Exact schedulability test for sporadic mixed-criticality real-time systems using antichains and oracles

Simon Picard
Ourway
Belgium
snspicard@gmail.com

Antonio Paolillo
Vrije Universiteit Brussel
Belgium
antonio.paolillo@vub.be

Gilles Geeraerts
Université libre de Bruxelles
Belgium
gilles.geeraerts@ulb.be

Joël Goossens
Université libre de Bruxelles
Belgium
joel.goossens@ulb.be

Abstract

This work addresses the problem of exact schedulability assessment in uniprocessor mixed-criticality real-time systems with sporadic task sets. We model the problem by means of a finite automaton that has to be explored in order to check for schedulability. To mitigate the state explosion problem, we provide a generic algorithm which is parameterised by several techniques called oracles and simulation relations. These techniques leverage results from the scheduling literature as "plug-ins" that make the algorithm more efficient in practice. Our approach achieves up to a 99.998% reduction in the search space required for exact schedulability testing, making it practical for a range of task sets, up to 8 tasks or maximum periods of 350. This method enables to challenge the pessimism of an existing schedulability test and to derive a new dynamic-priority scheduler, demonstrating its good performance.

CCS Concepts

• **Computer systems organization → Real-time systems**.

Keywords

hard real-time scheduling, formal methods, automata theory, sporadic tasks, exact schedulability test, mixed criticality, graph, antichain, dynamic-priority, schedulability analysis

ACM Reference Format:
Simon Picard, Antonio Paolillo, Gilles Geeraerts, and Joël Goossens. 2024. Exact schedulability test for sporadic mixed-criticality real-time systems using antichains and oracles. In *The 32nd International Conference on Real-Time Networks and Systems (RTNS 2024), November 07–08, 2024, Porto, Portugal.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3696355.3699702

1 Introduction

The industry push for integrating systems of varying criticality levels onto a single platform has driven the real-time community to develop the mixed-criticality system model, hence producing a large body of research results for systems designed to degrade gracefully [11]. The dual-criticality model enables the concurrent execution of high-criticality tasks (HI) alongside low-criticality tasks (LO). The HI tasks are certified with two estimates of their worst-case execution time — an optimistic estimate and a pessimistic upper bound — whereas the LO tasks are less critical and can be safely disabled should a HI task exceed its optimistic estimate.

Many results about this model have been produced, addressing variations of the model [7, 17] varying CPU speeds [42, 43], multi-core and parallel systems [6, 28], and applications in industrial settings [19, 27, 35]. Despite this progress, however, one fundamental problem remains open: given a generic scheduling algorithm, how to assess the schedulability of a mixed-criticality task set? Agrawal and Baruah proved that determining the schedulability of independent dual-criticality periodic or sporadic implicit-deadline tasks is NP-hard in the strong sense [3], hinting that the schedulability assessment of a given scheduling algorithm might be hard as well. Indeed, to the best of our knowledge, no exact — necessary *and* sufficient — schedulability test has been produced to date for a general scheduling algorithm, even for a uniprocessor platform.

The quest for schedulability tests has produced two main lines of research. On the one hand, the real-time community has focussed on efficient tests for specific schedulers, but they are in many cases not exact. For example, EDF-VD [9], a notable scheduling algorithm for mixed-criticality systems only provides a *sufficient* and notoriously pessimistic test. On the other hand, inspired by the formal methods community, several works have proposed to model the possible behaviours of the system by means of an *automaton* [5] whose states correspond to all the possible states of the system. In this case, looking for potential deadline misses can be done by analysing all the states of the automaton. While such approaches are guaranteed to provide an exact test [1, 4], they suffer from the state explosion problem making them impractical for realistic systems.

In this work, we seek to reconcile both approaches. While our work is rooted in the automaton-based approach, we show how results and knowledge from the real-time community can be exploited to make the traversal of automata states more tractable. We believe that such hybrid techniques are very rare in the literature, with some notable exceptions like the works of Asyaban [4] (which addresses mixed criticality in the FTP case) and Ranjha [39]. However, those papers develop *ad hoc* techniques (in the cases of FTP and FJP scheduling), while we strive for more generality.

Our contributions are thus as follows. (1) We provide an automaton model for systems of dual-criticality sporadic tasks on a uniprocessor platform where the scheduler is left as a parameter. Hence, (2) we obtain an ***exact*** schedulability test for any given scheduling algorithm that is deterministic and memoryless. (3) This test consists in exploring the states of the automaton and is parameterised by optimisations to leverage knowledge built in the

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
RTNS 2024, November 07–08, 2024, Porto, Portugal
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-1724-6/24/11
https://doi.org/10.1145/3696355.3699702

Table 1: Comparing with the related work.

	[5]	[4]	[29]	Us
Criticality	Single	**Dual**	Single	**Dual**
Priority classes	**Any**	FTP	**Any**	**Any**
Pruning rules	✖	✔	✖	✖
Antichains	✖	✖	✔	✔
Oracles	✖	✖	✖	✔
Multi-processor	✔	✖	✔	✖

real-time community. More precisely, sufficient and necessary tests can be exploited as *oracles* that tell the algorithm to avoid exploring some states. We also exploit a *simulation relation* between states to further prune the state space in the spirit of the antichain approach of the formal method community [14, 15, 21, 45]. (4) We show how to exploit this generic framework in our setting of mixed criticality by defining proper oracles and a simulation relation. On this basis, we evaluate empirically our approach. (5) Experiments on task sets generated randomly according to the model specification show promising results: the efficiency of our approach enables to reduce the state search space by up to 99.998%. The genericity of our approach also allows us (6) to evaluate the EDF-VD scheduling algorithm and its associated test, showing that the test is pessimistic w.r.t. its actual scheduling capabilities. (7) For illustration, we study and present a new dynamic-priority scheduler, LWLF. We demonstrate its excellent performance, thanks to its anticipation of mode change impact when operating in LO mode. Table 1 graphically compares our approach with the closest related work.

Due to space limits, additional oracles and proofs have been omitted and provided as supplementary material in an extended version of this article [37].

2 Problem definition

We consider a mixed-criticality sporadic task set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ with two levels of criticality, also referred to as a dual-criticality [11], to be scheduled on a uniprocessor platform. A dual-criticality sporadic task $\tau_i = \langle\langle C_i(\mathsf{LO}), C_i(\mathsf{HI})\rangle, D_i, T_i, L_i\rangle$ is characterised by a *minimum interarrival time* $T_i > 0$, a *relative deadline* $D_i > 0$, a *criticality level* $L_i \in \{\mathsf{LO}, \mathsf{HI}\}$ with $\mathsf{HI} > \mathsf{LO}$ and the *worst-case execution time* (WCET) tuple $\langle C_i(\mathsf{LO}), C_i(\mathsf{HI})\rangle$. Tasks will never execute for more than $C_i(L_i)$. Time is assumed to be discrete, ergo $\forall i : T_i, D_i, C_i(\mathsf{LO}), C_i(\mathsf{HI}) \in \mathbb{N} \setminus \{0\}$. I.e., all timing parameters are strictly-positive integers. It is assumed that $C_i(\mathsf{LO}) \leq C_i(\mathsf{HI})$ when $L_i = \mathsf{HI}$ and $C_i(\mathsf{HI}) = C_i(\mathsf{LO})$ when $L_i = \mathsf{LO}$. A dual-criticality sporadic task τ_i releases an infinite number of jobs, with each job release being separated by at least T_i units of time. The absolute deadline of jobs are set D_i units of time after their release, and jobs must signal completion before their absolute deadline. We assume jobs are independent, as formulated by Vestal [44]. At each clock-tick, the executing job can signal its completion. If a job did not signal completion after exhausting its $C_i(\mathsf{LO})$, then a mode change is triggered from LO to HI: jobs of LO tasks are discarded and LO tasks are not allowed to release jobs any more; active jobs of HI tasks receive an additional budget of $C_i(\mathsf{HI}) - C_i(\mathsf{LO})$, and future jobs of HI tasks will receive a budget of $C_i(\mathsf{HI})$. As a running example in the work, we define the task set $\tau^a = \{\tau_1, \tau_2\}$ with $\tau_1 = \langle\langle 1, 2\rangle, 2, 2, \mathsf{HI}\rangle$ and $\tau_2 = \langle\langle 1, 1\rangle, 2, 2, \mathsf{LO}\rangle$.

It is imperative, when validating a dual-criticality task sets, to first perform *"due diligence"* by ensuring the corresponding two following single-criticality task sets are schedulable: *(i)* the task set comprising only HI tasks, where $\forall \tau_i : C_i = C_i(\mathsf{HI})$, and *(ii)* the task set including both HI and LO tasks, where $\forall \tau_i : C_i = C_i(\mathsf{LO})$.

We aim to establish an exact schedulability test (necessary and sufficient) for any dual-criticality sporadic task set τ that tells us whether the set is schedulable — i.e., no job released by the tasks misses a deadline — with a given deterministic and preemptive scheduling algorithm with dynamic priorities. We support both *implicit* ($\forall \tau_i : D_i = T_i$) and *constrained* ($\forall \tau_i : D_i \leq T_i$) deadlines. The studied problem is difficult, notably due to two sources of non-determinism, the first relating to **the sporadic model** (we do not know when jobs are released) and the second to **mode change** (we do not know when it will occur).

Notations. Assuming $\alpha, \beta \in \{\mathsf{LO}, \mathsf{HI}\}$: the α-utilisation of the task $\tau_i \in \tau : U^\alpha(\tau_i) = C_i(\alpha)/T_i$, the α-utilisation of the task set $\tau : U^\alpha(\tau) = \sum_{\tau_i | L_i \geq \alpha} U^\alpha(\tau_i)$, the α-utilisation of the tasks of criticality β in the task set $\tau : U^\alpha_\beta(\tau) = \sum_{\tau_i | L_i = \beta} U^\alpha(\tau_i)$, and the average utilisation of the task set $\tau : U^{\text{avg}}(\tau) = \frac{U^{\mathsf{LO}}(\tau) + U^{\mathsf{HI}}(\tau)}{2}$.

3 Automaton based semantic

To obtain a formal definition of the semantics of the system and of the schedulability problem, we develop an automaton-based formalism inspired from Baker and Cirinei [5]. An automaton is a graph whose nodes are called *states* and whose (directed) edges are called *transitions*. States model the states of the system. Changes from a state to another, triggered by a task, a job, or the scheduler, are modelled by the transitions. Hence, each path in the automaton is a possible execution of the system, and we will thus look for executions that reach states where deadlines are missed (the so-called *failure states*) to check whether the system is schedulable or not. We use the following formal definition of automaton:

Definition 1. *An* **automaton** *is a tuple $A = \langle V, E, v_0, F\rangle$ where V is a set of states, $E \subseteq V \times V$ is a set of transitions between states, $v_0 \in V$ is the initial state and $F \subseteq V$ is a set of failure states. An automaton is finite iff V is a finite set.*

The problem we consider on automata is the problem of *safety* w.r.t. a designated set of *failure* states F that need to be avoided. A *path* in a finite automaton $A = \langle V, E, v_0, F\rangle$ is a finite sequence of states $v_1, v_2, \ldots, v_k$ such that $\forall\, 1 \leq j < k : (v_j, v_{j+1}) \in E$. For a subset of states $V' \subseteq V$, if there exists a path $v_1, v_2, \ldots, v_k$ in A such that $v_k \in V'$, we say that v_1 can reach V'. We denote the set of states that can be reached from a state $v \in V$ by $\text{Reach}(v)$. Then, the *safety problem* asks, given an automaton A, whether the initial state v_0 *cannot* reach the set of failure states F — i.e., there is no path from v_0 to any state $v \in F$ in the automaton, denoted by $\text{Reach}(v_0) \cap F = \emptyset$. If this is the case, we say that A is *safe*, otherwise (when v_0 can reach F) we say it is *unsafe*.

