a platform by a certain scheduling algorithm: a task set is said to be *schedulable* if all jobs complete executions 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 lower priority task is executing and a higher priority task is activated, the latter can *preempt* the former and execute in its place.

Since the seminal work of [24], 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* and 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 m highest priority jobs are executed at every instant.

Unfortunately, there is no easy solution for checking the schedulability of a task set scheduled by G-FP. The difficulty comes from two facts:

- 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.

- On the other hand, the sporadic behaviour of the tasks increases the number of possible interleavings.

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 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 ([1, 2]) to represent the tasks and the scheduler. In particular, deadline miss conditions are modeled as error locations in the automata. The analysis consists in performing a reachability analysis for such error states. Due to the non-deterministic sporadic task activations, the analysis complexity explodes for very small task sets. To defer the state explosion, 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. Furthermore, we prove the decidability of the proposed analysis by demonstrating that the schedulability test of a set of sporadic tasks under G-FP scheduling policy can be done in a bounded time interval, called *decidability interval*. We present the implementation of our model in a software tool, and we show that it can handle more complex task sets with respect to state-of-the-art exact algorithms based on discrete time. Here, by "more complex" we mean that tasks' parameters can be generic values and, as we are going to see, this makes a critical difference between previous works on exact multiprocessor schedulability analysis and our solution. Also, we evaluate the pessimism of current state-of-the-art approximate schedulability analysis of G-FP scheduling over sporadic tasks. Through extensive experiments, we investigate the factors that could affect the runtime performance of the proposed schedulability analysis.

Limitations. Unfortunately, as the number of sporadic tasks grows beyond 7, and for more than 4 processors, the complexity rises so much that all exact analysis techniques proposed so far can hardly terminate on current desktop computers, even when using our weak simulation relation. This is due to the exponential nature of the problem and it can only be mitigated by future improvements of the simulation relation. Thus, we will continue to work in this direction in the hope to further enhance the practicability of the method.

Organisation. The remainder of this paper is organised as follows. In Section 2 we discuss previous works on the same problem. In Section 3 we formally introduce the problem. In Section 4 we describe the formalism of Linear Hybrid Automata. In Section 5 we present our model for the G-FP scheduling problem over sporadic tasks. The core of the paper is Section 6, in which we propose the

pre-order relationship and prove its correctness as a weak simulation relation in our proposed LHA model; we also formalise the exact schedulability test, based on reachability analysis in LHA. In Section 7 we introduce the concept of decidability interval and demonstrate how to derive it for G-FP scheduling of sporadic tasks, then apply the obtained decidability interval result to prove the decidability of our exact test in case of constrained-deadline tasks. In Section 8 we report a set of experiments. Finally, in Section 9 we conclude the paper.

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.

The general properties of multiprocessor scheduling have been discussed in many previous papers. [19, 14] proposed upper bounds to the *feasibility interval* of a set of *periodic* tasks scheduled upon a multiprocessor (uniform or heterogeneous).

[3] developed sophisticated schedulability analysis techniques consisting in selecting a *problem window*, and in computing an upper bound to the maximum amount of workload and interference of each individual task in that window. Since then, reducing the pessimism in the estimation of workload and interference has been the main approach to improve analysis precision. [8] applied this technique to perform an iterative response time analysis of global scheduling. [22] developed RTA-LC (Response Time Analysis with Limited Carry-in) schedulability analysis for G-FP scheduling by integrating the response time analysis in [8] and the technique in [6] for Global Earliest Deadline First (G-EDF) scheduling of limiting the number of carry-in tasks. [30] developed RTA-CE (RTA with Carry-in task sets Enumeration) that explicitly enumerates all possible carry-in task sets. Among approximate G-FP schedulability tests, RTA-CE achieves the best precision. In this paper, we also evaluate how much pessimism lies between the RTA-CE test and the exact analysis.

Regarding exact analysis, the first brute force approach to the problem was proposed in [5]: 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 is in the range $\{3, 4, 5\}$; the tool produces an out-of-memory error for values of $T = 6$.

[15] proposed an exact schedulability test for a set of *periodic tasks*, but they did provide neither a tool, nor experiments with task sets. We believe that their algorithm is very complex and a naive implementation would not scale to a large number of tasks. [20] proposed a timed-automata model for schedulability analysis of *periodic tasks*. However, periodic tasks are simpler to analyse than sporadic tasks: we will provide a detailed comparison in Section 8.4.

Recently, [17] improved over [5] 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 produces a significant reduction on the number of states to be analysed in the reachability analysis. The simulation relation proposed in [17] is valid for any fixed job-level scheduling algorithm, including G-FP and G-EDF. Besides, [9] studied the feasibility problem of sporadic tasks in multiprocessor by reducing it to a safety game, where the two players are the scheduler and the set of the tasks respectively. As shown in [18], the antichain technique in [17] can be applied to [9] in order to improve the efficiency.

However, all such methods rely on explicit (discrete) techniques for time analysis and are limited to tasks with very small discrete parameters. For example, in their experiments [17] 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. As in [17], we define a weak simulation relation over the symbolic states, and prove its correctness for G-FP scheduling. This allows us to considerably reduce the analysis time, and thus to analyse more complex task sets. Due to the different features between explicit and symbolic techniques for state space exploration, when tasks are with small parameters, it is possible that existing works on exact multiprocessor schedulability analysis are more efficient than our solution; however, our exact analysis is the only work that can handle task sets with general configurations.

3 System Model

We consider the problem of checking the schedulability of a set of n independent 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 and a worst-case execution time (WCET) C_i ($\leq \min\{D_i, T_i\}$). With an abuse of notation, T_i is sometimes called *period*. The utilisation of a task is defined as $U_i = \frac{C_i}{T_i}$. The task emits jobs whose activation time is separated by at least T_i units of time; each job executes for at most 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. An arbitrary-deadline task has no requirement on the values of relative deadline D_i and period T_i, that is, D_i could be less than,

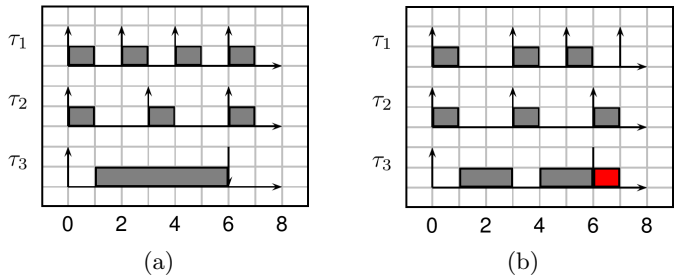


Figure 1: Example of schedule of sporadic tasks (a) jobs arrive as soon as possible (b) second job of τ_1 is delayed.

equal to or larger than T_i. We assume that all jobs of the same task must be executed sequentially and cannot be parallelised. Each task is pre-assigned a fixed and unique priority, and we assume that a lower task index corresponds to a higher priority. We assume by convention that time 0 corresponds to the first arrival instant of any sporadic task in the system: in other words, no arrival event happens before time 0.

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*). G-FP scheduling is sustainable ([4]), i.e. given a schedulable task set, by decreasing the WCET C_i, increasing T_i or enlarging D_i of tasks, the task set remains schedulable.

As mentioned in Section 1, the main obstacle to perform an exact analysis 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 [6]): 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. Task τ_1 has the highest priority and τ_3 is the lowest priority task. The schedule obtained when all tasks start at time 0 and arrive as soon as possible is shown in Figure 1a, where all tasks meet their deadlines. 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).

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

4 Linear Hybrid Automata

A hybrid automaton ([1], [23]) 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 continuous 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* $\mathcal{A} = (\mathit{Var}, \mathit{Loc}, \mathit{Init}, \mathit{Lab}, \mathit{Trans}, D, \mathit{Inv})$ *consists of seven components:*

- *a finite set* Var *of variables;*
- *a finite set* Loc *of locations including an initial location* l_0*;*
- *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 labeling 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 which is used to update the values of the variables in Var. We require that on each location, there is a *stutter transition* $(l, \mathit{true}, \epsilon, \mathit{Id}, l)$ where $\mathit{Id} = \{(\nu, \nu) | \nu \in V(\mathsf{Var})\}$ 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: $(l, \nu) \xrightarrow{\mathrm{t}} (l, \nu')$ at which t is a real-value represents time elapsed, and

$$\nu \models \mathsf{Inv}(l) \wedge \nu' \in \nu \uparrow^t_{D(l)} \wedge \nu' \models \mathsf{Inv}(l) \wedge t \geq 0$$

  where $\nu \uparrow^t_{D(l)}$ represents the set of valuations that can be reached by letting variables continuously evolve for t time units, according to derivatives constrained by $D(l)$, and starting from the valuation ν.

We use $\rightarrow$ to represent a *generic 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 Schedulability in LHA

In this section we describe the automaton used for modeling our scheduling problem. In particular, we use two different types of automata that synchronise with each other: the task automata and the scheduler automaton. Indeed, the LHA model we are going to propose can be also encoded by using Timed Automata with stopwatches ([12]). As it will be clear in the next section, the only difference is that we allow some variables to decrease at unitary rate, whereas in Stopwatch Timed Automata all variables are either stopped or increasing at unitary rate. We think that using the LHA model is more straightforward to understand our analysis scheme in Section 6.

5.1 The task automata

We start by presenting the LHA that models one single sporadic task. Such a LHA model for the sporadic task is called *task automaton*. A concrete task

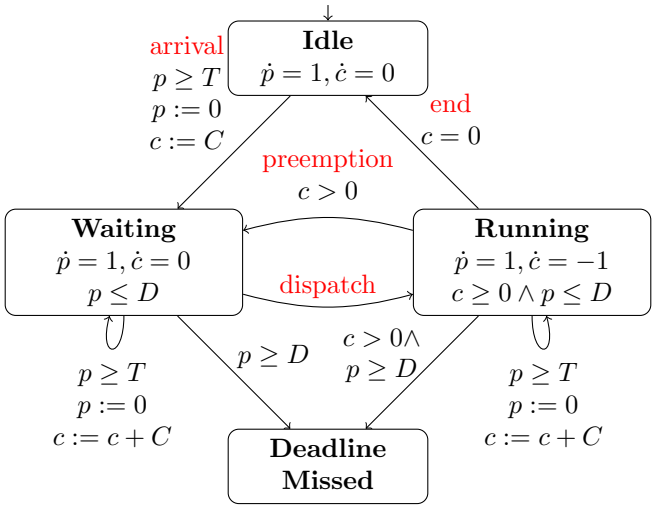


Figure 2: Task Automaton

automaton $\mathsf{TA} = (C, D, T)$ is depicted in Figure 2. It has two continuous variables p and c, and four locations.