Let $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ be a task set as defined in section 2. We model the behaviour of τ by means of an automaton A, and we reduce the schedulability problem of τ to an instance of the safety

problem in A. A state captures the following dynamic (or "run-time") information about each task τ_i: *(i)* the *earliest next arrival time* $\mathsf{nat}(\tau_i)$ relative to the current instant, and *(ii)* the *worst-case remaining execution time* $\mathsf{rct}(\tau_i)$ of the current job of τ_i for the current level of criticality. In addition, we need to remember the current global *criticality level* (or mode) $\mathsf{cri} \in \{\mathsf{LO}, \mathsf{HI}\}$ of the system. Hence, each system state will be a tuple of the form $\langle \mathsf{rct}_S, \mathsf{nat}_S, \mathsf{cri}_S \rangle$:

Definition 2 (System states). *Let $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ be a mixed-criticality sporadic task system. Let $T_{\max} = \max_i T_i$ and $C_{\max} = \max_i C_i(L_i)$. A* **system state** *of τ is a tuple $S = \langle \mathsf{rct}_S, \mathsf{nat}_S, \mathsf{cri}_S \rangle$ where: $\mathsf{nat}_S : \tau \mapsto \{0, 1, \ldots, T_{\max}\}$ associates each task τ_i to the minimal delay $\mathsf{nat}_S(\tau_i)$ that must elapse before the next job of the task can be released; $\mathsf{rct}_S : \tau \mapsto \{0, 1, \ldots, C_{\max}\}$ associates each task τ_i to its maximal remaining execution time $\mathsf{rct}_S(\tau_i)$ for the current criticality of the system; and $\mathsf{cri}_S \in \{\mathsf{LO}, \mathsf{HI}\}$ is the current criticality of the system. We denote by $\mathsf{States}(\tau)$ the set of all system states of τ.*

Intuitively, each state contains the current run-time information of the systems at a particular instant, i.e., how far we are in the execution of the active jobs (with the $\mathsf{rct}_S(\tau)$ values) and how close we are to the next releases of jobs (with the $\mathsf{nat}_S(\tau)$ values). From these definitions, we derive other useful definitions in the following. Notice that the symbols $\wedge$, $\vee$ and $\neg$ respectively denote the *conjunction*, *inclusive disjunction* and *negation* logical operators.

Definition 3 (Time to deadline). *Let $\mathsf{ttd}_S(\tau_i) = \mathsf{nat}_S(\tau_i) - (T_i - D_i)$ be the* **time to deadline**, *the time remaining before the absolute deadline of the last submitted job [5] of $\tau_i \in \tau$ in state S. Note that when deadlines are implicit, we have $\mathsf{ttd}_S(\tau_i) = \mathsf{nat}_S(\tau_i)$.*

Definition 4 (Active tasks). *A task τ_i is* **active** *in a state S iff it currently has a job that is not completed in S. The set of active tasks in S is $\mathsf{Active}(S) = \{\tau_i \mid \mathsf{rct}_S(\tau_i) > 0\}$.*

Definition 5 (Eligible task). *A task τ_i is* **eligible** *in the state $S = \langle \mathsf{rct}_S, \mathsf{nat}_S, \mathsf{cri}_S \rangle$ iff it can release a job in this state — i.e., the task does not currently have an active job, the last job was submitted at least T_i time units ago and its criticality is greater than or equal to the state's. The set of eligible tasks in S is: $\mathsf{Eligible}(S) = \{\tau_i \mid \mathsf{rct}_S(\tau_i) = \mathsf{nat}_S(\tau_i) = 0 \wedge L_i \geq \mathsf{cri}_S\}$.*

Definition 6 (Implicitly completed task). *A task τ_i is* **implicitly completed** *in the state $S = \langle \mathsf{rct}_S, \mathsf{nat}_S, \mathsf{cri}_S \rangle$ iff its latest job has been executed for its maximal execution time. The set of implicitly completed tasks in S is $\mathsf{Completed}(S) = \{\tau_i \mid \mathsf{rct}_S(\tau_i) = 0 \wedge C_i(\mathsf{cri}_S) = C_i(L_i)\}$.*

Note that $C_i(\mathsf{cri}_S) = C_i(L_i)$ represents the condition "its latest job has been executed for its maximal execution time". The condition $\mathsf{cri}_S = L_i$ is not appropriate because it is possible for a HI-critical job to have the same execution time in LO and HI. Therefore, when $C_i(\mathsf{LO}) = C_i(\mathsf{HI})$, $\mathsf{rct}_S(\tau_i) = 0$ and $\mathsf{cri}_S = \mathsf{LO}$ then τ_i is implicitly completed because $C_i(\mathsf{cri}_S) = C_i(\mathsf{LO}) = C_i(L_i) = C_i(\mathsf{HI})$ even if $\mathsf{cri}_S = \mathsf{LO} \neq L_i = \mathsf{HI}$.

Definition 7 (Deadline-miss state). *A state S is a* **deadline-miss state** *if at least one task's job reached its deadline without executing all its maximal execution time. The set of deadline-miss states on τ is $\mathsf{DeadlineMiss}(\tau) = \{S \mid \exists \tau_i \in \tau : \mathsf{rct}_S(\tau_i) > 0 \wedge \mathsf{ttd}_S(\tau_i) \leq 0\}$.*

Definition 8 (Scheduler). *A uniprocessor* **memoryless scheduler** *for τ is a function $\mathsf{sch} : \mathsf{States}(\tau) \mapsto \tau \cup \{\bot\}$ s.t. $\mathsf{sch}(S) \in \mathsf{Active}(S)$ or $\mathsf{sch}(S) = \bot$ when no task is to be scheduled. Moreover, we say that the scheduler sch is* **deterministic**, *iff for all $S_1, S_2 \in \mathsf{States}(\tau)$ s.t. $\mathsf{Active}(S_1) = \mathsf{Active}(S_2)$ and $\mathsf{cri}_{S_1} = \mathsf{cri}_{S_2}$, for all $\tau_i \in \mathsf{Active}(S_1) : \mathsf{nat}_{S_1}(\tau_i) = \mathsf{nat}_{S_2}(\tau_i) \wedge \mathsf{rct}_{S_1}(\tau_i) = \mathsf{rct}_{S_2}(\tau_i)$ implies $\mathsf{sch}(S_1) = \mathsf{sch}(S_2)$.*

A *memoryless scheduler* is a scheduler that makes decisions based only on the current state, and not on the history of previous states. The *deterministic* property means that the scheduler always takes the same decision given the same criticality level and the same active task characteristics. It implies that the decisions of a deterministic scheduler do not involve any randomness in the job selection and are unaffected by the inactive tasks. In this work, we will only consider deterministic memoryless schedulers. As an example, we define the EDF-VD scheduler [9] within our framework:

Definition 9 (EDF-VD scheduler). *With $\lambda = \frac{U_{\mathsf{HI}}^{\mathsf{LO}}(\tau)}{1 - U_{\mathsf{LO}}^{\mathsf{LO}}(\tau)}$ a discount factor, let $\mathsf{ttvd}_S(\tau_i)$ be the time remaining before the virtual deadline of the last submitted job of $\tau_i \in \mathsf{Active}(S)$ in state S defined as follows:*

$$\mathsf{ttvd}_S(\tau_i) = \begin{cases} \mathsf{nat}_S(\tau_i) - (T_i - D_i \cdot \lambda) & \text{if } L_i = \mathsf{HI} \\ \mathsf{ttd}_S(\tau_i) & \text{otherwise.} \end{cases}$$

Further, we let $\mathsf{min_d}(S)$ be the task $\tau_i \in \mathsf{Active}(S)$ which has the minimal deadline in S. That is, τ_i is s.t. for all $\tau_k \in \mathsf{Active}(S) \setminus \{\tau_i\}$: $\mathsf{ttd}_S(\tau_k) > \mathsf{ttd}_S(\tau_i)$ or $\mathsf{ttd}_S(\tau_k) = \mathsf{ttd}_S(\tau_i) \wedge k > i$. We define $\mathsf{min_{vd}}(S)$ similarly, using virtual deadlines instead of deadlines (substituting ttvd_S for ttd_S in the definition). Then, for all states S, we let: $\mathsf{sch}_{EDF\text{-}VD}(S)$

$$= \begin{cases} \bot & \text{if } \mathsf{Active}(S) = \emptyset \\ \mathsf{min_d}(S) & \text{else if } \mathsf{cri}_S = \mathsf{HI} \vee U_{\mathsf{LO}}^{\mathsf{LO}}(\tau) + U_{\mathsf{HI}}^{\mathsf{HI}}(\tau) \leq 1 \\ \mathsf{min_{vd}}(S) & \text{otherwise.} \end{cases}$$

In the two first cases, the scheduler is behaving exactly as EDF. The original work of EDF-VD [9] assumes implicit deadlines (i.e., $\forall \tau_i : D_i = T_i$). Notice we cannot have $\lambda > 1$ as it would mean $U^{\mathsf{LO}} > 1$, the task set not being schedulable in LO mode, contradicting the requisites outlined in Section 2 (*"due diligence"*).

Thanks to these notions, we can define the transitions of the automaton. Those transitions must embed all the modifications happening to the system state within one clock-tick. Each of those modifications can be seen as an intermediary transition between two system states, and they happen in the following order: (1) *Release transitions* model the release of jobs by sporadic tasks at a given instant in time, (2) *Run transitions* model the elapsing of one time unit, and running the job selected by the scheduler, if any and (3) *Signal transitions* model a job signalling, or not, completion, and a potential resulting mode change.