Variable p represents the *time passed* since the latest 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, where $p \geq 0$ and $c = 0$; $p \geq 0$ models the fact that the first job release of a task can happen at any time. From there, when the guard constraint $p \geq T$ is satisfied, i.e. at least T time units have been passed since latest activation of the task, it can move non-deterministically to location Waiting. Along with this new job arrival transition, p variable is reset to 0 and C is assigned to variable c. Also, it synchronises with the scheduler (see next section) on the task arrival label. Notice that every task always executes for its WCET: as mentioned in Section 3, G-FP is *sustainable*, thus if the system is schedulable when every task always executes for its WCET, it is also schedulable when a task is allowed to execute for less than its WCET.

While in Waiting, the rate of c will remain equal to 0. The automaton moves to location Running after synchronising with the scheduler on label dispatch, which notifies that some processor is available for the task to run. While a task is running, the rate of c is set to -1, so its remaining computation time decreases. Its execution can be preempted by the arrival of a higher priority task, at which point the task will move back to location Waiting after synchronising with the scheduler on label preemption.

We say a task is *active* if it is in location Waiting or Running. An active task must finish its computation time before reaching its deadline. This means the c

variable must reach 0 no later than the time at which p reaches D. Otherwise, a task misses its deadline and goes (from Waiting or Running) to the Deadline-Missed location. If a task finishes its execution before deadline, i.e. $c = 0$ and $p \leq D$, the task is forced to move to location Idle (transition from Running to Idle).

In case that a task has unconstrained deadline, there could be a new job arrival for an active task. This is modeled as a non-deterministic transition from Waiting or Running to itself. Since the new instance must wait for its precedence completes, variable c is 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: $\mathsf{R} = \mathsf{Scheduler}(\mathsf{A})$.

In this paper, we consider a G-FP Scheduler, which chooses the $\min\{m, |\mathsf{A}|\}$ highest priority tasks to run.

The scheduling function can be modeled by a finite automaton synchronised with the task automata the system is composed of. More formally, the scheduling (or scheduler) 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;
- $\mathsf{Lab} = \bigcup_i Lab_i$ with $Lab_i = \{\mathsf{arrival}_i, \mathsf{end}_i, \mathsf{dispatch}_i, \mathsf{preemption}_i\}$ is the set of synchronisation labels.

The responsibility of a scheduling automaton is to synchronise with the task automata, i.e. to decide which tasks to run (staying in location Running) and which tasks to wait (staying in location Waiting). Every time a task completes its execution or releases a new job, the active task set A changes to A′ and a new running task set R′ is computed according to the scheduling function $\mathsf{R}' = \mathsf{Scheduler}(\mathsf{A}')$. Then, for the task that is in R but was not in R′, the scheduling automaton informs its preemption from the processor through synchronisation on the $\mathsf{preemption}_i$ label; and for the task that was not in R but is now in R′, the scheduling automaton synchronises on the $\mathsf{dispatch}_i$ label with it.

An example of scheduling automaton for $n = 3$ tasks on $m = 2$ processors is shown in Figure 3. 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 the 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*[1]. Finally, on the edges we show the synchronisation labels (in short form for graphical reasons), hence arr1 stands for $\mathsf{arrival}_1$, etc. For simplicity, when there is a preemption (for example τ_3 is preempted by the arrival of τ_1), we put the two synchronisation labels pre3 and dis1 in the same transition. This means between the two synchronisations, no time will elapse (as we assume no context switch cost). This can be realised by inserting a committed location in between the two. In the special case of one processor, a formal modeling of the fixed-priority scheduler can be found in [29].

The number of locations needed for representing the scheduler automaton is exponential in the number of tasks. 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 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 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.

The following theorem shows that SA models the real-time scheduling of a set $\mathcal{T}$ of sporadic tasks with G-FP, including all legal task arrival and execution patterns.

Theorem 1. *Given a task set $\mathcal{T}$, all tasks in $\mathcal{T}$ are guaranteed to meet their deadlines if and only if* DeadlineMissed *locations are not reachable in the system automaton* SA*.*

Proof. We must demonstrate that, if there is a deadline miss for the set of tasks under G-FP scheduling, there is a sequence of transitions in SA starting from the initial location until to the DeadlineMissed location. Vice versa, if some DeadlineMissed locations can be reached using a sequence of transitions, then there exists a configuration of arrival times for the sporadic instances such that one task will miss its deadline.

[1] A committed location is a location where time is not allowed to elapse; more details can be found in [7].

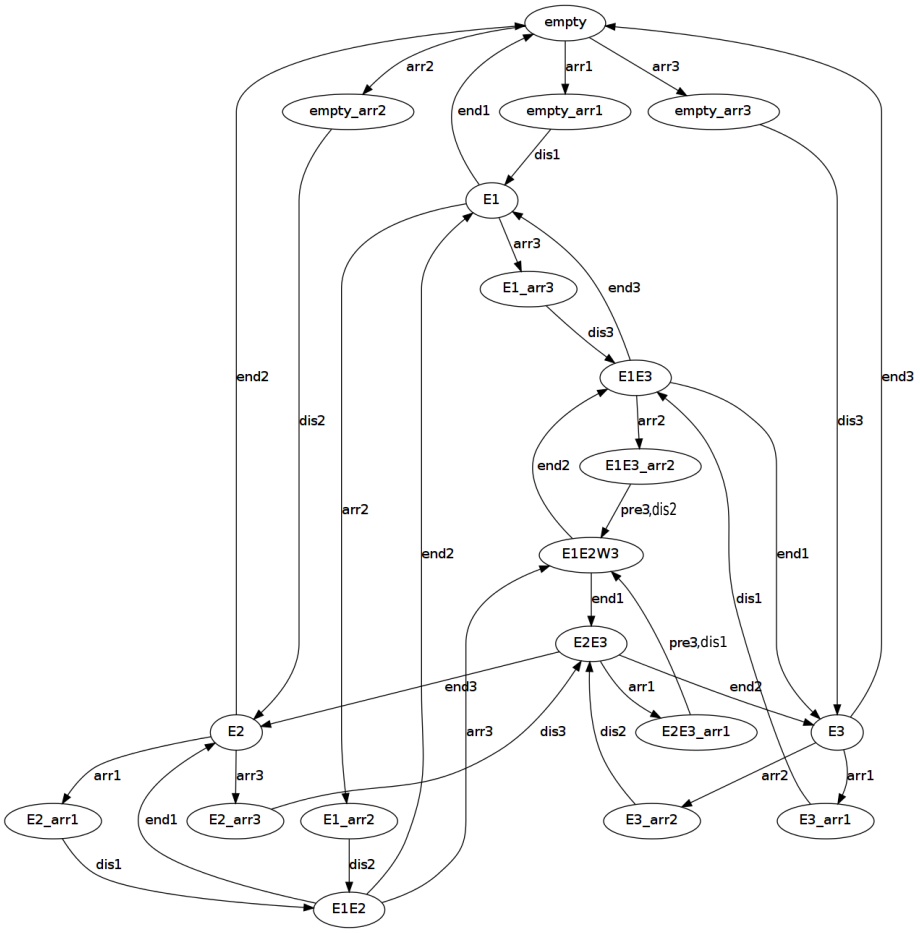


Figure 3: Scheduler for 3 tasks on 2 processors

Suppose that the task set is not schedulable under G-FP and the first deadline miss happens at time t. We now build a sequence of transitions (i.e. a *trace*) in the SA that reproduces the schedule.

Suppose that the first instance of a task τ_i is released at time $r_{i,1}$. Then, the corresponding TA remains in the Idle location while $p < r_{i,1}$, and the transition to location Waiting is taken when $p = r_{i,1}$; correspondingly p is reset to 0. The scheduling automaton Sched at each instant represents the state of the ready queue, so it mimics the state of the schedule. For example, upon arrival of a high priority task, a sequence of transitions synchronized by dispatch and/or preemption is triggered in zero time so that the TA corresponding to the high priority task is moved to location Running, whereas the TA corresponding to the preempted task is moved to location Waiting. If a task executes for Δt units of time, the corresponding variable c is decreased by the same amount, so the value of the variable always represents the execution time.

Successive releases at $r_{i,j}$ are treated differently: if the task has already completed its execution, the corresponding automaton is in location Idle, so they are treated as in the first release.

If at time $r_{i,j}$ the previous instance of τ_i has not yet completed, the TA may decide non-deterministically to update c and p to account for the new instance or decide to ignore the new arrival and analyse the previous one. In practice, in the analysis we need to consider both cases. If the instance that misses the deadline is the $(j-1)$-th instance of task τ_i, we ignore the release of the $j-th$ instance (which has no impact on the current schedule), and continue the analysis of the current instance until the DeadlineMissed location is reached.

If instead the deadline miss happens on a different instance of the same task or of another tasks, we update variable $c \leftarrow c+C$ and variable $p \leftarrow 0$ to account for the new instance.

Then, it is clear that each task arrival, execution, preemption and completion corresponds to a transition in the task automaton TA. Therefore, a deadline miss for a task always corresponds to the corresponding TA going into the DeadlineMissed location.

Vice versa, if there exists a sequence of transitions in SA that finally reaches the DeadlineMissed location, following the same reasoning as above, we can find a corresponding task arrival and preemption pattern in the scheduling of tasks that causes a deadline miss.

In conclusion, the task set is schedulable if and only if the location DeadlineMissed is not reachable in the system automaton SA. □

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. We denote with q_0 the 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

In the previous section we proved that analysing the schedulability of a task set is equivalent to performing a reachability analysis of DeadlineMissed locations in SA. Due to the non-deterministic and sporadic behaviour of task arrivals, 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 from state space without interfering with the final schedulability analysis result.

6.1 Weak simulation in concrete state space

We first discuss the weak simulation relation in concrete state space of SA, then we will extend it for symbolic states.

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 the following two conditions are satisfied:*

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.* $s_1 \succeq s_2 : \forall i\ s_2.q_i = \mathsf{DeadlineMissed}$ *implies* $s_1.q_i = \mathsf{DeadlineMissed}$.

If $s_1 \succeq s_2$*, we say that* s_1 *simulates* s_2*. If* $s_1 \succeq s_2$ *but* $s_2 \not\succeq s_1$*, we write* $s_1 \succ s_2$*.*

Roughly speaking, $s_1 \succeq s_2$ means that the scenario in s_1 is worse than in s_2 for tasks to finish execution before their deadlines.

The first condition in Definition 2 says that, given $s_1 \succeq s_2$ and given s_2 that performs a (discrete or time) step to s_4, there exists a state s_3 reachable from s_1 such that $s_3 \succeq s_4$. The second condition says that if some task in the simulated state (s_2) misses its deadline, so does it in the simulating state (s_1). For these two reasons, during state space exploration, we can safely eliminate states that are simulated by others without violating the reachability analysis result. Note that for two states $s_1 \succeq s_2$ and $s_2 \succeq s_1$, we only need to keep one of them.

Now, we present a specific pre-order relation in space(SA) that satisfies the definition of a weak simulation relation, thus it can be used to simplify the state space exploration in SA.