In the automaton, an *actual transition* will exist only if those three intermediary transitions happen one after the other. We now formally define how each kind of intermediary transitions alter the state. We start by defining *release transitions*. Let S be a state in $\mathsf{States}(\tau)$. Intuitively, when the system is in state S, a task τ_i releasing a new job has the effect to update S by setting $\mathsf{nat}(\tau_i)$ to T_i and $\mathsf{rct}(\tau_i)$ to $C_i(\mathsf{cri}_S)$. Formally:

Definition 10 (Release transition). *Let* $S = \langle \mathsf{rct}_S, \mathsf{nat}_S, \mathsf{cri}_S \rangle \in \mathsf{States}(\tau)$ *be a system state and* $\tau^+ \subseteq \mathsf{Eligible}(S)$ *be a set of tasks that are eligible to release a new job in the system. Then, we say that* $S^+ = \langle \mathsf{rct}_{S^+}, \mathsf{nat}_{S^+}, \mathsf{cri}_{S^+} \rangle$ *is a* τ^+**-release successor** *of* S*, denoted* $S \xrightarrow{\tau^+}_{\mathsf{rl}} S^+$ *iff:*

(1) $\forall \tau_i \in \tau^+ : \mathsf{nat}_{S^+}(\tau_i) = T_i$ *and* $\mathsf{rct}_{S^+}(\tau_i) = C_i(\mathsf{cri}_S)$
(2) $\forall \tau_i \notin \tau^+ : \mathsf{nat}_{S^+}(\tau_i) = \mathsf{nat}_S(\tau_i)$ *and* $\mathsf{rct}_{S^+}(\tau_i) = \mathsf{rct}_S(\tau_i)$
(3) $\mathsf{cri}_{S^+} = \mathsf{cri}_S$.

Notice that we allow $\tau^+ = \emptyset$, that is, no task releases a new job in the system. Also note that no time elapsed in that transition, so no $\mathsf{nat}(\tau_i)$ nor $\mathsf{rct}(\tau_i)$ must be updated. Furthermore, observe that changing the definition with $\tau^+ = \mathsf{Eligible}(S)$ would lead to consider a periodic task model (without offset), as all tasks that could release a job would then have to release it immediately. Next, we move to *run transitions*. Let S be a state in $\mathsf{States}(\tau)$, and let run be the scheduling decision to apply, i.e., either $\mathsf{run} \in \tau$ and run must be executed or $\mathsf{run} = \bot$ and the processor remains idle. Then, letting one time unit elapse from S under the run scheduling decision leads to decrementing the rct of the task run (and only this task) if $\mathsf{run} \in \tau$, and to decrementing the nat of all tasks. Formally:

Definition 11 (Run transition). *Let* $S = \langle \mathsf{rct}_S, \mathsf{nat}_S, \mathsf{cri}_S \rangle \in \mathsf{States}(\tau)$ *be a system state and* $\mathsf{run} \in \tau \cup \{\bot\}$ *be a task to be executed or* $\bot$ *if no task is to be executed. Then, we say that* $S' = \langle \mathsf{rct}_{S'}, \mathsf{nat}_{S'}, \mathsf{cri}_{S'} \rangle$ *is a* **run successor** *of* S *under* run*, denoted by* $S \xrightarrow{\mathsf{run}}_{\mathsf{rn}} S'$ *iff:*

(1) *For all* $\tau_j \in \tau \setminus \{\mathsf{run}\}$*:* $\mathsf{rct}_{S'}(\tau_j) = \mathsf{rct}_S(\tau_j)$ *and* $\mathsf{run} \neq \bot$ *implies* $\mathsf{rct}_{S'}(\mathsf{run}) = \mathsf{rct}_S(\mathsf{run}) - 1$
(2) $\forall \tau_i \in \tau : \mathsf{nat}_{S'}(\tau_i) = \max\{\mathsf{nat}_S(\tau_i) - 1, 0\}$
(3) $\mathsf{cri}_{S'} = \mathsf{cri}_S$.

Finally, let us define *signal transitions*. When the system is in state $S \in \mathsf{States}(\tau)$, there are typically two possible scenarios for the task τ_r which has just been executed: it can signal completion, or not. We call the combination of this information a *setup*: S, τ_r and whether τ_r signals completion. This non-deterministic behaviour leads to three different outcomes for the state S. Outcome (1) is when S does not change at all, which can result from different setups. Either no task has been executed; or τ_r must signal completion because it has exhausted all its execution time budget for its worst criticality mode, i.e. τ_r is *implicitly* completed; or the task τ_r does not signal completion without having exhausted all its execution time budget; or the task τ_r signals completion and has exhausted all its execution time budget for the current criticality mode, which is not its worst one. Outcome (2) is when τ_r signals completion with remaining execution time budget for the current criticality mode, i.e. τ_r is *explicitly* completed. Outcome (3) happens when τ_r does not signal completion, has exhausted its execution time budget for the current criticality mode but still has execution time budget in the higher (HI) criticality mode, which triggers a mode change. When a task τ_r signals completion explicitly, then the resulting state is identical to S except for the rct of τ_r, which is set to 0. Such state is formalised as $\mathsf{sigCmp}_S(\tau_r)$ where $\mathsf{sigCmp}_S(\tau_r)$ is the state S' s.t. : $\mathsf{nat}_S = \mathsf{nat}_{S'}$; $\mathsf{cri}_S = \mathsf{cri}_{S'}$; $\mathsf{rct}_{S'}(\tau_r) = 0$; and, for all $\tau_i \in \tau \setminus \{\tau_r\}$: $\mathsf{rct}_{S'}(\tau_i) = \mathsf{rct}_S(\tau_i)$. When a mode change occurs, cri becomes HI and all the active HI-critical tasks see their rct increase by the difference between their execution time in levels HI and LO. All LO-critical tasks are discarded, having rct set to 0. The state obtained from S when τ_r triggers a mode change is thus denoted as $\mathsf{critUp}_S(\tau_r)$. Hence, $\mathsf{critUp}_S(\tau_r)$ is formalised as the state S' s.t. : $\mathsf{nat}_S = \mathsf{nat}_{S'}$; $\mathsf{cri}_{S'} = \mathsf{HI}$; and, for all $\tau_i \in \tau$: $\mathsf{rct}_{S'}(\tau_i)$

$$= \begin{cases} \mathsf{rct}_S(\tau_i) + C_i(\mathsf{HI}) - C_i(\mathsf{LO}) & \text{if } L_i = \mathsf{HI} \wedge \mathsf{rct}_S(\tau_i) > 0 \\ C_i(\mathsf{HI}) - C_i(\mathsf{LO}) & \text{else if } \tau_i = \tau_r \\ 0 & \text{otherwise.} \end{cases}$$

Thanks to these two functions, we can now define formally signal transitions.

Definition 12 (Signal transition). *Let* $\mathsf{ran} \in \tau \cup \{\bot\}$ *be the task that has just been executed if any,* $\bot$ *otherwise. Let* $\theta \in \{\mathsf{True}, \mathsf{False}\}$ *denote whether* ran *signals completion explicitly. Then, we say that* $S^- = \langle \mathsf{rct}_{S^-}, \mathsf{nat}_{S^-}, \mathsf{cri}_{S^-} \rangle$ *is a* **signal successor** *of* S *under* ran*, denoted* $S \xrightarrow{\mathsf{ran},\theta}_{\mathsf{sg}} S^-$ *iff:*

$$S^- = \begin{cases} S & \text{if } \mathsf{ran} \in \mathsf{Completed}(S) \cup \{\bot\} \\ & \quad \vee (\neg\theta \wedge \mathsf{rct}_S(\mathsf{ran}) > 0) \quad (1) \\ & \quad \vee (\theta \wedge \mathsf{rct}_S(\mathsf{ran}) = 0) \\ \mathsf{sigCmp}_S(\mathsf{ran}) & \text{if } \mathsf{ran} \notin \mathsf{Completed}(S) \cup \{\bot\} \quad (2) \\ & \quad \wedge\theta \wedge \mathsf{rct}_S(\mathsf{ran}) > 0 \\ \mathsf{critUp}_S(\mathsf{ran}) & \text{if } \mathsf{ran} \notin \mathsf{Completed}(S) \cup \{\bot\} \quad (3) \\ & \quad \wedge\neg\theta \wedge \mathsf{rct}_S(\mathsf{ran}) = 0. \end{cases}$$

The cases' number match the outcomes previously described. Note that S always has itself has a signal successor. Then, when $\mathsf{ran} \notin \mathsf{Completed}(S) \cup \{\bot\}$, S has another signal successor which is either $\mathsf{sigCmp}_S(\mathsf{ran})$ or $\mathsf{critUp}_S(\mathsf{ran})$ depending on the setup.

Finally, we define the automaton $A(\tau, \mathsf{sch})$ that formalises the behaviour of the system of dual-criticality sporadic task set τ, when executed under a scheduling algorithm sch:

Definition 13. *Given a system of dual-criticality sporadic tasks* τ *and a scheduler* sch*, the automaton* $A(\tau, \mathsf{sch})$ *is the tuple* $\langle V, E, v_0, F \rangle$ *where:*

(1) $V = \mathsf{States}(\tau)$
(2) $(S_1, S_2) \in E$*, iff there are* $S^+, S' \in \mathsf{States}(\tau)$*,* $\tau^+ \subseteq \tau$ *and* $\theta \in \{\mathsf{True}, \mathsf{False}\}$ *s.t.:* $S_1 \xrightarrow{\tau^+}_{\mathsf{rl}} S^+ \xrightarrow{\mathsf{sch}(S^+)}_{\mathsf{rn}} S' \xrightarrow{\mathsf{sch}(S^+),\theta}_{\mathsf{sg}} S_2$
(3) $v_0 = \langle \mathsf{rct}_{v_0}, \mathsf{nat}_{v_0}, \mathsf{cri}_{v_0} \rangle$ *where* $\mathsf{cri}_{v_0} = \mathsf{LO}$ *and* $\forall \tau_i \in \tau : \mathsf{nat}_{v_0}(\tau_i) = \mathsf{rct}_{v_0}(\tau_i) = 0$
(4) $F = \mathsf{DeadlineMiss}(\tau)$

As we assumed that sch must be deterministic, run and ran, equalling both to $\mathsf{sch}(S^+)$, will be the same, as intended by the definition of the automaton. Each possible execution of the task set corresponds to a path in $A(\tau, \mathsf{sch})$ and vice versa. States in $\mathsf{DeadlineMiss}(\tau)$ correspond to states of the system where a deadline is or has been missed. Hence, the set of dual-criticality sporadic tasks τ is feasible under scheduler sch iff $A(\tau, \mathsf{sch})$ is *safe*, i.e., $\mathsf{DeadlineMiss}(\tau)$ is not reachable in $A(\tau, \mathsf{sch})$.

Figure 1 illustrates such an automaton, representing the possible execution of a task set scheduled with the EDF-VD scheduler. In this example, the automaton depicts the dual-criticality sporadic task set τ^a as specified in section 2. System states are represented by nodes. For the purpose of saving space, we represent a state S with

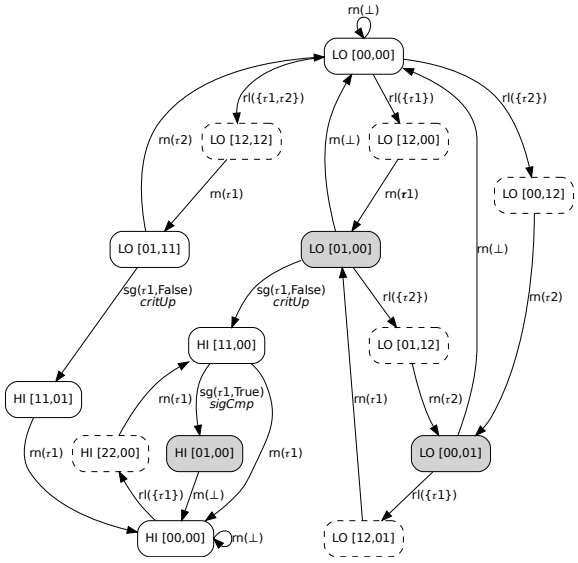

Figure 1: $A(\tau^a, \text{sch}_{\text{EDF-VD}})$ developed automaton with intermediary transitions. Intermediary states have a dashed outline, and greyed out states are simulated.

the $\chi[\alpha\beta, \gamma\delta]$ format, meaning $\text{cri}_S = \chi$, $\text{rct}_S(\tau_1) = \alpha$, $\text{nat}_S(\tau_1) = \beta$, $\text{rct}_S(\tau_2) = \gamma$ and $\text{nat}_S(\tau_2) = \delta$. We explicitly represent run transitions by edges labelled with rn, signal transitions by edges labelled with sg, and release transitions by edges labelled with rl. The $\tau^+ = \emptyset$ release loops and $S^- = S$ signal loops are omitted for readability. Notice that the graph in Figure 1 is a developed way of representing $A(\tau^a, \text{sch}_{\text{EDF-VD}})$ as it is built with intermediary transitions.

We can observe different behaviour from the system. There might be several release transitions from a single state, as it can be seen from the initial node $\text{LO}[00, 00]$. Indeed, four combinations of releases are possible. When $\tau^+ = \emptyset$, there is a release transition from the state to itself, which is omitted on the figure. The signal transition from the state $\text{LO}[01, 11]$ contains a mode change which leads to a HI-criticality system state and the LO-criticality task τ_2 is discarded. Finally, in the signal transition from the state $\text{HI}[11, 00]$, the task τ_1 signals completion explicitly before exhausting all its execution time budget $C_i(\text{HI}) = 2$, and its rct is set to 0. Per the automaton definition, a signal transition exists only when it is preceded by a run transition, emanating from $\text{HI}[22, 00]$ in this case. Thus, the first time that the state $\text{HI}[11, 00]$ was visited via a signal transition from $\text{LO}[01, 00]$, the signal transition to $\text{HI}[01, 00]$ was not yet generated.

4 An efficient algorithm for safety with simulation relation and oracles

In this section, we present an efficient algorithm to check for safety in an automaton, which is based on two hypothesis: *(i)* we can query an *oracle* which can sometimes tell us whether a given state can or cannot reach the set of failure states F; *(ii)* we have at our disposal a *simulation relation* between states. Intuitively, when v' simulates v, it means that all paths that can occur from v can be witnessed by a path from v'. In particular, when v can reach the failure states, then v' can reach them too.

Both the oracle and the simulation relation can improve the performance of an algorithm answering the safety problem. Whenever the oracle guarantees that v cannot reach the failure states, there is no need to compute the paths starting in v (and it is safe to do so, as no path to the failure states will be dropped). When the oracle says that a state v (that is reachable from v_0) can reach the failure states F, we can immediately conclude that F is reachable from v_0 as well, hence that the automaton is *unsafe*. Finally, when two reachable states v and v' are found s.t. v' simulates v, only the paths starting from v' need to be computed.

We show in section 5 how such an oracle and such a simulation relation can be derived in our mixed criticality setting, but the generality of the concepts we explain here make them applicable to other scheduling problems. We formalise the notion of oracle:

Definition 14 (Safe and unsafe states). *Let $A = \langle V, E, v_0, F\rangle$ be a finite automaton. Then, a state v is* **safe** *iff there is no path starting in v that reaches F, i.e.,* $\text{Reach}(v) \cap F = \emptyset$.

A state v that is not safe (i.e., there is a path from v that reaches F) is called unsafe.

In order to formalise the oracle, we assume that we have at our disposal a set *Safe* containing only safe states and a set *Unsafe* containing only unsafe states. Without loss of generality, we assume that $F \subseteq \mathit{Unsafe}$. Observe that we can have $\mathit{Safe} = \emptyset$ and $\mathit{Unsafe} = F$, so we do not request that $\mathit{Safe} \cup \mathit{Unsafe} = V$. However, the bigger those sets, the more efficient our algorithm will potentially be. Next, we define the notion of *simulation relation*.

Definition 15 (Simulation relation). *Let $A = \langle V, E, v_0, F\rangle$ be a finite automaton. A relation $\preccurlyeq \subseteq V \times V$ is a* **simulation relation** *iff it respects the following properties:*

1. *$\preccurlyeq$ is a preorder, i.e., it is reflexive and transitive.*
2. *for all states v_1 and v_1' s.t. $v_1 \preccurlyeq v_1'$, for all v_2 s.t. $(v_1, v_2) \in E$, there exists $v_2' \in V$ s.t.: (i) $(v_1', v_2') \in E$; (ii) and $v_2 \preccurlyeq v_2'$.*
3. *for all states v and v' s.t. $v \preccurlyeq v' : v \in F$ implies that $v' \in F$.*

Whenever $v_1 \preccurlyeq v_1'$, we say that v_1' *simulates* v_1. Thus, the definition says that, for every move (v_1, v_2) that can be performed from v_1, there is a matching move (v_1', v_2') from v_1'. Here, 'matching' means that the state v_2' simulates v_2 ($v_2 \preccurlyeq v_2'$) and that v_2' is a failure state if v_2 is. From this definition, we establish the following:

Proposition 1. *Let $A = \langle V, E, v_0, F\rangle$ be a finite automaton. For all pairs of states v and v' s.t. $v \preccurlyeq v'$, the following holds: (1) if v is unsafe, then v' is unsafe too; (2) if v' is safe, then v is safe too.*

This observation prompts for the definition of the set of 'all states that are simulated by/simulate a given state v'. Given a state v and a simulation relation $\preccurlyeq$, we let the *upward-closure* of v be the set of all states that simulate v, i.e., $\uparrow^{\preccurlyeq}(v) = \{v' \mid v \preccurlyeq v'\}$; and its *downward-closure* be $\downarrow^{\preccurlyeq}(v) = \{v' \mid v' \preccurlyeq v\}$ (the set of all states that are simulated by v). The upward and downward-closures are also defined on a set of states, for which it becomes the union of the respective closures of all the states within the set. Finally, given a set of states $Y \subseteq V$, we let $\max^{\preccurlyeq}(Y) = \{v \in S \mid \nexists v' \in S : v \preccurlyeq v'\}$ be the set of *maximal* states of S w.r.t. the simulation $\preccurlyeq$, i.e., those states in S that are not simulated by any other state in S.

Let us now explain how these notions (sets of safe and unsafe states, and simulation relation) can be exploited in a generic algorithm to solve the safety problem in a finite automaton. First, let us recall the general strategy of *breadth first search* to solve safety in an automaton. It can be formalised as computing two sequences of sets $(R_i)_{i\geq 0}$ and $(N_i)_{i\geq 0}$ s.t., for all $i \geq 0$, R_i and N_i are the sets of all states that can be reached from v_0 in at most i steps and exactly i steps, respectively. We denote by $\mathrm{Succ}(Y) = \{v' | \exists (v, v') \in E \, \forall v \in Y\}$ the set of all successors of all states in $Y \subseteq V$, and we have $N_0 = \{v_0\}$, $\forall i \geq 0 : N_{i+1} = \mathrm{Succ}(N_i) \setminus R_i$, $R_0 = \{v_0\}$, and $\forall i \geq 0 : R_{i+1} = R_i \cup N_{i+1}$. Then, an algorithm to decide reachability consists in computing $N_1, R_1, N_2, R_2, \ldots N_j, R_j, \ldots$ until: *(i)* either $N_j \cap F \neq \emptyset$, in which case F is reachable from v_0; or *(ii)* $R_j = R_{j+1}$, in which case R_j contains all the reachable states and we know that F is not reachable (otherwise, we would have already returned 'fail'). In order to improve this algorithm, we compute the sequences $\left(\widetilde{R}_i\right)_{i\geq 0}$ and $\left(\widetilde{N}_i\right)_{i\geq 0}$:

$$\widetilde{N}_0 = \widetilde{R}_0 = \{v_0\} \setminus \downarrow^{\preccurlyeq}(\mathit{Safe})$$
$$\forall i \geq 0 : \widetilde{N}_{i+1} = \max^{\preccurlyeq}\left(\mathrm{Succ}\left(\widetilde{N}_i\right) \setminus \downarrow^{\preccurlyeq}\left(\widetilde{R}_i \cup \mathit{Safe}\right)\right)$$
$$\forall i \geq 0 : \widetilde{R}_{i+1} = \max^{\preccurlyeq}\left(\widetilde{R}_i \cup \widetilde{N}_{i+1}\right).$$

Intuitively, these sequences contain less elements than the original ones, but still retain enough information to solve the safety problem. Let us explain why. Assume the algorithm has computed set $\widetilde{R}_i$ and $\widetilde{N}_i$, and let us consider how it computes $\widetilde{N}_{i+1}$. First, the algorithm computes the successors of $\widetilde{N}_i$, then computes $\mathrm{Succ}\left(\widetilde{N}_i\right) \setminus \downarrow^{\preccurlyeq}\left(\widetilde{R}_i \cup \mathit{Safe}\right)$, i.e. it removes, from the successors of $\widetilde{N}_i$, all the elements that are simulated by an element from $\widetilde{R}_i$ or from *Safe*. Finally, it keeps only the maximal elements from this resulting set.

To understand why these optimisations are correct, assume there is a state $v \in \mathrm{Succ}\left(\widetilde{N}_i\right)$ which is also simulated by an element of *Safe* (hence $v \in \downarrow^{\preccurlyeq}(\mathit{Safe})$). With our optimisations, v is not in $\widetilde{N}_{i+1}$, and its successors will not be computed at the next step. However, this is not a problem, because v is simulated by a safe state, hence it is *safe too* by Proposition 1. Similarly, if $v \in \downarrow^{\preccurlyeq}\left(\widetilde{R}_i\right)$, it means we have already computed all the necessary successors at a previous step, keeping only those that are potentially unsafe. So, it is correct to avoid computing the successors of v, since we are looking for paths that lead to *unsafe* states. Now, assume that there is $v \in \mathrm{Succ}\left(\widetilde{N}_i\right) \setminus \downarrow^{\preccurlyeq}\left(\widetilde{R}_i \cup \mathit{Safe}\right)$ which is not $\preccurlyeq$-maximal. This implies that there is $v' \in \widetilde{N}_{i+1}$ that simulates v. There are two possibilities. Either v is *safe*, and we do not need to compute its successors. Or v is *unsafe*, but then, so is v'. So, again, we keep in $\widetilde{N}_{i+1}$ enough information to discover a path to the failure states. With these definitions, all sets $\widetilde{R}_i$ and $\widetilde{N}_i$ are *$\preccurlyeq$-antichains of $\preccurlyeq$-maximal states* (i.e., sets of states which are all incomparable and maximal w.r.t. $\preccurlyeq$) that retain enough information to find paths to F.