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

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

For an active task, the difference between the time left before its deadline and the remaining computation time is called the *slack* of the task. When the slack is less than 0, the task is doomed to miss its deadline. Intuitively, a smaller slack corresponds to a more urgent scenario. In a task automaton TA_i, the slack can be computed by $D_i - p_i - c_i$. Given two states s_1, s_2 such that $s_1 \succeq_{st} s_2$,

there is $s_1.p_i \geq s_2.p_i$ and $s_1.c_i \geq s_2.c_i$. So, an active task's slack in s_1 is no larger than its slack (if in s_2 it is also active) in s_2.

We now prove that $\succeq_{st}$ satisfies the two conditions for a weak simulation relation in Definition 2.

Theorem 2. *The pre-order relation $\succeq_{st}$ is a weak simulation relation in* space(SA).

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 $s_2 \to s_4$, we prove that there exists s_3 such that $s_1 \Rightarrow s_3$ and $s_3 \succeq_{st} s_4$.

$s_2 \to s_4$ 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.

1. *$s_2 \to s_4$ 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 let us assume that 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, a key observation for the proof is that $\forall i, s_1.c_i \geq s_2.c_i$ implies $\mathsf{A}(s_2) \subseteq \mathsf{A}(s_1)$; so 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 $s_3 \succeq_{st} s_4$.

2. *$s_2 \to s_4$ 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 $s_3 \succeq_{st} s_4$.

3. *$s_2 \to s_4$ 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. Then, $s_4.p_i = s_2.p_i \leq s_1.p_i$ 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

As continuous variables in LHA vary in dense time domain, the weak simulation relation in concrete state space cannot be applied to reachability analysis

directly. In this section, we extend the slack time weak simulation relation $\succeq_{st}$ to symbolic states.

We remind here the concepts in Section 4 that a symbolic state is defined as a pair $S = (l, \mathcal{C})$, where l is a location and $\mathcal{C}$ is a linear constraint (or a convex region). A symbolic state abstracts a (possibly infinite) set of concrete states. We could define the weak simulation relation in symbolic state space (Space) by employing its counterpart in concrete state space (space).

Given symbolic states S_1 and S_2, we say S_1 simulates S_2 if

$$\forall s_2 \in S_2 \ , \ \exists s_1 \in S_1 \quad s.t. \quad s_1 \succeq s_2 \tag{1}$$

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. In the following, we are going to explore a more general relationship between convex regions that could be used to judge the simulation relation between symbolic states.

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 *prevails* ν', denoted by $\nu \geq \nu'$, if for all i it holds $x_i \geq y_i$. We say a valuation ν is prevailed 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 prevail $\mathcal{C}_2$, denoted as $\mathcal{C}_1 \geq \mathcal{C}_2$ if for all $\nu \models \mathcal{C}_2$, ν is prevailed by $\mathcal{C}_1$. We can see that the prevailing relation is transitive. The convex region inclusion is a sufficient (but not necessary) condition for prevailing relation.

Given two valuations ν and ν' such that ν prevails ν', if we pair them with location names we can obtain two concrete states $s = (l, \nu)$ and $s' = (l', \nu')$. The prevailing relation between ν and ν' implies that $s \succeq_{st} s'$. Similarly, the weak simulation relation between symbolic states can be decided by employing the prevailing relation between two convex regions.

We first extend the slack time pre-order from concrete state space to symbolic state space.

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

Theorem 3. *The pre-order* $\succeq_{st} \subseteq$ Space $\times$ Space *is a weak simulation relation.*

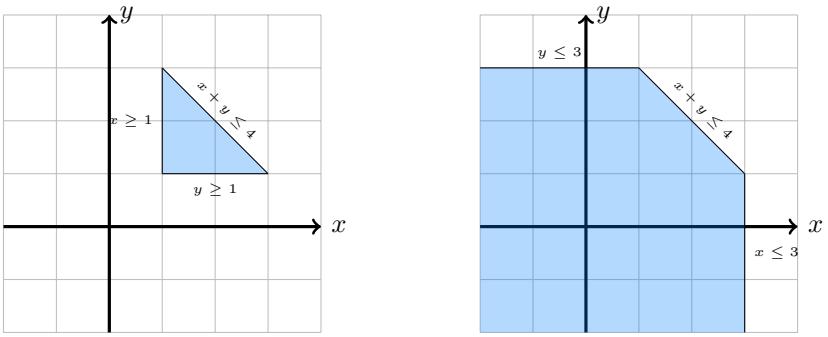
Proof. From the definition of convex region prevailing. □

We now need an efficient method for checking if two convex regions are in a relationship of prevailing. 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

$$(y_1, \ldots, y_N) \models \mathcal{C} \quad \wedge \quad \forall i, \ x_i \leq y_i$$



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

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

- Remove the space dimensions higher than N in $\mathcal{C}'$.

$\nabla(\mathcal{C})$ represents the largest region that is prevailed 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 $\nabla(\mathcal{C}) \geq \mathcal{C}$. An example for the widening operation is shown in Figure 4.

Finally, the prevailing relation between two convex regions, thus the simulation relation between two symbolic states, can be decided by the following lemma.

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 prevailed 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 Optimizing the slack-time pre-order relation

We now present another more efficient simulation relation as an extension of the slack-time pre-order relation $\succeq_{st}$ in case that tasks have constrained deadlines. Although a similar pre-order can also be defined for arbitrary-deadline sporadic tasks, for simplicity we restrict our extension only for tasks with deadlines less than or equivalent to their respective minimum interarrival times.

Let us first have a look at a simple example. Given two states s_1 and s_2 with $s_1.c_i = s_2.c_i = 0$, $s_1.p_i = T_i + 1$, $s_2.p_i = T_i + 10000$ and for any $j \neq i$, there is $s_1.c_j \geq s_2.c_j$ and $s_1.p_j \geq s_2.p_j$. Following the definition of slack-time pre-order, we would see $s_1 \not\succeq s_2$ because of $s_1.p_i < s_2.p_i$. However, for a task with $p_i \geq T_i$ that is ready for releasing a new job, the exact valuation of p_i does not really matter. In the following, we are going to extend the original slack-time pre-order given this observation.

Definition 5 (Extended slack-time pre-order)**.** *For automaton* SA*, the extended* slack-time *pre-order* $\succeq'_{st} \subseteq \mathsf{space} \times \mathsf{space}$ *is defined as follows:* $\forall s_1, s_2$*,* $s_1 \succeq'_{st} s_2$ *if and only if for any* τ_i

$$s_1.p_i \geq s_2.p_i \quad \wedge \quad s_1.c_i \geq s_2.c_i$$

or

$$s_1.p_i > T_i \quad \wedge \quad s_1.c_i \geq s_2.c_i \tag{C1}$$

The extension comes from the condition in (C1): $s_1.c_i \geq s_2.c_i$ is the same as in $\succeq_{st}$; $s_1.p_i > T_i$ says that τ_i is eligible to release a new job in s_1 at any time. The extension part is meaningful when $s_2.p_i > s_1.p_i > T_i$ and in such a case the new job release of τ_i in both s_1 and s_2 can happen at any time regardless of the exact values of $s_1.p_i$ or $s_2.p_i$.

Theorem 4. *The extended pre-order relation* $\succeq'_{st}$ *is still a weak simulation relation in* space(SA)*.*

Proof. Let us say $s_1 \succeq'_{st} s_2$. If $s_2.p_i \leq T_i$, then the original $\succeq_{st}$ pre-order relation simply holds. Otherwise, if there is $s_1.c_i = 0$, then for any job release of τ_i from s_2, τ_i can trigger its job release also in s_1; given the structure of a Task Automaton and the constraint $s_1.p_i > T_i$, there is no possibility that $s_1.c_i > 0$ (suppose that s_1 be not in a DeadlineMissed location). In the end, our proof for Theorem 2 still holds for the extended slack-time pre-order. □

Additionally, we can define the new pre-order for unconstrained-deadline tasks by modifying the condition in (C1) as: $s_1.p_i > D_i \quad \wedge \quad s_1.c_i \geq s_2.c_i$. The reasoning behind this is similar as in proof of Theorem 4 and we are not going into details of it.

As in the case of $\succeq_{st}$, we are going to adapt $\succeq'_{st}$ for symbolic state space of SA. We first extend the widening operator ∇. We define a new widening operator ∇' such that given a convex polyhedron $\mathcal{C}$, $\nabla'(\mathcal{C})$ is constructed as follows.

- For each task τ_i, let us say the dimension x_i (in $\mathcal{C}$) corresponds to its p_i variable. If $x_i > T_i$, then we unconstrain x_i; that is, after the operation there is $-\infty < x_i < +\infty$.
- Suppose $\mathcal{C}'$ be the resulted convex polyhedron after the first step; the original widening operator ∇ is then called on $\mathcal{C}'$. That is, the final widened convex region is $\nabla(\mathcal{C}')$.

Now, let us define the extended slack-time pre-order in the symbolic state space Space(SA) and prove its validity.

Definition 6. *For the* SA *automaton, the extended* slack-time *pre-order* $\succeq'_{st} \subseteq \mathsf{Space} \times \mathsf{Space}$ *is defined such that* $\forall S_1, S_2$*,* $S_1 \succeq'_{st} S_2$ *if and only if* $\nabla'(S_1.\mathcal{C})$ *includes* $\nabla'(S_2.\mathcal{C})$*.*

Theorem 5. *The extended pre-order* $\succeq'_{st} \subseteq \mathsf{Space} \times \mathsf{Space}$ *is a weak simulation relation.*

Proof. After the ∇' operation, $-\infty < p_i < +\infty$ notifies the existence of such tasks that are eligible to release new jobs at any time from then on. As in condition (1), the weak simulation relation between symbolic states can be derived from corresponding weak simulation relation in concrete state space. Given the definition of extended slack-time pre-order $\succeq'_{st}$ in $\mathsf{space(SA)}$, the claim in this theorem is valid. □

In the remainder of this work, we implicitly use the notation slack-time pre-order $\succeq_{st}$ to denote also the extended one.

6.4 Schedulability Analysis in SA

In this section, we formulate the algorithm to explore the state space of SA by using a breadth-first traversal for reachability 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 from 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

$$\forall S \in \mathsf{Max}^{\succeq}(R') : \nexists S' \in R' \ s.t. \ S' \succ_{st} S$$
$$\forall S, S' \in \mathsf{Max}^{\succeq}(R') : S' \not\succeq_{st} S$$

At line 8 of the algorithm, $\mathsf{Max}^{\succeq}$ operation eliminates from R' the states that are simulated by others. When no new state is produced (line 9), the algorithm terminates and the task set is deemed schedulable.

As long as it terminates, the correctness of Algorithm 1 can be proved following the same scheme as in [17]. We skip the details here.