Based on these sequences, we propose Algorithm 1 to check for safety in a given finite automaton $A = \langle V, E, v_0, F\rangle$, when we have at our disposal a simulation relation $\preccurlyeq$ on V, and two sets *Safe* and *Unsafe* $\supseteq F$ of safe and unsafe states respectively, provided to us by an Oracle. This algorithm consists in computing $\widetilde{N}_0, \widetilde{R}_0, \widetilde{N}_1, \widetilde{R}_1, \ldots$ until either *(i)* $\widetilde{N}_i \cap \uparrow^{\preccurlyeq}(\mathit{Unsafe}) \neq \emptyset$, in which case we know that F is reachable; or *(ii)* $\widetilde{N}_i = \emptyset$, in which case we have explored enough states to guarantee that F is not reachable.

Algorithm 1 Antichain breadth first search with safe and unsafe states optimisation

```
1:  i ← 0
2:  R̃_0 ← {v_0} \ ↓^≼(Safe)
3:  Ñ_0 ← {v_0} \ ↓^≼(Safe)
4:  repeat
5:      if Ñ_i ∩ ↑^≼(Unsafe) ≠ ∅ then
6:          return Fail
7:      end if
8:      Ñ_{i+1} ← max^≼(Succ(Ñ_i) \ ↓^≼(R̃_i ∪ Safe))
9:      R̃_{i+1} ← max^≼(R̃_i ∪ Ñ_{i+1})
10:     i ← i + 1
11: until Ñ_i = ∅
12: return Safe
```

Proposition 2. *On all automata A, Algorithm 1 terminates and returns 'Fail' iff A is unsafe.*

Proof Sketch. We first establish termination by showing that all states computed in some $\widetilde{R}_i$ are reachable states, i.e., $\widetilde{R}_i \subseteq \mathrm{Reach}(A)$ for all $i \geq 0$. We then observe that the sets $\downarrow^{\preccurlyeq}\left(\widetilde{R}_i\right)$ keep growing, i.e., $\downarrow^{\preccurlyeq}\left(\widetilde{R}_0\right) \subseteq \downarrow^{\preccurlyeq}\left(\widetilde{R}_1\right) \subseteq \cdots \subseteq \downarrow^{\preccurlyeq}\left(\widetilde{R}_i\right) \subseteq \cdots$. However, since $\downarrow^{\preccurlyeq}\left(\widetilde{R}_i\right) \subseteq \downarrow^{\preccurlyeq}(\mathrm{Reach}(A))$ for all $i \geq 0$, and since $\downarrow^{\preccurlyeq}(\mathrm{Reach}(A))$ is a finite set, this sequence must eventually stabilise, i.e., $\downarrow^{\preccurlyeq}\left(\widetilde{R}_\ell\right) = \downarrow^{\preccurlyeq}\left(\widetilde{R}_{\ell+1}\right)$ for some ℓ. From this, we can conclude that $\widetilde{N}_{\ell+1} = \emptyset$ and the algorithm terminates.

Next, we prove soundness, i.e., when the algorithm returns 'Fail', we have indeed $\mathrm{Reach}(A) \cap F \neq \emptyset$. This stems again from the fact that $\widetilde{N}_i \subseteq \mathrm{Reach}(A)$ for all $i \geq 0$. Hence, when we return 'Fail', we have indeed found a reachable state that is unsafe.

Finally, we establish completeness, i.e., when $\mathrm{Reach}(A) \cap F \neq \emptyset$, the algorithm returns 'Fail'. To obtain this result, we consider one path $v_0, v_1, \ldots, v_k, v_{k+1}$ s.t., $v_{k+1} \in \uparrow^{\preccurlyeq}(\mathit{Unsafe})$, and $v_i \notin \uparrow^{\preccurlyeq}(\mathit{Unsafe})$ for all $0 \leq i \leq k$. That is, the path reaches $\uparrow^{\preccurlyeq}(\mathit{Unsafe})$ in its last state only. We show that the prefix $v_0, v_1, \ldots, v_k$ can be 'found' in the sequence $\widetilde{R}_0, \ldots, \widetilde{R}_k$ in the following sense: $0 \leq i \leq k$: $v_i \in \downarrow^{\preccurlyeq}\left(\widetilde{R}_i\right)$. From that, we conclude that $\widetilde{N}_{k+1} \cap \uparrow^{\preccurlyeq}(\mathit{Unsafe}) \neq \emptyset$, and that $\widetilde{N}_{k+1}$ will be computed by the algorithm, hence returning 'Fail'. □

5 A simulation relation and oracles for dual-criticality scheduling

In order to leverage the optimised algorithm of section 4 on the automaton defined in section 3 (to model the behaviour of dual-criticality task set), we define in this Section a suitable simulation relation and *ad hoc* oracles. In section 6, we will show that these optimisations are efficient in practice to reduce the size of the state space that needs to be explored.

5.1 Idle tasks simulation relation

We first define a simulation relation $\preccurlyeq_{idle}$, called the *idle tasks simulation relation* that can be computed efficiently by inspecting the values nat, rct and cri stored in the states.

Definition 16. *Let τ be a set of dual-criticality sporadic tasks. Then, the* **idle tasks preorder** $\preccurlyeq_{idle} \subseteq \text{States}(\tau) \times \text{States}(\tau)$ *is s.t. for all $S_1, S_2 : S_1 \preccurlyeq_{idle} S_2$ iff: (i) $\text{cri}_{S_2} = \text{cri}_{S_1}$; (ii) $\text{rct}_{S_2} = \text{rct}_{S_1}$; (iii) for all τ_i s.t. $\text{rct}_{S_1}(\tau_i) = 0 : \text{nat}_{S_2}(\tau_i) \leq \text{nat}_{S_1}(\tau_i)$; and (iv) for all τ_i s.t. $\text{rct}_{S_1}(\tau_i) > 0 : \text{nat}_{S_2}(\tau_i) = \text{nat}_{S_1}(\tau_i)$.*

This relation is transitive and reflexive, so it is indeed a preorder. The relation also defines a partial order on Active (S), because it is antisymmetric. Note that $S_1 \preccurlyeq_{idle} S_2$ implies that Active (S_2) = Active (S_1) since $\text{rct}_{S_2} = \text{rct}_{S_1}$. We show that this preorder is indeed a simulation relation for a deterministic scheduler:

Theorem 1. *Let τ be a dual-criticality sporadic task system and* sch *a deterministic uniprocessor scheduler for τ. Then $\preccurlyeq_{idle}$ is a simulation relation for $A(\tau, \text{sch})$.*

Proof Sketch. Let S_1, S_1^- and S_2 be three states in States (τ) s.t. $(S_1, S_1^-) \in E$ and $S_1 \preccurlyeq_{idle} S_2$, let us show that there exists $S_2^- \in \text{States}(\tau)$ with $(S_2, S_2^-) \in E$ and $S_1^- \preccurlyeq_{idle} S_2^-$.

Let S_1^+ and $S_1' \in \text{States}(\tau)$, $\tau^+ \subseteq \tau$ and $\theta \in \{\text{True}, \text{False}\}$ be such that : $S_1 \xrightarrow{\tau^+}_{\text{rl}} S_1^+ \xrightarrow{\text{sch}(S_1^+)}_{\text{rn}} S_1' \xrightarrow{\text{sch}(S_1^+),\theta}_{\text{sg}} S_1^-$. Those exist by Definition 13 and since $(S_1, S_1^-) \in E$.

First, observe that, by Definition 16, Eligible $(S_1) \subseteq$ Eligible (S_2). So, by Definition 10, there exists a τ^+-release transition from S_2. Let S_2^+ be the state s.t. $S_2 \xrightarrow{\tau^+}_{\text{rl}} S_2^+$. By Definition 10, again, it is easy to check that $S_1^+ \preccurlyeq_{idle} S_2^+$.

Next, let us show that we can simulate the run transition from S_2^+. To alleviate notations, let us denote, from now on, $\text{sch}(S_1^+)$ by run_1 and $\text{sch}(S_2^+)$ by run_2. Since the scheduler is deterministic, we have $\text{run}_1 = \text{run}_2$, by Definition 8. Hence, we can let S_2' be the state s.t. $S_2^+ \xrightarrow{\text{run}_2}_{\text{rn}} S_2'$. Finally, by Definition 16 and 8, it is easy to check that $S_1' \preccurlyeq_{idle} S_2'$.

Then, let us show that we can simulate the signal transitions from S_1'. Let S_2^- be the state s.t. $S_2' \xrightarrow{\text{run}_2,\theta}_{\text{sg}} S_2^-$ and let us observe S_1^-. By Definition 12, there are three possible values for S_1^- which are all simulated by S_2^-. Indeed, keeping in mind that, $\text{run}_1 = \text{run}_2$, $S_1' \preccurlyeq_{idle} S_2'$ is already established. Then, it is easy to verify that $\text{sigCmp}_{S_1'}(\text{run}_1) \preccurlyeq_{idle} \text{sigCmp}_{S_2'}(\text{run}_2)$ and $\text{critUp}_{S_1'}(\text{run}_1) \preccurlyeq_{idle} \text{critUp}_{S_2'}(\text{run}_2)$.

It remains to prove that *if* $S_1 \preccurlyeq_{idle} S_2$ and $S_1 \in \text{DeadlineMiss}(\tau)$, *then* $S_2 \in \text{DeadlineMiss}(\tau)$ too. Let τ_i be such that $\text{rct}_{S_1}(\tau_i) > 0 \wedge \text{ttd}_{S_1}(\tau_i) \leq 0$. Since $S_1 \preccurlyeq_{idle} S_2 : \text{rct}_{S_2}(\tau_i) = \text{rct}_{S_1}(\tau_i)$ and $\text{nat}_{S_2}(\tau_i) \leq \text{nat}_{S_1}(\tau_i)$, thus $\text{ttd}_{S_2}(\tau_i) \leq \text{ttd}_{S_1}(\tau_i)$. Hence $\text{rct}_{S_2}(\tau_i) > 0 \wedge \text{ttd}_{S_2}(\tau_i) \leq \text{ttd}_{S_1}(\tau_i) \leq 0$ and therefore $S_2 \in \text{DeadlineMiss}(\tau)$ as per Definition 7. □

Figure 1, presented in section 3, illustrates the effect of $\preccurlyeq_{idle}$ with Algorithm 1. If a state S_2 has been encountered previously, and we find another state S_1 s.t. $S_1 \preccurlyeq_{idle} S_2$, then we can avoid exploring S_1 and its successors. However, this does not mean never encountering a successor of S_1 as they may be encountered through other paths, or have been encountered already. In Figure 1, grey states can be avoided as they are simulated by another state: we have $\text{HI}[01, 00] \preccurlyeq_{idle} \text{HI}[00, 00]$ and, $\text{LO}[00, 01]$ and $\text{LO}[01, 00]$ are simulated by $\text{LO}[00, 00]$.