Moreover, we can also define $\mathsf{Max}^{\supseteq}(R')$ as

$$\forall S \in \mathsf{Max}^{\supseteq}(R') : \nexists S' \in R' \ s.t. \ S'.l = S.l \wedge S'.\mathcal{C} \supset S.\mathcal{C}$$
$$\forall S, S' \in \mathsf{Max}^{\supseteq}(R') : \neg(S'.l = S.l \wedge S'.\mathcal{C} \supseteq S.\mathcal{C})$$

This is a common and very basic strategy to eliminate unnecessary states during state space exploration. 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, which is a normal reachability analysis procedure. In Section 8.1, we will compare the efficiency of these two versions of SA-SA, with and without simulation relation. As $\succeq_{st}$ does not require the equivalence of location names between two states such that $s_1 \succeq_{st} s_2$ and convex region inclusion is a special case of convex region prevailing, there is $\mathsf{Max}^{\succeq}(R') \subseteq \mathsf{Max}^{\supseteq}(R')$. We will investigate how much efficiency improvement the schedulability analysis can obtain through simulation relation enhancement.

Unlike 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 the 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 by 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 number of states generated at each step will be exactly the same for the two cases, whereas this may not be true for the method in [5] and [17].

We have implemented SA-SA in the software FOrmal Real-Time Scheduler (FORTS)([27]).