5.2 Safe state oracles

We present a safe state oracle, which is a sufficient schedulability condition depending on the state of the system.

Oracle 1. HI idle point.

$$\{S \mid \text{cri}_S = \text{HI} \wedge \text{Active}(S) = \emptyset\} \subseteq \textit{Safe}.$$

Proof. As per the *"due diligence"* outlined in section 2, the task set comprising only HI tasks, where $\forall \tau_i : C_i = C_i(\text{HI})$, must be schedulable. Hence, as of reaching an idle point in HI mode, no more deadline misses are possible. □

5.3 Unsafe state oracles

In this section, we present several *necessary* conditions depending on the states of the system and provide their equivalent formulation in terms of unsafe oracles. Note that the oracles presented below are scheduler-agnostic.

Definition 17 (Laxity). *The* **laxity** *of an active task τ_i in the state S is:* $\text{laxity}_S(\tau_i) = \text{ttd}_S(\tau_i) - \text{rct}_S(\tau_i)$.

Lemma 1. *For feasibility it is* necessary *to have* $\forall \tau_i \in \text{Active}(\tau) : \text{laxity}_S(\tau_i) \geq 0$.

Proof. It is obvious that if $\text{ttd}_S(\tau_i) < \text{rct}_S(\tau_i)$, then even if the processor is given to the job immediately until its deadline, we will miss its deadline. □

Oracle 2. Negative laxity.

$$\{S \mid \exists \tau_i \in \text{Active}(\tau) : \text{laxity}_S(\tau_i) < 0\} \subseteq \textit{Unsafe}.$$

Proof. It holds trivially from Lemma 1 that any state S violating the necessary condition will lead to a deadline miss. □

In LO mode, we introduce a *stronger* condition that anticipates an imminent mode change. The *worst-laxity* of a LO task is simply its laxity. However, for a HI task, we incorporate its "bonus" execution time $(C_i(\text{HI}) - C_i(\text{LO}))$ that would be incurred if it does not explicitly signal completion, eventually causing a mode change.

Definition 18 (Worst-laxity). *The* **worst-laxity** *of an active task τ_i in state S is:* $\text{worstLaxity}_S(\tau_i) = \text{laxity}_S(\tau_i) - (C_i(L_i) - C_i(\text{cri}_S))$.

Lemma 2. *$\forall \tau_i$, it is* necessary *to have* $\text{worstLaxity}_S(\tau_i) \geq 0$.

Proof. In HI mode the laxity is equivalent to the notion of worst-laxity, the property follows from Lemma 1. In LO mode we have to distinguish between LO and HI tasks. For a LO task it is necessary to have non negative laxity. For a HI task if this condition is not satisfied and if a mode change occurs *simultaneously*, by definition the remaining computation time of the active job of τ_i is larger than the time we have before reaching the deadline. Therefore, we will inevitably miss the task deadline, which proves the property. □

Oracle 3. Negative worst-laxity.

$$\{S \mid \exists \tau_i \in \text{Active}(\tau) : \text{worstLaxity}_S(\tau_i) < 0\} \subseteq \textit{Unsafe}.$$

Proof. It holds trivially from Lemma 2 that any state S violating the necessary condition will lead to a deadline miss. □

For the next oracle, we propose to adapt the reasoning of demand bound functions (dbf) — as leveraged for EDF and its variations in many prior works [10] — to formulate a demand function related to the state of the system — i.e., the cri, rct and nat information. Instead of defining, per-task τ_i, a demand function between two *absolute* instants t_1 and t_2, we define a demand function *relative* to the current state, with a single parameter t which is a time in the future *relative* to the "current time" (or current instant) in S.

We need to separate concerns between current jobs — that can be disabled (LO case) or extended (HI case) — and future jobs. We define the number of future jobs of a task, i.e., the number of jobs the task can release until this future instant.

Definition 19 (Number of future jobs of a task until a future instant). *the* **maximum number of jobs** *that τ_i can release* strictly *after the current instant in S and before a future instant t, and that have a deadline no later than t, assuming S is in mode α, is:*

$$\mathsf{nj}_S^\alpha(\tau_i, t) = \begin{cases} 0 & \text{if } L_i < \alpha \\ \left\lfloor \frac{\max\{t - \mathsf{ttd}_S(\tau_i), 0\}}{T_i} \right\rfloor & \text{otherwise.} \end{cases}$$

The first case represents the fact that no job can be released by a LO task when assuming HI mode. The second case is the amount of periods that fits within the interval from the deadline of the current job and t. Based on that, we can define the demand function.

Definition 20 (Demand function of a task in a state). *The* **demand function** *of a task τ_i in state S in mode α is a lower bound on the maximum amount of computation time required by jobs of τ_i between the current instant in S and a future instant t, only for jobs of τ_i having deadlines before t, assuming S is in mode α or that the mode α will be reached at the next possible instant. Formally, with $\alpha \in \{\mathsf{LO}, \mathsf{HI}\}$:* $\mathsf{df}_S^\alpha(\tau_i, t)$

$$= \begin{cases} 0 & \text{if } t < \mathsf{ttd}_S(\tau_i) \vee L_i < \alpha \\ \mathsf{nj}_S^\alpha(\tau_i, t) \cdot C_i(\alpha) & \text{else if } \mathsf{rct}_S(\tau_i) = 0 \\ \mathsf{nj}_S^\alpha(\tau_i, t) \cdot C_i(\alpha) + C_i(\alpha) & \\ \qquad - C_i(\mathsf{cri}_S) + \mathsf{rct}_S(\tau_i) & \text{otherwise.} \end{cases}$$

No work must be accounted in the demand $\mathsf{df}_S^\alpha(\tau_i, t)$ if t is before the deadline of τ_i or if the criticality of τ_i is below the considered mode α. Otherwise, if τ_i is idle, we only consider the demand of *future* jobs, and do not account for the "bonus" execution time ($C_i(\mathsf{HI}) - C_i(\mathsf{LO})$) a running job of a HI task receives if a mode switch occurs. This is accounted in the third case, together with the remaining time ($\mathsf{rct}_S(\tau_i)$) of the current job. Notice that $\mathsf{nj}_S^{\mathsf{cri}_S}(\tau_i, \mathsf{ttd}_S(\tau_i)) = 0$, so $\mathsf{df}_S^{\mathsf{cri}_S}(\tau_i, \mathsf{ttd}_S(\tau_i)) = \mathsf{rct}_S(\tau_i)$. We can now define the global demand bound.

Definition 21 (Demand bound function in a state). *The global* **demand bound function** *is the total maximum amount of computation time required between the current instant in S and a future instant t by jobs of the task set τ, only for jobs having deadlines before t, assuming mode α. Formally:* $\mathsf{dbf}_S^\alpha(t) = \sum_{\tau_i \in \tau} \mathsf{df}_S^\alpha(\tau_i, t)$.

The definitions of nj, df and dbf uses the mode parameter α. This allows us to consider the current mode when we set $\alpha = \mathsf{cri}_S$ and to anticipate the demand if a mode switch occurs when we set $\alpha = \mathsf{HI}$.

Lemma 3. *For feasibility it is* necessary *to have* $\forall \tau_i \in \mathsf{Active}(\tau) : \mathsf{ttd}_S(\tau_i) \geq \mathsf{dbf}_S^{\mathsf{cri}_S}(\mathsf{ttd}_S(\tau_i))$.

Oracle 4. Over demand.

$\{S \mid \exists \tau_i \in \mathsf{Active}(\tau) : \mathsf{ttd}_S(\tau_i) < \mathsf{dbf}_S^{\mathsf{cri}_S}(\mathsf{ttd}_S(\tau_i))\} \subseteq$ *Unsafe.*

Proof. For any $\tau_k \in \tau$, $\mathsf{nj}_S^{\mathsf{cri}_S}(\tau_k, \mathsf{ttd}_S(\tau_i))$ is the highest possible ℓ such that $\mathsf{nat}_S(\tau_k) + D_k + (\ell - 1) \cdot T_k \leq \mathsf{ttd}_S(\tau_i)$, showing it is the highest possible number of jobs released by τ_k between the current instant in S and the next deadline of τ_i having a deadline before $\mathsf{ttd}_S(\tau_i)$. Note that $\mathsf{df}_S^{\mathsf{cri}_S}(\tau_k, \mathsf{ttd}_S(\tau_i))$ represents the total amount of execution time required by τ_k including its potentially active job and all its future jobs until $\mathsf{ttd}_S(\tau_i)$, assuming no mode change. Consequently, $\mathsf{dbf}_S^{\mathsf{cri}_S}(\mathsf{ttd}_S(\tau_i))$ is the amount of computation time required by all jobs of the system with deadline $\leq \mathsf{ttd}_S(\tau_i)$. This quantity corresponds to the work that must be scheduled before $\mathsf{ttd}_S(\tau_i)$ without mode switch, so it is necessary, in a uniprocessor system, that this quantity does not exceed $\mathsf{ttd}_S(\tau_i)$. □

The above result does not anticipate a mode switch (i.e., $\alpha = \mathsf{cri}_S$), it gives the necessary condition foreseeing that no mode switch occurs. Using a similar construct but setting $\alpha = \mathsf{HI}$, a necessary condition that foresees a mode switch can be derived as follows.

Lemma 4. *For feasibility it is* necessary *to have* $\forall \tau_i \in \mathsf{Active}(\tau) : \mathsf{ttd}_S(\tau_i) \geq \mathsf{dbf}_S^{\mathsf{HI}}(\mathsf{ttd}_S(\tau_i))$.

This necessary condition guarantees that all HI-tasks in S will have enough time to execute all the rct of both their current and future jobs, with their deadline prior to those of any active tasks' jobs, under the assumption that no job explicitly signals completion, eventually triggering a mode switch at the next possible instant. Note that this necessary condition optimistically assumes that no future computing time would be wastefully given to a LO task until $\mathsf{ttd}_S(\tau_i)$. In practice, should a LO task be scheduled, then $\mathsf{dbf}_S^{\mathsf{HI}}(\mathsf{ttd}_S(\tau_i))$ would remain constant, whereas $\mathsf{ttd}_S(\tau_i)$ will decrease, potentially violating this necessary condition.

Oracle 5. HI over demand.