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
```

In general the reachability analysis of LHAs is not decidable. However, specific LHAs may be analysable in a finite number of steps. We now prove that the problem under investigation is indeed decidable.

7 The Decidability Interval

In this section, we formulate the concept of *decidability interval*, a bounded time interval starting from time 0 (that is the beginning of the schedule) such that a set of sporadic tasks is schedulable if and only if there exists no configuration of release patterns that provokes a deadline miss inside this interval. In other words, we must check the schedules generated by all possible patterns of releases, but each one only inside the decidability interval.

We then propose a formula for computing the decidability interval for G-FP scheduling of sporadic tasks with constrained deadlines, and we show that this interval is indeed quite short. Note that the concept of decidability interval is applicable to scheduling of sporadic tasks and does not rely on the system automaton SA.

In the end, based on the decidability interval obtained for G-FP scheduling, we prove the decidability of SA-SA algorithm with respect to constrained-deadline sporadic tasks. In the remainder of this paper, by default we assume that all tasks have constrained deadlines.

7.1 System statuses and the dominance relation

We first introduce some preliminary concepts and properties that will be used for defining and deriving the decidability interval.

The *task status* $\delta_i(t)$ of a task τ_i at time t is defined as a pair $\delta_i(t) = (p_i(t), c_i(t))$.

- $p_i(t)$ is the *activation variable* and records the time passed since τ_i's latest activation.
- $c_i(t)$ is the *execution variable* and represents the unfinished computation of τ_i at time t.

Clearly, $p_i(t)$ and $c_i(t)$ here mimic their respective counterparts (p_i variable and c_i variable) in the task automaton TA. Please refer to Section 5.1 for a more detailed explanation on these variables. At any time t, there could be infinite possibilities of task statuses $\delta_i(t)$, subject to different task release patterns.

The *system status* $\Delta(t)$ at time t for the task set $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ is defined as the composition of all its tasks' statuses, $\Delta(t) = (\delta_1(t), \ldots, \delta_n(t))$. A system status $\Delta(t)$ maps to a concrete state (l, ν) in SA, as all the values of $p_i(t)$ and $c_i(t)$ for each task: 1) on one hand represent the valuation ν in a state, and 2) on the other hand encode corresponding location information l.

Let $\mathcal{T}_k$, with $k \leq n$, be the subset containing the first k tasks from $\mathcal{T}$. We define $\Delta_k(t)$ as the *partial system status* of the tasks in $\mathcal{T}_k$.

We say that a task τ_i is active under the system status $\Delta(t)$ if in the corresponding task status $\delta_i(t)$ there is $c_i(t) > 0$. $\mathsf{A}_{\Delta(t)}$ denotes the set of active tasks under system status $\Delta(t)$. If at time t there is $p_i(t) = D_i$ and $c_i(t) > 0$, then τ_i misses its deadline.

If a task τ_i is released at time t, then task statuses for τ_i before and after the release are also different. Prior to the release, there is $p_i(t) \geq T_i$ and $c_i(t) = 0$; after the release, we have $p_i'(t) = 0$ and $c_i'(t) = C_i$. To make an explicit distinction between the two cases, we use $p_i'(t)$ and $c_i'(t)$ to denote the latter. Such a convention also applies for task status $\delta_i'(t)$ and system status $\Delta'(t)$.

(The initial configuration) We assume $\Delta(0)$ is the initial status of the system. Tasks in the system are released asynchronously and we do not know when the first instance of a task is released. Thus, we configure the initial system status as

$$\forall \tau_i \in \mathcal{T}\ p_i(0) \geq 0 \text{ and } c_i(0) = 0$$

This means that the $p_i(0)$ could be an arbitrary non-negative value.

(The dominance relation) Given two task statuses $\delta_i(t)$ and $\delta_i(t')$, at time t and t' respectively, we say $\delta_i(t)$ *dominates* $\delta_i(t')$, denoted as $\delta_i(t) \succeq \delta_i(t')$, if the following conditions hold.

- $p_i(t) \geq p_i(t')$.
- $c_i(t) \geq c_i(t')$.

The relationship can be lifted to system status: given two system statuses $\Delta(t)$ and $\Delta(t')$, we say $\Delta(t)$ *dominates* $\Delta(t')$, denoted as $\Delta(t) \succeq \Delta(t')$, if $\forall \tau_i \in \mathcal{T}$ there is $\delta_i(t) \succeq \delta_i(t')$. The dominance relation can naturally be applied between partial system statuses.

The dominance relation between system statuses mimics the slack-time preorder relation in system automaton SA but without the underlying LHA model. By understanding this, the following property is straightforward.

Lemma 2. *Assume there are two system statuses $\Delta(t)$ and $\Delta(t')$ at t and t' respectively and $\Delta(t) \succeq \Delta(t')$. Suppose ϵ be an arbitrary non-negative value and $\Delta(t'+\epsilon)$ be a system status at $(t'+\epsilon)$ such that from $\Delta(t')$ to $\Delta(t'+\epsilon)$*

- *there is no task activation;*
- *or, there is exactly one task activation at time $(t'+\epsilon)$.*

Then, there exists a system status $\Delta(t+\epsilon)$ such that $\Delta(t+\epsilon) \succeq \Delta(t'+\epsilon)$.

Proof. We first consider the case that from $\Delta(t')$ to $\Delta(t'+\epsilon)$ there is no new task release. As tasks are activated sporadically, there exists a system status $\Delta(t+\epsilon)$ such that from $\Delta(t)$ to $\Delta(t+\epsilon)$ there is also no new task arrival. In such a case, all activation variables continuously increase. For any task τ_i there is $p_i(t+\epsilon) = p_i(t) + \epsilon$ and $p_i(t'+\epsilon) = p_i(t') + \epsilon$. As $\Delta(t) \succeq \Delta(t')$, for any task τ_i, there is $p_i(t) \geq p_i(t')$ and $c_i(t) \geq c_i(t')$; this further implies that $p_i(t+\epsilon) \geq p_i(t'+\epsilon)$.

Now, it is the turn to analyze the execution variables. If τ_i is not an active task in $\Delta(t')$, as there is no new task arrival, there will be $c_i(t') = c_i(t'+\epsilon) = 0$; thus, $c_i(t+\epsilon) \geq c_i(t'+\epsilon)$ trivially holds. On the other hand, if τ_i is indeed an active task in $\Delta(t')$, then it is also an active task in $\Delta(t)$. Suppose π and π' be the higher-priority interference that τ_i suffers from t to $(t+\epsilon)$ and from t' to $(t'+\epsilon)$ respectively. Since $\Delta(t) \succeq \Delta(t')$, $\mathsf{A}_{\Delta(t)} \supseteq \mathsf{A}_{\Delta(t')}$ and for any task $\tau_j \in \Delta(t')$ its unfinished execution in $\Delta(t)$ is no smaller than its unfinished execution in $\Delta(t')$ as $c_i(t) \geq c_i(t')$. According to the G-FP scheduling policy, π will not be smaller than π', i.e., $\pi \geq \pi'$. Suppose that $c_i(t'+\epsilon) > 0$ (otherwise, $c_i(t+\epsilon) \geq c_i(t'+\epsilon)$ will trivially hold). There is actually $c_i(t'+\epsilon) = c_i(t') - (\epsilon - \pi') = c_i(t) + \pi' - \epsilon$; on the other side, $c_i(t+\epsilon) = c_i(t) + \pi - \epsilon \geq c_i(t'+\epsilon)$. As a result, we have $\Delta(t+\epsilon) \succeq \Delta(t'+\epsilon)$.

Now, let us consider the second case, that is some task τ_k is activated task at time $(t'+\epsilon)$. That is, before τ_k is released, there is $p_k(t'+\epsilon) \geq T_k$; after τ_k is activated, there is $p'_k(t'+\epsilon) = 0$ and $c'_k(t'+\epsilon) = C_k$. From t' (t) to $(t'+\epsilon)$

$((t+\epsilon))$, tasks that are different from τ_k can be analysed in the same way as in the above discussion. Thus, We are going to concentrate on the task τ_k.

As $\Delta(t) \succeq \Delta(t')$, there is $p_k(t) \geq p_k(t')$. From t to $(t+\epsilon)$, if τ_k does not release a new task instance, there is $p_k(t+\epsilon) = p_k(t) + \epsilon \geq p_k(t') + \epsilon \geq T_k$. So, at time $(t+\epsilon)$, a new instance from τ_k is eligible to be released, which will result in $p'_k(t+\epsilon) = 0 = p'_k(t'+\epsilon)$ and $c'_k(t+\epsilon) = C_k = c'_k(t'+\epsilon)$. As a result, $\Delta'(t+\epsilon) \succeq \Delta'(t'+\epsilon)$.

Hence, the lemma is proved. □

Then, we generalise the above Lemma 2 as follows.

Theorem 6. *Suppose $\Delta(t)$ and $\Delta(t')$ are two system statuses such that $\Delta(t) \succeq \Delta(t')$. Then, for any non-negative value Θ and for any system status $\Delta(t'+\Theta)$, there exists a system status $\Delta(t+\Theta)$ such that $\Delta(t+\Theta) \succeq \Delta(t'+\Theta)$.*

Proof. From $\Delta(t')$ to $\Delta(t'+\Theta)$, the time interval $[t', t'+\Theta]$ can be divided into a finite number (let us say N) of successive sub-intervals $[t', t'+\epsilon_1], [t'+\epsilon_1, t'+\epsilon_2], \ldots, [t'+\epsilon_{N-1}, t'+\Theta]$ such that in each such sub-interval there is no new task arrival or there is exact one new task arrival and it happens at the end of the sub-interval. Note that in case there are multiple task arrivals at a single time point, we can build multiple sub-intervals with length 0. Then, Lemma 2 can be applied successively to each sub-interval and the Theorem is proved. □

7.2 The decidability interval for G-FP scheduling

First, we formally define the decidability interval for a set of sporadic tasks under G-FP scheduling as follows.

Definition 7. ***(Decidability Interval)*** *The decidability interval is defined as a time interval $[0, L]$ such that the schedulability of task set $\mathcal{T}$ on m processors under a G-FP scheduler can be decided inside it. That is, the task set $\mathcal{T}$ is schedulable if and only if no task will miss its deadline in the interval $[0, L]$.*

In order to compute the decidability interval for G-FP scheduling, we need to introduce another concept called *dominant interval*.

Definition 8. ***(Level-k Dominant Interval)*** *For the task subset $\mathcal{T}_k \subseteq \mathcal{T}$, its level-$k$ dominant interval is defined as a time interval $[0, L_k]$ such that for any t' and for any system status $\Delta_k(t')$, there exists $t \in [0, L_k]$ and there exists a system status $\Delta_k(t)$ such that $\Delta_k(t) \succeq \Delta_k(t')$.*

A level-k dominant interval must be a decidability interval for $\mathcal{T}_k$. Suppose that a task $\tau_i \in \mathcal{T}_k$ misses its deadline at time t' under system status $\Delta_k(t')$, i.e., $p_i(t') = D_i$ and $c_i(t') > 0$. Given the definition of a level-k dominant interval, there exists $\Delta_k(t)$ with $t \in [0, L_k]$ such that $\Delta_k(t) \succeq \Delta_k(t')$. That is, $p_i(t) \geq p_i(t')$ and $c_i(t) \geq c_i(t')$; this implies that τ_i also misses its deadline at time t. The decidability interval and level-k dominant interval can be bridged through the following two theorems.

Theorem 7. *Let $[0, L_k]$ be the level-k dominant interval for $\mathcal{T}_k$. Then, $[0, L_k + D_{k+1}]$ is the decidability interval for $\mathcal{T}_{k+1}$.*

Proof. Task τ_{k+1} cannot interfere the execution of higher-priority ones from $\mathcal{T}_k$, and $\mathcal{T}_k$ is schedulable if and only if no task from it misses a deadline in the interval $[0, L_k] \subset [0, L_k + D_{k+1}]$.

As for τ_{k+1}, we will show that if there is a deadline miss for it after the time point $(L_k + D_{k+1})$, there exists also a deadline miss for τ_{k+1} inside the interval $[0, L_k + D_{k+1}]$.

Since L_k is the level-k dominant interval, for any time $t' > L_k$ and for any $\Delta_k(t')$, there exists time $t \in [0, L_k]$ and there exists $\Delta_k(t)$ such that $\Delta_k(t) \succeq \Delta_k(t')$. Suppose that task τ_{k+1} is released at an arbitrary time $t' > L_k$, after which there is $p'_{k+1}(t') = 0$ and $c'_{k+1}(t') = C_{k+1}$. If we consider the time $t \in [0, L_k]$ with $\Delta_k(t) \succeq \Delta_k(t')$, as tasks are sporadically activated and at the initial time the activation variable can have any value, then there exists a situation in which $p_{k+1}(t) \geq T_{k+1}$; this means the task τ_{k+1} can also be activated at time t. After τ_{k+1} is released, there will be also $p'_{k+1}(t) = 0$ and $c'_{k+1}(t) = C_{k+1}$.

Thus, for any release of τ_{k+1} at any $t' > L_k$, there also exists the release for τ_{k+1} at some time $t \in [0, L_k]$ such that after the release $\Delta'_{k+1}(t) \succeq \Delta'_{k+1}(t')$. Suppose that there exists $\Delta(t' + D_{m+1})$ such that τ_{k+1} misses its deadline, i.e., $p_{k+1}(t' + D_{k+1}) = D_{k+1}$ and $c_{k+1}(t' + D_{k+1}) > 0$; according to Theorem 6, there also exists $\Delta_{k+1}(t+D_{k+1})$ with $\Delta_{k+1}(t+D_{k+1}) \succeq \Delta_{k+1}(t'+D_{k+1})$, which implies $p_{k+1}(t + D_{k+1}) = D_{k+1}$ and $c_{k+1}(t + D_{k+1}) \geq c_{k+1}(t' + D_{k+1}) > 0$, i.e., τ_{k+1} also misses its deadline at time $(t + D_{k+1})$.

For any release of τ_k at time $[0, L_k]$, its absolute deadline is bounded by $(L_k + D_{k+1})$. That is, the task τ_{k+1} is schedulable if and only if there is no deadline miss in the interval $[0, L_k + D_{k+1}]$. In conclusion, $[0, L_k + D_{k+1}]$ is the decidability interval for $\mathcal{T}_{k+1}$. □

The above Theorem 7 demonstrates how to derive a decidability interval from a dominant interval. On the contrary, the next theorem shows that a decidability interval can itself be also a dominant interval as long as the following property is verified.

Theorem 8. *The decidability interval $[0, L_k + D_{k+1}]$ in Theorem 7 is a level-$(k+1)$ dominant interval if τ_{k+1} is schedulable.*

Proof. We are going to discuss two cases given the task status $\delta_{k+1}(t')$ of τ_{k+1} at time $t' > L_k + D_{k+1}$: $c_{k+1}(t') = 0$ or $c_{k+1}(t') > 0$.

(Case 1: $c_{k+1}(t') = 0$) Because that L_k is the level-k dominant interval, there exists $t \in [0, L_k]$ and $\Delta_k(t)$ such that $\Delta_k(t) \succeq \Delta_k(t')$. As for τ_{k+1}, given the initial configuration and the sporadic activation behaviour, there exists task status such that $p_{k+1}(t) \geq p_{k+1}(t')$ and $c_{k+1}(t) = 0 = c_{k+1}(t')$. As a result, $\Delta(t)_{k+1} \succeq \Delta_{k+1}(t')$.

(Case 2: $c_{k+1}(t') > 0$) For such a system status $\Delta_{k+1}(t')$ with $c_{k+1}(t') > 0$, the corresponding τ_{k+1} is released at time $(t' - p_{k+1}(t'))$, denoted as t'_0. Before the release, there must be $p_{k+1}(t'_0) \geq T_{k+1}$; and after the release, there

is $p'_{k+1}(t'_0) = 0$ and $c'_{k+1}(t'_0) = C_{k+1}$. Then, there exists $t_0 \in [0, L_k]$ such that $\Delta_k(t_0) \succeq \Delta_k(t'_0)$ and $p_{k+1}(t_0) \geq p_{k+1}(t'_0)$ and $c_{k+1}(t_0) = 0$; this implies, by triggering the release of τ_{k+1}, $\Delta'_{k+1}(t_0) \succeq \Delta'_{k+1}(t'_0)$. According to the Theorem 6, after $p_{k+1}(t')$ time elapsing, there must exist system status $\Delta_{k+1}(t_0 + p_{k+1}(t')) \geq \Delta_{k+1}(t')$. Let us denote with $t = t_0 + p_{k+1}(t')$. As τ_{k+1} is schedulable and $c_{k+1} > 0$, there is $p_{k+1}(t') < D_{k+1}$; together with $t_0 \in [0, L_k]$, we have $t \in [0, L_k + D_{k+1})$.

In conclusion, $\forall t' > L_k + D_{k+1} \forall \Delta_{k+1}(t')$, there exists $t \in [0, L_k + D_{k+1}]$ and $\Delta_{k+1}(t)$ such that $\Delta_{k+1}(t) \succeq \Delta_{k+1}(t')$. That is, $[0, L_k + D_{k+1}]$ is a level-$(k+1)$ dominant interval. □

Theorem 7 and Theorem 8 provide a recursive way to construct the decidability interval for a given task set $\mathcal{T}$. However, the starting point of such a procedure is still missing. The following theorem fills in the gap.

Lemma 3. *The level-m dominant interval is* $[0, L_m]$ *with* $L_m = \max_{\tau_i \in \mathcal{T}_m} \{C_i\}$.

Proof. Suppose that $m = 2$ and $\mathcal{T}_2 = \{\tau_1, \tau_2\}$.

Given any time point $t' > L_2$, the system status $\Delta(t')$ is in one of the following situations.

- Both τ_1 and τ_2 are active such that $0 \leq p_1(t') < C_1$, $c_1(t') = C_1 - p_1(t')$ and $0 \leq p_2(t') < C_2$, $c_2(t') = C_2 - p_2(t')$.
- τ_1 is active and τ_2 is not active such that $0 \leq p_1(t') < C_1$, $c_1(t') = C_1 - p_1(t')$ and $p_2(t') \geq 0$, $c_2(t') = 0$.
- τ_1 is not active and τ_2 is active such that $p_1(t') \geq 0$, $c_1(t') = 0$ and $0 \leq p_2(t') < C_2$, $c_2(t') = C_2 - p_2(t')$.
- Both τ_1 and τ_2 are not active such that $p_1(t') \geq 0$, $c_1(t') = 0$ and $p_2(t') \geq 0$, $c_2(t') = 0$.

For any of the cases listed above, it is not difficult to find a time point $t \in [0, L_m]$ with a configuration $\Delta(t)$ such that $\Delta(t) \succeq \Delta(t')$. For example, in case both τ_1 and τ_2 are active, the corresponding scenario is depicted in Figure 5, where τ_1 and τ_2 are released at time r_1 and r_2 respectively. At time t', there is $c_1(t') = C_1 - t' + r_1$, $p_1(t') = t' - r_1$, $c_2(t') = C_2 - t' + r_2$, $p_2(t') = t' - r_2$. Note that $p_1(t') = t' - r_1$ and $p_2(t') = t' - r_2$ are upper bounded by $L_m = \max_{\tau_i \in \mathcal{T}_m} \{C_i\}$. Then, let us consider the time $t \in [0, L_m]$ such that $t = \max\{t' - r_1, t' - r_2\}$. There exists the task release of τ_1 at time $(t - p_1(t'))$ and the task release of τ_2 at $(t - p_2(t'))$ such that at time t we will see $p_1(t) = p_1(t')$ and $p_2(t) = p_2(t')$. This implies that $c_1(t) = C_1 - p_1(t) = C_1 - p_1(t') = c_1(t')$ and $c_1(t) = C_1 - p_1(t) = C_1 - p_1(t') = c_1(t')$. As a result $\Delta(t) \succeq \Delta(t')$.

In case there is an active task (let us say τ_1) and an inactive task (let us say τ_2, i.e., $p_2(t') \geq 0$ and $c_2(t') = 0$) at time t', for the active task the above analysis still can be applied, and the time $t \in [0, L_m]$ can be found such that $c_1(t) = c_1(t')$ and $p_1(t) = p_1(t')$. For the inactive task τ_2, we remind that, at

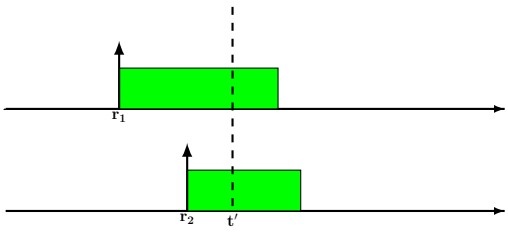


Figure 5: Both τ_1 and τ_2 are active

time 0, $c_2(0) = 0$ and $p_2(0)$ can be any non-negative value; when there is no new task arrival, an activation variable will monotonically increase. This means that at time t, there must exist a task status $\delta_2(t)$ for τ_2 with $p_2(t) \geq p_2(t')$ and $c_2(t) = 0$. As a result, $\Delta(t) \succeq \Delta(t')$.

In the last case, both tasks are inactive. There exists $\Delta(0)$ such that $\Delta(0) \succeq \Delta(t')$. Moreover, if we consider a general $m \geq 2$, it is simply a matter of expanding the list for combinations of active and inactive tasks. □

Starting from the level-m dominant interval and by repeatedly applying Theorems 7 and 8, we are able to compute the decidability interval for any task set $\mathcal{T}$.

Theorem 9. *For a task set $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ with n tasks running on m processors, its decidability interval is $[0, L]$ with $L = \sum_{1 \leq i \leq m} C_i + \sum_{m < i \leq n} D_i$.*

Proof. Suppose that $\mathcal{T}$ is not schedulable. Let us say τ_k is the highest-priority task among all non-schedulable tasks. According to Theorem 7, Theorem 8 and Lemma 3, the decidability interval for $\mathcal{T}_k$ is $[0, \sum_{1 \leq i \leq m} C_i + \sum_{m < i \leq k} D_i]$, which is contained in $[0, L]$. This implies that if $\mathcal{T}$ is not schedulable, then there exists necessarily a configuration of releases such that a deadline miss happens in the interval $[0, L]$. Thus, $[0, L]$ is the decidability interval for $\mathcal{T}$. □

Given the fact that a task will never have its job run for more than its WCRT, we could refine the formulation of a decidability interval.

Lemma 4. *For a task set $\mathcal{T} = \{\tau_1, \ldots, \tau_n\}$ with n tasks running on m processors, its decidability interval can be further refined as $[0, L]$ with $L = \sum_{1 \leq i \leq m} C_i + \sum_{m < i \leq n} \min\{R_i, D_i\}$.*

Proof. By replacing D_i with $\min\{R_i, D_i\}$, the proofs for Theorem 7 and Theorem 8 still hold. In the end, the new decidability interval length is valid. □

Discussion A concept called *feasibility interval* has been extensively studied in [13], [14], and [15] for multiprocessor scheduling of periodic tasks. For a set of periodic tasks, its feasibility interval is a finite interval such that if all

jobs released within it can meet their deadlines, then the system is schedulable. Such a result benefits from the determinism of tasks' periodic activations and its complexity is sensitive to the tasks' hyperperiod. The hyperperiod for a task set is defined as the *least common multiple* among all tasks' periods.

On the other hand, the concept of feasibility interval cannot be applied to multiprocessor scheduling of sporadic tasks. In fact, we are not aware of any work for bounding a time interval for exact multiprocessor schedulability analysis of sporadic tasks.