$\{S \mid \exists \tau_i \in \mathsf{Active}(\tau) : \mathsf{ttd}_S(\tau_i) < \mathsf{dbf}_S^{\mathsf{HI}}(\mathsf{ttd}_S(\tau_i))\} \subseteq$ *Unsafe.*

Proof. If $\mathsf{cri}_S = \mathsf{HI}$, then the Lemma 4 is equivalent to Lemma 3, and the proof provided for Lemma 5.3 holds. If $\mathsf{cri}_S = \mathsf{LO}$, we consider two cases, one where a mode switch, triggered by a HI-task $\tau_k \in \{\tau \mid L_k = \mathsf{HI}\}$, can occur before $\mathsf{ttd}_S(\tau_i)$, and the other where it may not. For a (current or future) job of τ_k to trigger a mode switch, it is sufficient to have $\mathsf{rct}(\tau_k) > 0 \wedge \mathsf{ttd}_S(\tau_i) \geq \mathsf{ttd}_S(\tau_k)$ or $\mathsf{rct}(\tau_k) = 0 \wedge \mathsf{ttd}_S(\tau_i) \geq \mathsf{ttd}_S(\tau_k) + T_k$. In such cases, we can anticipate the mode switch and enforce $\forall \tau_i \in \mathsf{Active}(\tau) : \mathsf{ttd}_{S'}(\tau_i) \geq \mathsf{dbf}_{S'}^{\mathsf{cri}_{S'}}(\mathsf{ttd}_{S'}(\tau_i))$ as a necessary condition as per Lemma 3, with $S' = \mathsf{critUp}_S(\bot)$. Indeed be noted that $\mathsf{critUp}_S(\bot)$ merely increases the rct of the active HI-jobs and discards the LO tasks. Therefore, if anything, this anticipation is more optimistic since it forestalls the wasteful scheduling of LO-jobs. Furthermore, given $\mathsf{cri}_{S'}$=HI, we can express $\mathsf{dbf}_{S'}^{\mathsf{cri}_{S'}}(\mathsf{ttd}_{S'}(\tau_i))$ as $\mathsf{dbf}_{S'}^{\mathsf{HI}}(\mathsf{ttd}_{S'}(\tau_i))$, so if this inequality does not hold, S' is deemed unsafe and so is S as S'

is more optimistic. If a mode switch occurrence cannot be guaranteed before $\text{ttd}_S(\tau_i)$, i.e., when $\text{rct}(\tau_k) > 0 \wedge \text{ttd}_S(\tau_i) < \text{ttd}_S(\tau_k)$ or $\text{rct}(\tau_k) = 0 \wedge \text{ttd}_S(\tau_i) < \text{ttd}_S(\tau_k) + T_k$, then $\text{df}_S^{\text{HI}}(\tau_k, \text{ttd}_S(\tau_i)) = 0$ per definition. Moreover, for LO-tasks $\tau_j \in \{\tau \mid L_j = \text{LO}\}$, $\text{df}_S^{\text{HI}}(\tau_j, \text{ttd}_S(\tau_i)) = 0$, and $\text{ttd}_S(\tau_i) \geq 0 = \text{dbf}_S^{\text{HI}}(\text{ttd}_S(\tau_i))$ per definitions. Thus, under these circumstances, this oracle's inequality never holds and S is never deemed unsafe by it.

□

6 Evaluation

To demonstrate the benefits of the search space reduction and the accuracy of the exact test, we developed a C++ tool [36] that takes a task set and scheduling policy as inputs, converts it into an automaton as in section 3, and explores it according to Algorithm 1, allowing to assess the schedulability of the task set. The same task set and scheduling policy can be explored with different optimisations (simulation relation and oracles), resulting in the same outcome (safe or unsafe) but with different time and space complexity. Experiments were run on a server computer with 128 GB of RAM and a 128-core AMD Ryzen Threadripper 3990X CPU running at 2.9GHz.

6.1 Random task set parameter generation

Inspired by the works of Baruah [8] and Ekberg [18], the task sets used in our experiments were randomly generated using a Python tool [36] as follows: T_i is an integer number generated according to a log-uniform distribution from the range of $[T^{\min}, T^{\max}]$; $L_i = \text{HI}$ with probability $0 \leq P_{\text{HI}} \leq 1$, otherwise $L_i = \text{LO}$, and $D_i = T_i$. The generation of C_i values depends on a target average utilisation $U^* \in [0, 1]$. Then δ is a `float` drawn from the uniform distribution $\mathcal{U}(-\mu, \mu)$ with $\mu = \min\{U^*, 1 - U^*\}$. The target LO-criticality utilisation of tasks, $U^{*\text{LO}}$, are generated using the Dirichlet-Rescale (DRS) algorithm [24], that gives an unbiased distribution of utilisation values. We give the following inputs: n the number of tasks, $U = U^* + \delta$, $u_i^{\min} = 1/T_i$ and $u_i^{\max} = 1$ for $1 \leq i \leq n$. Similarly, the target HI-criticality utilisations of tasks, $U^{*\text{HI}}$, are generated using DRS too with, as inputs, the same number of tasks n, $U = U^* - \delta$, $u_i^{\min} = U^{*\text{LO}}(\tau_i)$ and $u_i^{\max} = 1$ if $L_i = \text{HI}$, $u_i^{\min} = 0 = u_i^{\max}$ otherwise, for $1 \leq i \leq n$. $C_i(\text{LO}) = U^{*\text{LO}}(\tau_i) \cdot T_i$ rounded to the nearest integer. $C_i(\text{HI}) = U^{*\text{HI}}(\tau_i) \cdot T_i$ rounded to the nearest integer if $L_i = \text{HI}$ and $C_i(\text{HI}) = C_i(\text{LO})$ otherwise. Task sets were dropped if $U^{\text{LO}}(\tau) > 1$ or $U^{\text{HI}}(\tau) > 1$. Duplicate task sets were discarded, as were task sets with all tasks sharing the same criticality level. Additionally, task sets were dropped if $\mid U^{\text{avg}}(\tau) - U^* \mid > 0.005$. Deadlines are implicit because our explorations will use among others EDF-VD (see Definition 9) which can only cope with such deadlines [9]. Hereunder, a range of numbers from f to t with a step increase of s will be denoted by $[f; t; s]$.

6.2 Antichain impact on state space exploration

Figure 2 compares the performance of antichain breadth first search using the idle tasks simulation relation (ACBFS) and using classical breadth first search (BFS) with EDF-VD and without any oracle. Task sets were generated with varying number of tasks and maximum periods, the rest of the parameters being fixed. For each combination of parameters, 10 sets were generated. All task sets

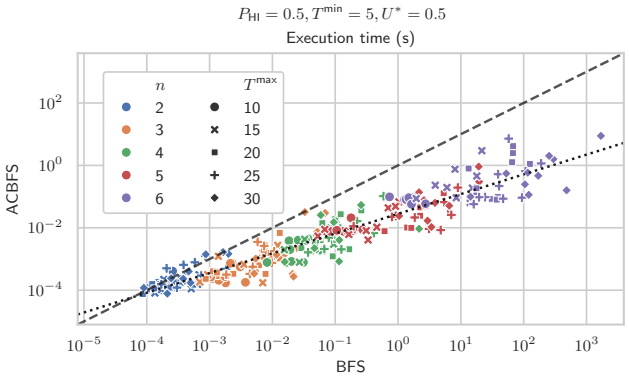

Figure 2: Execution times in seconds before halt for BFS and ACBFS. Dashed line is $x = y$, dotted line is the linear regression on the samples (after $\log_{10}$).

were schedulable, hence, their automaton was fully developed as every single state reachable from the initial state was visited (or simulated). ACBFS outperforms BFS in execution time, except for the few cases where the search space is very small — in such cases, both algorithms ran for less than 0.01 second. This shows that ACBFS and the idle tasks simulation relation scale better than BFS, reducing the exploration time by at least one order of magnitude. As the state space size increases, the execution time gap increases, as indicated by the regression line. Results on the number of states visited for each of the algorithms are on par, but analysing their execution time takes into account the (potential) extra computing time of the antichains.

Figure 3(a) illustrates the scalability of ACBFS with a varying quantity of tasks and maximum period. Both parameters contribute to prolonging the exploration time, but the number of tasks exerts a stronger and more consistent effect. The automaton's size being on the order of $O((T^{\max})^n)$ explains this stronger impact for the

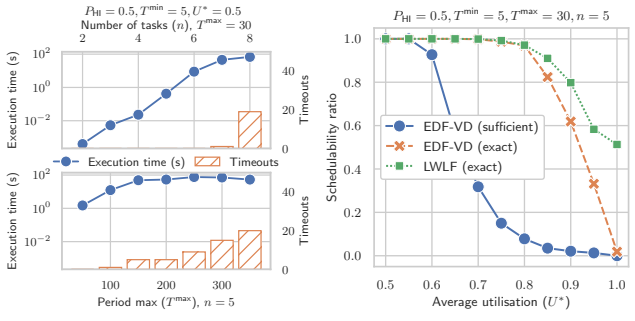

Figure 3: (a) ACBFS' median execution times over 40 explorations before halt, with varying number of task and maximum period and a 15 minutes timeout.
(b) Schedulability ratio over average utilisation per test. Exploration with ACBFS and HI over demand for exact tests.

number of tasks. The period's effect is less stable, given that the $T^{\max}$ parameter serves as an upper bound to the distribution from which the period is drawn; as such, even with a high $T^{\max}$ value, the periods can still be set at lower levels. The majority of task sets, composed of 8 tasks, were successfully explored within the allocated 15-minute timeout. As for the maximum period parameter, timeouts were observed as early as $T^{\max} = 100$. Nearly a third of the task sets with $T^{\max} = 350$ exceeded the timeout, suggesting that specific combinations of periods can lead to a larger state space.

While the antichain optimization enables a reduction in the state space, the problem remains challenging and experiences exponential growth, especially considering that it deals with sporadic tasks. Most of the task sets with $n = 7$ and some with $n = 8$ can be explored within an acceptable timeframe, as ACBFS can manage at least one more task than BFS (see Figure 2). Beyond, both the execution time and required amount of memory (the maximal explored states are stored as per Algorithm 1) to explore some of the task sets become prohibitive. In comparison, other papers providing an exact test for (single-criticality) *sporadic* tasks sets with FJP schedulers experimented with task sets parameters up to $T^{\max} = 6$ [5], $n = 8, T^{\max} = 8$ [21] and $n = 4, T \in \{1, 2, 5, 10, 20, 50, 100, 200, 1000\}$ [46]. In real-world situations, certifying a task set requires a single exploration; in those cases, the exploration time could extend longer than in our experiments, allowing to increase task number or periods.

6.3 Oracle impact

To measure the impact of the oracles, 2100 task sets were generated with $T^{\min} = 5, T^{\max} = 30, P_{\mathsf{HI}} = 0.5, n = 5$ and $U^* \in [0.8; 1; 0.01]$ with 100 task sets for each values of U^*. Figure 3(a) shows our simulator is capable of accommodating larger systems. The parameters here selected expedite the exploration process, thereby enhancing the quantity of results and facilitating the accrual of more robust, aggregated metrics. All task sets were explored with ACBFS and EDF-VD, several times with different oracles. As baseline, an exploration with ACBFS without any oracle is used. During those explorations, 923 task sets were found to be unschedulable. Thus, about half of the task sets are schedulable, obtained over several (omitted) iterations of the experiments to tune the range of utilisations. We expect the impacts of oracles to differ depending on the actual schedulability of the evaluated task sets. For example, unsafe oracles computed on schedulable task sets will not reduce the search space and will only constitute overheads — this, however, cannot be known a priori by the system designer. For each oracle, Figure 4 shows the avoided state ratio and speed-up ratio between ACBFS used with and without the oracle. The reported values are median over all explorations, split over task sets schedulability.