The decidability interval proposed in this work is the first result that proposes a *small* bounded interval to check the schedulability of a set of sporadic tasks in global multiprocessor scheduling. Moreover, as shown in Theorem 9, the computed decidability interval for a task set could be much shorter than its counterpart for multiprocessor periodic tasks, as it does not rely on the value of hyperperiod.

7.3 Decidability for SA-SA algorithm

Now, we are going to answer the decidability question (in the end of Section 6.4) for the SA-SA algorithm. Thanks to the decidability interval computed by Theorem 9, we know that in case of a set of constrained-deadline tasks, it is enough to perform the reachability analysis of DeadlinMissed location in the bounded time interval $[0, L]$.

The reachability analysis of a generic LHA is undecidable ([1]), even in bounded time ([10]). However, [10] proposed a subclass of LHA, subject to well defined language restrictions, for which the reachability problem in bounded time is decidable. More specifically, clock variables must all be monotonically increasing or stopped. The class of Stopwatch Timed Automata falls in this category.

Since our model is equivalent to a Stopwatch Timed Automaton (with an appropriate transformation of variables), it can be seen that the reachability problem of SA in bounded time is also decidable.

Theorem 10. *The termination of Algorithm 1 (SA-SA) is guaranteed in case of sporadic tasks with constrained deadlines.*

Proof. It follows from Theorem 9 in our work and from Theorem 1 in [10]. □

8 Experiments

In this section we evaluate the runtime performance of SA-SA by applying the algorithm to randomly generated schedulability problems. Each task set in the experiment is characterised by a tuple (m, n, U), where $m = 2$ is the number of processors, $n \in \{5, 6\}$ 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 the number of tasks n and the total task set utilisation U, the utilisation of each task is generated according to Randfixedsum algorithm ([16]). Task periods are distributed in

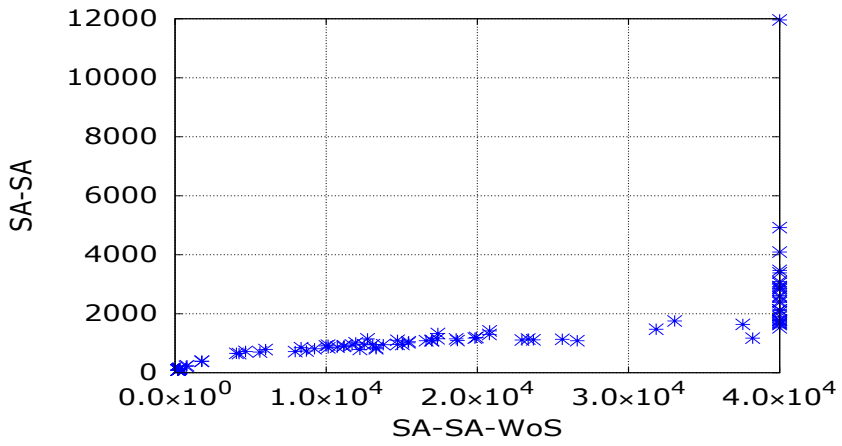


Figure 6: SA-SA v.s. SA-SA-WoS

the range $[100, 1000]$. After selecting a task period T_i, the WCET is computed as $C_i = T_i \cdot U_i$; the relative deadline is then randomly sampled from C_i to T_i, After a task set is randomly generated, priorities are assigned to its tasks by the Deadline Monotonic strategy; that is, a task with shorter deadline is assigned higher priority.

Discussion It is important to underline the advantages of our methodology with respect to the methods proposed in [5], [17], [9]. These methods typically assume task parameters such as WCETs, deadlines and periods to be rather small integers. For example, for testing their method, [17] restricted tasks to have period no larger than 8. This is mainly due to the fact that they use discrete time model checking, with integer time values. Thus, these methods are sensitive to the absolute values of task parameters. On the other hand, relying on the formalism of LHA in continuous time domain, we are able to apply exact schedulability test on more general task configurations.

8.1 SA-SA algorithm with and without slack-time pre-order relation

First, we demonstrate how much the slack-time pre-order ($\succeq_{st}$) can benefit the reachability analysis in system automaton SA, by comparing the SA-SA algorithm and the SA-SA Without weak Simulation (SA-SA-WoS). We randomly generate 100 task sets with $m = 2$, $n = 5$ and utilisation randomly chosen in $[0.5, 2.0]$. Then we apply SA-SA and SA-SA-WoS to each task set, and we respectively record state space size for the two to decide the schedulability of

every task set. Without the enhancement of slack-time pre-order, SA-SA-WoS faces the danger of going out of memory for some task sets. So, we put an upper threshold of 40000 for the symbolic state space size during the test of a task set by SA-SA-WoS. When the number of generated states exceeds this threshold, we stop the analysis and return with error.

Results are reported in Figure 6. Each point in the graphic represents a task set: the x-axis value records the number of states generated by SA-SA-WoS for checking the schedulability, and the y-axis is the state space size generated by SA-SA. As shown in the figure, by employing the slack-time weak simulation relation $\succeq_{st}$, the size of state space for schedulability check is reduced significantly. In most cases, state space sizes resulted by SA-SA and SA-SA-WoS can differ by an order of magnitude.

8.2 Runtime complexity of SA-SA algorithm

In Theorem 10, we have proved the termination of schedulability analysis by SA-SA. In this part, we will evaluate the factors that could affect SA-SA's runtime performance.

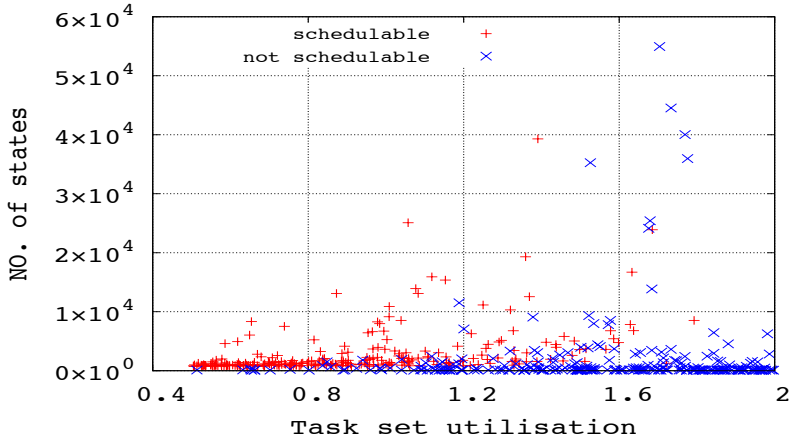
All simulations are conducted in a MacBook with 2.5 GHz Intel Core i5 and 8 GB memory. At first, we consider 2 processors ($n = 2$) and 5 tasks ($m = 5$). More specifically, we randomly generate a series of task sets with U uniformly distributed within $[0.5, 2]$.

By applying SA-SA on these task sets, we record the time consumed and final state space size for each test.

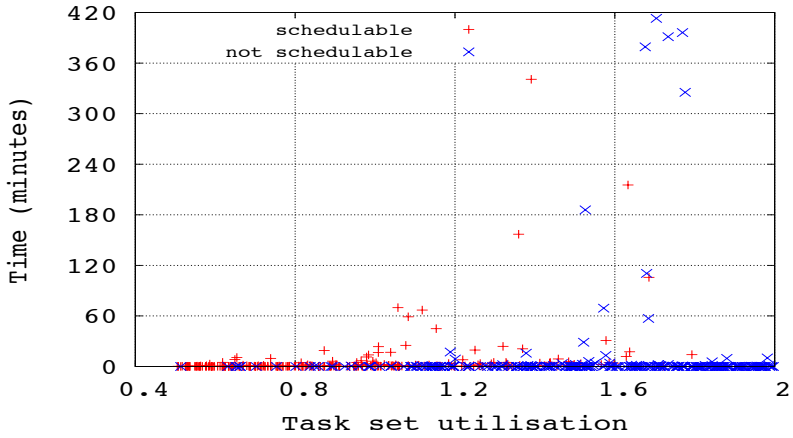
The results are shown in Figure 7a that displays the state space size of all tests by SA-SA. Each point in the figure is a task set, which is further distinguished by being schedulable and not schedulable. The x-axis denotes the utilisation of the task set and the y-axis counts the final state space size by SA-SA to decide its schedulability. Similarly, Figure 7b reports the time cost (in minutes) to decide a task set's schedulability and Figure 7c shows the relation between state space size and time cost.

As we can observe, most tests terminate in a short time with a relatively small state space size. However, with an increase of the task set utilisation, we may experience cases with a rather high runtime cost, with respect to states generated and time spent. This is due to the fact that a higher task set utilisation has a larger chance to result in a longer decidability interval (Lemma 4), within which there could be more complex task execution interleavings, thus complicating SA-SA's reachability analysis.

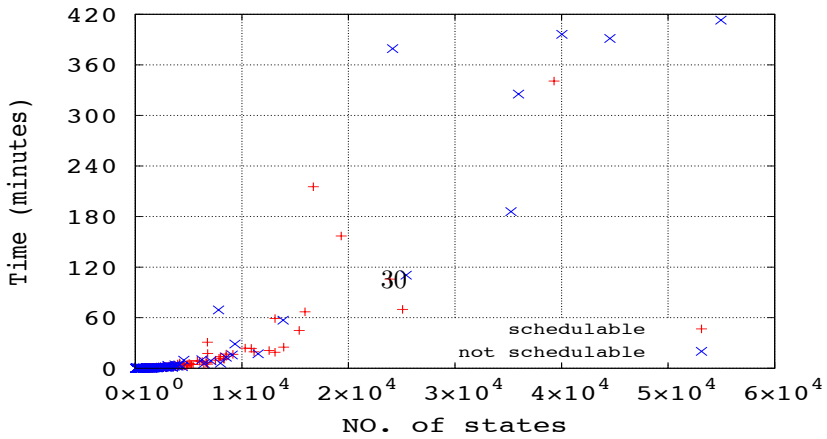
Furthermore, we run simulations with $m = 2$ and $n = 6$. This time, we fix several different utilisation levels for generating task sets. The time spent and state space size on each task set by SA-SA are plotted in Figure 8. Note that we manually stop the procedure when analysis time exceeds 7 hours (upper bound on the maximum time cost we experienced for 5 tasks running on 2 processors), and we use black colors to denote these cases in Figure 8a. In the same figure, we use red colors and blue colors for schedulable and unschedulable task sets respectively. Still, a task set with higher utilisation tends to complicate the



(a) State space size



(b) Time cost



(c) State space size v.s. Time cost

analysis by SA-SA, and this is compatible with our observation from $m = 2$ and $n = 5$. On the other side, we must understand that for the unschedulable task set, SA-SA may terminate without exploring the complete state space, as long as the corresponding deadline miss condition is encountered. Thus, the schedulability check of an unschedulable task set may finish very soon. When it comes to the simulation results, as the total utilisation keeps increasing, there will more unschedulable task sets. In fact, the average runtime performance (time cost and state space size) of SA-SA improves in the high utilisation end of our simulations, as shown in Figure 8b and Figure 8c.

8.3 Comparison with state-of-the-art over-approximate approach

In this part, we assess the pessimism of state-of-the-art over-approximate schedulability test for sporadic tasks under G-FP policy. The analytic test we use is the Response Time Analysis with Carry-in Enumeration (RTA-CE) proposed in [30]. Although being the most accurate over-approximate schedulability test for G-FP scheduling problem to date, RTA-CE is still pessimistic. That is, RTA-CE may judge a schedulable task set as not-schedulable. It is interesting and meaningful to see how much gap there is between the approximate result of RTA-CE and the exact result of SA-SA.

We pick up the task set utilisation U from the set $\{0.5, 0.6 \ldots, 1.5, 1.6\}$ with $m = 2$ and $n = 5$. Then, for each U we generate 100 task sets, for which RTA-CE and SA-SA are applied. The number of schedulable tasks discovered by RTA-CE and SA-SA on each utilisation level are recorded respectively. Results are plotted in Figure 9. The x-axis shows the utilisation and the y-axis represents the percentage of schedulable tasks over the total number of randomly sampled tasks (at each utilisation level).

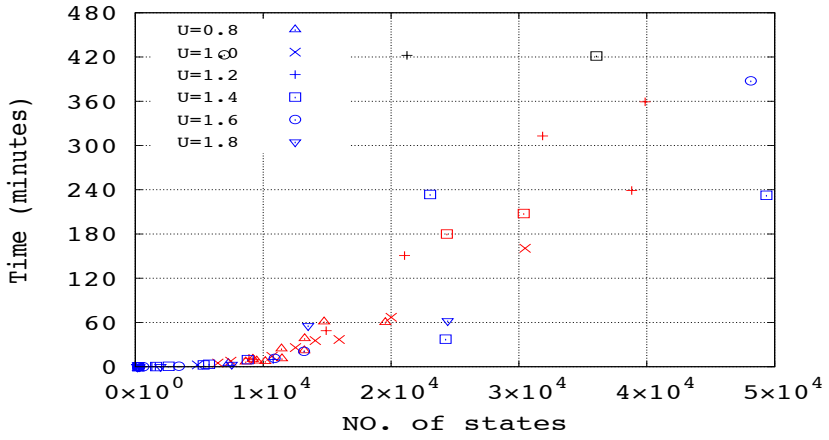
As shown in Figure 9, there is still a considerable number of schedulable task sets that RTA-CE failed to find. Such a gap could be seen as a motivation to further develop more precise approximate schedulability analysis.

8.4 Exact schedulability analysis for periodic tasks in G-FP

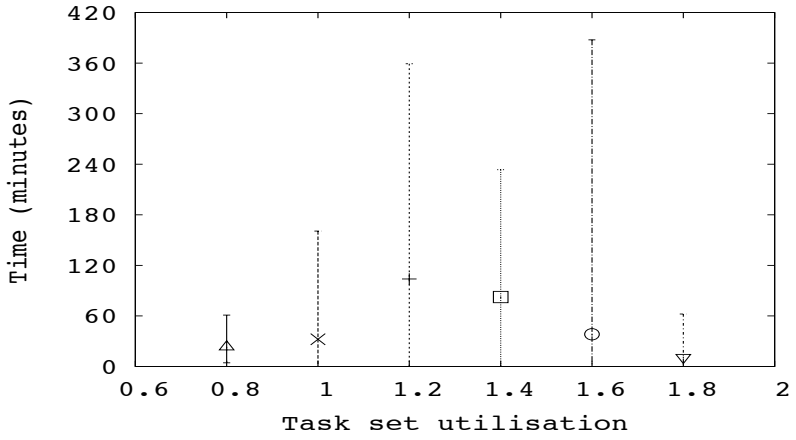
Finally, we would like to discuss the difference in complexity between exact analysis of *periodic tasks* and *sporadic tasks*.

In the case of single processor, for the fixed-priority scheduling, the schedulability analyses of sporadic tasks and synchronous periodic tasks are not really different, as they share the same worst-case scenario. When it comes to multiprocessor, no worst-case scenario has even been found for sporadic or periodic tasks under G-FP. Therefore, to perform an exact analysis we need to analyse all possible task activations and interleavings, for both sporadic and periodic tasks.

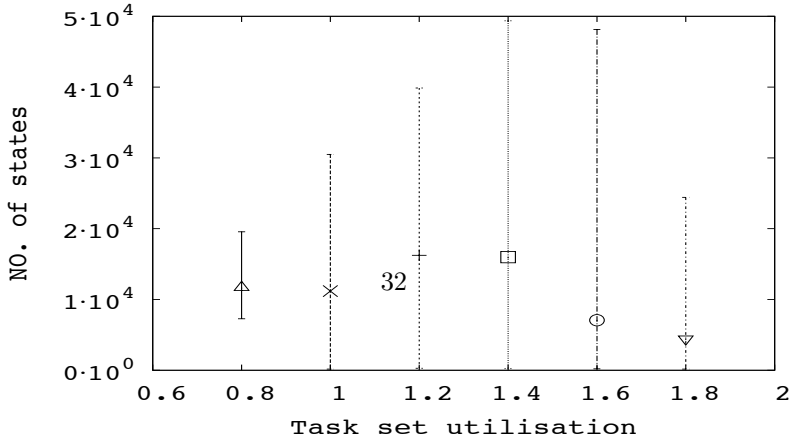
As we proved in Theorem 9, the exact analysis of sporadic tasks can be done in a decidability interval, which is relatively small. Therefore, the complexity



(a) State space size v.s. Time cost



(b) Average time cost



(c) Average state space size

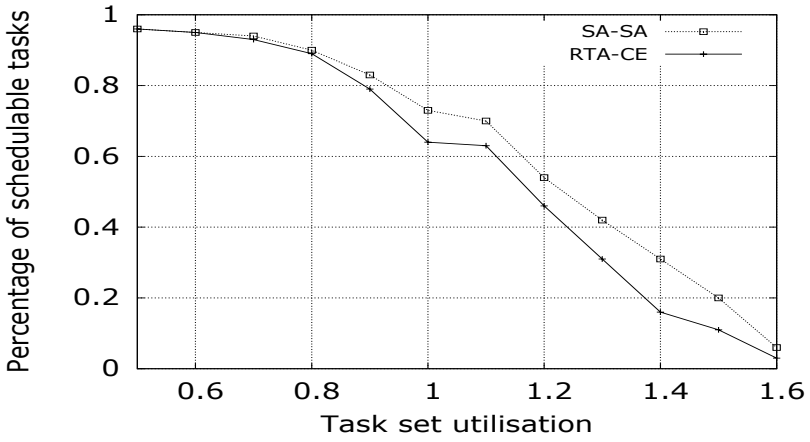


Figure 9: Comparison between RTA-CE and SA-SA

of analysing sporadic tasks comes from the non-deterministic activations, which produce many different arrival patterns that need to be analysed. As for periodic tasks, they have deterministic activation patterns, and the complexity of an exact analysis is affected by the length of the feasibility interval, which is directly related to the hyperperiod of the tasks.

In this section, we refer to [20] and [21] for the exact test for periodic tasks. Both works adopt the discrete approach to model the schedule of a set of periodic tasks running on multiprocessor. Even though [20] builds the model by Timed Automata, the continuous time is discretised by using a global clock. Thus, they restrict the absolute values of task parameters to small integers. Actually, in both cases the authors choose task periods in a range $[8, 20]$ for their experiments.

In principle, the analysis scheme for periodic tasks considers all possible combinations of tasks' initial offsets; for an arbitrary combination of initial offsets, tasks' periodic activations are fixed, the system is simulated and any deadline miss is checked.

In the following, we replicate the experiment in [21], where there are 8 tasks running on 4 processors. Our goal here is not to solve the exact schedulability analysis of periodic tasks, but rather to demonstrate the different runtime complexity. Therefore, we consider all offsets to be equal to 0, so all tasks arrive at the same time.

We randomly generate 1000 periodic task sets. The system automaton for schedulability check for periodic tasks can be generated similarly as the SA for sporadic tasks: in Figure 2, the guard on the transition from Idle to Waiting is set to $p = T$. We also restrict ourselves to tasks with constrained deadline (i.e.

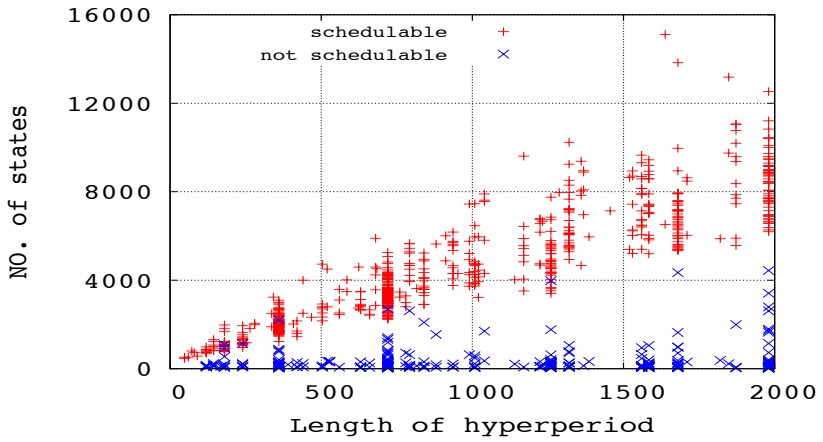


Figure 10: The exact analysis for periodic tasks

$D \leq T$), so we remove all guards and actions from the transitions from Waiting to itself, and from Running to itself. Finally, we do not use our simulation relation, because it is only valid for sporadic tasks and cannot be easily extended to periodic tasks.

In the end, we use FORTS to check the schedulability of each task set and records the time cost for deciding the schedulability. Results are reported in Figure 10. The x-axis denotes the length of the hyperperiod of a task set and the y-axis denotes the size of the state space when schedulability check is completed for the task set. As we anticipated, the number of states to explore increases with the length of hyperperiod. The maximum time we experienced for deciding a task set's schedulability is less than 3 minutes. It follows that, deciding the schedulability of a set of periodic tasks is in average *easier* than checking the schedulability of a set of sporadic tasks.

9 Conclusions and future work

In this paper, we formally model the exact G-FP scheduling on a multiprocessor platform; we propose a weak simulation relation for reducing the complexity of state space exploration. Then, we formulate SA-SA algorithm for the exact schedulability test. Even though the reachability problem in LHA is undecidable, we prove the correctness and termination condition for SA-SA. Compared to previous works on exact analysis, our methodology allows more complex task sets: we are able to analyse tasks with arbitrary values of parameters. On the other hand, our simulation relation work can be regarded as a general approach

to mitigate the complexity brought by sporadic events when modeling real-time systems with a formalism like Stopwatch Timed Automata.

However, as task set size increases, the complexity of exact analysis is still too high even with our approach. We are working on other ways of reducing the complexity: first, we would like to use a different representation for the symbolic states (e.g. Octagons ([26]) or Difference Bound Matrices ([25])), which requires an approximate analysis similar to the ones used for Stopwatch Timed Automata and Time Petri Nets ([11]). Second, we are investigating the possibility to enhance and extend our simulation relation, so to further reduce the state space.

Furthermore, although exact critical instants for G-FP scheduling are not known, [30] proved a class of release patterns that could lead to worst-case response time. By exploring the advanced study in scheduling theory, we may achieve simpler models or even faster state space analysis methods.

Finally, we proved that the schedulability analysis of a sporadic task set could be done in bounded time. To our knowledge, this is the first paper to demonstrate the decidability of the problem and to formulate such an upper bound. It will be interesting to see if we can apply this result to further enhance the approximate schedulability analysis methods.

References

[1] Alur R, Courcoubetis C, Henzinger TA, Ho PH (1993) Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. Springer

[2] Alur R, Courcoubetis C, Halbwachs N, Henzinger TA, Ho PH, Nicollin X, Olivero A, Sifakis J, Yovine S (1995) The algorithmic analysis of hybrid systems. Theoretical computer science 138(1):3–34

[3] Baker T (2003) Multiprocessor EDF and deadline monotonic schedulability analysis. IEEE Real-Time Systems Symposium (RTSS)

[4] Baker T, Baruah SK (2009) Sustainable multiprocessor scheduling of sporadic task systems. In: Real-Time Systems, 2009. ECRTS'09. 21st Euromicro Conference on, IEEE, pp 141–150

[5] Baker T, Cirinei M (2007) Brute-Force Determination of Multiprocessor Schedulability for Sets of Sporadic Hard-Deadline Tasks. In: Tovar E, Tsigas P, Fouchal H (eds) Principles of Distributed Systems, Lecture Notes in Computer Science, vol 4878, Springer Berlin Heidelberg, p 62–75

[6] Baruah S (2007) Techniques for Multiprocessor Global Schedulability Analysis. In: Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International, pp 119–128, DOI 10.1109/RTSS.2007.35

[7] Bengtsson J, Yi W (2004) Timed Automata: Semantics, Algorithms and Tools. In: , Springer-Verlag, pp 87–124

[8] Bertogna M, Cirinei M (2007) Response-time analysis for globally scheduled symmetric multiprocessor platforms. In: Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International, IEEE, pp 149–160

[9] Bonifaci V, Marchetti-Spaccamela A (2012) Feasibility analysis of sporadic real-time multiprocessor task systems. Algorithmica 63(4):763–780

[10] Brihaye T, Doyen L, Geeraerts G, Ouaknine J, Raskin JF, Worrell J (2011) On reachability for hybrid automata over bounded time. In: Automata, Languages and Programming, Springer, pp 416–427

[11] Bucci G, Fedeli A, Sassoli L, Vicario E (2004) Timed state space analysis of real-time preemptive systems. Software Engineering, IEEE Transactions on 30(2):97–111

[12] Cassez F, Larsen K (2000) The Impressive Power of Stopwatches. In: Proc. of CONCUR 2000: Concurrency Theory, Springer, pp 138–152

[13] Cucu L, Goossens J (2006) Feasibility intervals for fixed-priority real-time scheduling on uniform multiprocessors. In: Emerging Technologies and Factory Automation, 2006. ETFA'06. IEEE Conference on, IEEE, pp 397–404

[14] Cucu L, Goossens J (2007) Feasibility intervals for multiprocessor fixed-priority scheduling of arbitrary deadline periodic systems. In: Proceedings of the conference on Design, automation and test in Europe, EDA Consortium, pp 1635–1640

[15] Cucu-Grosjean L, Goossens J (2011) Exact schedulability tests for real-time scheduling of periodic tasks on unrelated multiprocessor platforms. Journal of systems architecture 57(5):561–569

[16] Emberson P, Stafford R, Davis RI (2010) Techniques for the synthesis of multiprocessor tasksets. In: 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS), pp 6–11

[17] Geeraerts G, Goossens J, Lindstrõm M (2013) Multiprocessor schedulability of arbitrary-deadline sporadic tasks: complexity and antichain algorithm. Real-Time Systems 49(2):171–218, DOI 10.1007/s11241-012-9172-y, URL `http://dx.doi.org/10.1007/s11241-012-9172-y`

[18] Geeraerts G, Goossens J, Stainer A (2014) Synthesising succinct strategies in safety and reachability games. In: Reachability Problems - 8th International Workshop, RP 2014, Oxford, UK, September 22-24, 2014. Proceedings, pp 98–111

[19] Grolleau E, Goossens J, Cucu-Grosjean L (2013) On the periodic behavior of real-time schedulers on identical multiprocessor platforms. arXiv preprint arXiv:13053849

[20] Guan N, Gu Z, Deng Q, Gao S, Yu G (2007) Exact schedulability analysis for static-priority global multiprocessor scheduling using model-checking. In: Software Technologies for Embedded and Ubiquitous Systems, Springer, pp 263–272

[21] Guan N, Gu Z, Lv M, Deng Q, Yu G (2008) Schedulability analysis of global fixed-priority or EDF multiprocessor scheduling with symbolic model-checking. In: Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on, IEEE, pp 556–560

[22] Guan N, Stigge M, Yi W, Yu G (2009) New response time bounds for fixed priority multiprocessor scheduling. In: Real-Time Systems Symposium, 2009, RTSS 2009. 30th IEEE, IEEE, pp 387–397

[23] Halbwachs N, Proy YE, Roumanoff P (1997) Verification of real-time systems using linear relation analysis. Formal Methods in System Design 11(2):157–185

[24] Liu C, Layland J (1973) Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. Journal of the Association for Computing Machinery 20(1):46–61

[25] Miné A (2001) A New Numerical Abstract Domain Based on Difference-Bound Matrices. In: Proceedings of the Second Symposium on Programs as Data Objects, Springer-Verlag, London, UK, UK, PADO '01, pp 155–172, URL http://dl.acm.org/citation.cfm?id=645774.668110

[26] Miné A (2006) The octagon abstract domain. Higher-Order and Symbolic Computation 19(1):31–100, DOI 10.1007/s10990-006-8609-1, URL http://dx.doi.org/10.1007/s10990-006-8609-1

[27] Sun Y, Lipari G (2014) FOrmal Real-Time Scheduler (FORTS). Web page: https://github.com/glipari/forts

[28] Sun Y, Lipari G (2014) A weak simulation relation for real-time schedulability analysis of global fixed priority scheduling using linear hybrid automata. In: Proceedings of the 22nd International Conference on Real-Time Networks and Systems, ACM, p 35

[29] Sun Y, Lipari G, André É, Fribourg L (2014) Toward parametric timed interfaces for real-time components. In: Proceedings 1st International Workshop on Synthesis of Continuous Parameters, SynCoP 2014, Grenoble, France, 6th April 2014., pp 49–64, DOI 10.4204/EPTCS.145.6, URL http://dx.doi.org/10.4204/EPTCS.145.6

[30] Sun Y, Lipari G, Guan N, Yi W (2014) Improving the response time analysis of global fixed-priority multiprocessor scheduling. In: Embedded and Real-Time Computing Systems and Applications (RTCSA), 2014 IEEE 20th International Conference on, IEEE, pp 1–9