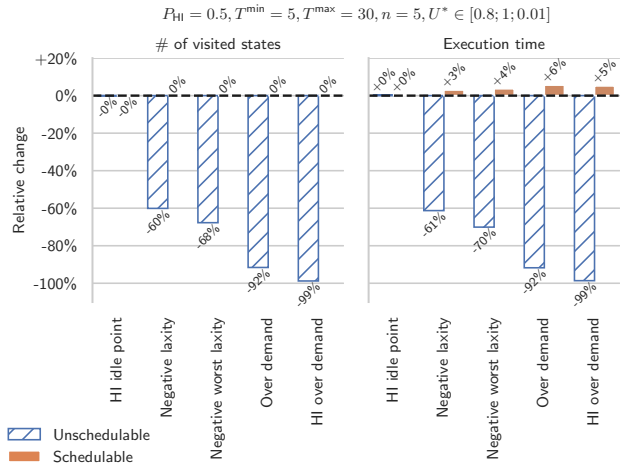

Figure 4: Oracles' impact on the number of states visited and the execution time.

The HI idle point safe oracle barely reduces the number of visited states for schedulable and unschedulable task sets (0.02% and 0.07%). The execution time increases are both 0.4%, reflecting the additional computing cost of the oracle.

The first unsafe oracle, negative laxity, already allows avoiding 60.1% of the states, leading to an execution time reduction of 61.2% for unschedulable task set, while increasing the execution time by 3.0% of the schedulable ones — computing the laxity only requires $O(n)$ operations on each visited state.

Then, the negative worst laxity unsafe oracle builds on top of the laxity and brings another level of impact. 67.7% of the states are dropped, translating to a time execution reduction of 70.1% for unschedulable task sets, while only having a median time execution increase of 3.6% on schedulable task sets. This significant impact comes from the anticipation of what will happen in HI mode when still in LO mode, hence enabling to detect a deadline miss *earlier*.

The over demand oracle computes the current mode demand. This approach brings a step change as it avoids 91.6% of the states and reduces the execution time by 91.7% for unschedulable task sets, while increasing the execution time of the schedulable task sets by 5.6%. While strong, it is more expensive than the other unsafe oracles, as shown by the increased time with schedulable task sets.

Finally, the HI over demand oracle combines both worlds by analysing all tasks and anticipating the demand at the next criticality level. This approach is the strongest, dropping 98.8% of states, reducing the execution time by 98.6% for unschedulable task sets and increasing the execution time by 5.2% on schedulable task sets.

To summarise the combined impacts of antichains with idle tasks simulation and the best oracle (HI over demand), Table 2 outlines statistics. Generation parameters are the same as the oracle impact experiment, Figure 4, except for $T^{\max} = 20$, resulting in balanced schedulability. The table shows statistics about the composed effect of both the antichains and the best oracle, achieving up to 99.998% reduction in the searched state space (2968037 visited states for BFS against 42 for ACBFS with the oracle).

Table 2: Statistics on the number of visited states.

$P_{\mathsf{HI}} = 0.5, T^{\min} = 5, T^{\max} = 20, n = 5, U^* \in [0.8; 1; 0.01]$

Search	BFS		ACBFS	
Oracle	None	HI over demand	None	HI over demand
min	9905	35 (-100%)	668 (-93%)	29 (-100%)
mean	746974	480688 (-36%)	75374 (-90%)	46024 (-94%)
std	999984	819378 (-18%)	126110 (-87%)	107731 (-89%)
median	410063	217928 (-49%)	35888 (-91%)	15459 (-96%)
max	11875126	11875126 (-0%)	2687577 (-77%)	2687577 (-77%)

6.4 Scheduling analysis

Our framework enabled us to step into scheduling experimentations, aiming to understand the real performance of EDF-VD. Specifically, automata were explored with ACBFS, the HI over demand unsafe oracle and without safe oracles. Task sets were generated with $T^{\min} = 5, T^{\max} = 30, P_{\mathsf{HI}} = 0.5, n = 5$ and $U^* \in [0.5; 1; 0.05]$. For each utilisation, 1000 task sets were generated. Figure 3(b) shows the average schedulability ratio for each target average utilisation. It shows the schedulability ratio of EDF-VD's sufficient test [9], EDF-VD exact test using our automaton exploration approach, and of a novel algorithm *Least Worst Laxity First* (LWLF).

Definition 22 (LWLF scheduler). *The Least Worst Laxity First scheduler, or LWLF, is defined as follows. We let* $\mathsf{min}_{\mathsf{wl}}(S)$ *be the task* $\tau_i \in \mathsf{Active}\,(S)$ *which has the minimal worst laxity in* S*. That is,* τ_i *is s.t.* $\forall \tau_k \in \mathsf{Active}\,(S) \setminus \{\tau_i\}$: $\mathsf{worstLaxity}_S(\tau_k) > \mathsf{worstLaxity}_S(\tau_i)$ *or* $\mathsf{worstLaxity}_S(\tau_k) = \mathsf{worstLaxity}_S(\tau_i) \wedge k > i$*. Then, for all states* S*, we let:*

$$\mathsf{sch}_{LWLF}(S) = \begin{cases} \bot & \text{if } \mathsf{Active}\,(S) = \emptyset \\ \mathsf{min}_{\mathsf{wl}}(S) & otherwise. \end{cases}$$

EDF-VD's actual schedulability significantly outperforms its sufficient test, underscoring the test's pessimism. Notice, however, that this exact test — even with antichains and oracles — can be a time-intensive process, whereas the EDF-VD's sufficient test is computed in polynomial time. For large values of n, the exact test might be impractical, so sufficient tests remain relevant. One can always first run a sufficient test, and if negative, try the exact analysis.

LWLF brings further schedulability significant improvements over EDF-VD. We derived LWLF after observing the strong impact of the worst laxity oracle — since it can detect a failure earlier, it hinted the existence of a scheduling algorithm integrating this information to guide the scheduling decision. Similarly, we believe that a scheduling algorithm building on top of the HI over demand oracle can be derived and is expected to bring even better results.

7 Related Work

There is extensive related research on real-time scheduling. Below, we categorize and summarize the most relevant prior work to ours.

Scheduling tests. Utilisation based conditions were produced, both sufficient [9] and necessary [41], the latter also adapting them for multiprocessor. Demand-bound function sufficient schedulability tests have also been produced, for dual-criticality [13, 16], then extended to any number of criticalities [25] and to cope with deadline reduction on mode change [18].

Formal verification for mixed-criticality scheduling. Burns and Jones used time bands and rely-guarantee conditions [12, 26]. Inspired by Hoare's logic, they proposed a framework to formally specifying the temporal behaviour of schedulers, hence allowing the verification of schedulers and their code generation. Abdeddaïm [1] provided exact schedulability tests for FJP schedulers for a more modular mixed criticality model, where only specified subsets of LO (sporadic) tasks are discarded in HI mode depending on the subset of HI tasks exceeding their LO budget.

Using automata to check schedulability. Already discussed in our introduction (see section 1).

Antichains. The *antichain technique* has been introduced by De Wulf *et al.* [45] to improve the practical performances of costly algorithms on automata. These ideas have later been applied to produce a series of efficient algorithms for automata [2, 14], model-checking [15] and games [20, 22, 40].

Schedule-Abstraction Graphs (SAG). Another reachability-based response-time analysis was developed for an exact test on sets of non-preemptive jobs with release jitter and variable execution time with a FJP scheduler upon a uniprocessor [30], then upon multiprocessor [31], with preemption [23], and several kinds of inter-job constraints [32–34]. Applying partial-order reduction rules into SAG mitigated combinatorial explosion [38, 39]. Notice a sequence of jobs — the expected input of SAG methods — *cannot support* the sporadic task model. By definition, the set of jobs is not known in advance, which is not the case in systems that are periodic or with release jitters.

8 Conclusion

We have developed a generic framework for exact schedulability assessment in uniprocessor mixed-criticality systems, by reducing it to the safety problem in an automaton. We combined insights from the formal verification community — like antichains, simulation relations and the ACBFS algorithm — and from the real-time research — like safe and unsafe oracles — to make the approach more practical. We demonstrated in simulations that those allow to reduce the search space by up to 99.998%, enabling the evaluation of existing and new scheduling algorithms, EDF-VD and LWLF, and comparing it to a prior sufficient test [9]. Our experimental evaluation also quantifies the advantages and the limits of the antichain technique.

Future work. Although the scalability of our approach remain limited (up to 8 tasks), our algorithm is generic, allowing one to incorporate a range of other optimisations, and can be applied to other scheduling problems. Our work can be extended to consider *more realistic* mixed criticality models [11], e.g., regarding LO-criticality tasks that should not be abandoned callously. We then need to modify the completion transition, in order to let LO-tasks run until completion or adjust the period and deadline of certain LO-tasks. A *multi-processor* version of the model is already supported in prior work for sequential jobs [21, 29]; we would have to combine it with the extensions proposed in this work. In addition, the reduction can be adapted to support other platforms and task models such as modelling *varying CPU speed* and *job-level parallelism* (e.g., the gang model), by reducing the rct values according to the CPU speed and the number of CPU cores assigned to the job. The $\mathsf{sch}(S)$ function would return the selected CPU speed factor together with how many CPUs are assigned to each job. A power function could then be used to label the clock-tick transitions with numeric values, hinting shortest path exploration (instead of breadth-first search) to minimize power consumption. Supporting the *DAG task model* and *federated scheduling* would require the schedule function to account for precedence constraints. To account for *preemption delays*, the model could force idle CPU time by forcing $\mathsf{run} = \bot$ for a given number of run transitions when the scheduler is switching to a different task across calls.

References

[1] Yasmina Abdeddaïm. 2020. Accurate Strategy for Mixed Criticality Scheduling. In *Verification and Evaluation of Computer and Communication Systems*, Belgacem Ben Hedia, Yu-Fang Chen, Gaiyun Liu, and Zhenhua Yu (Eds.). Springer International Publishing, Cham, 131–146.

[2] Parosh Aziz Abdulla, Yu-Fang Chen, Lukáš Holík, Richard Mayr, and Tomáš Vojnar. 2010. When Simulation Meets Antichains. In *Tools and Algorithms for the Construction and Analysis of Systems*, Javier Esparza and Rupak Majumdar (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 158–174.

[3] Kunal Agrawal and Sanjoy Baruah. 2018. Intractability Issues in Mixed-Criticality Scheduling. In *30th Euromicro Conference on Real-Time Systems (ECRTS 2018) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 106)*, Sebastian Altmeyer (Ed.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 11:1–11:21. https://doi.org/10.4230/LIPIcs.ECRTS.2018.11

[4] Sedigheh Asyaban and Mehdi Kargahi. 2018. An exact schedulability test for fixed-priority preemptive mixed-criticality real-time systems. *Real-Time Systems* 54 (2018), 32–90.

[5] Theodore P. Baker and Michele Cirinei. 2007. Brute-Force Determination of Multiprocessor Schedulability for Sets of Sporadic Hard-Deadline Tasks. In *Principles of Distributed Systems*, Eduardo Tovar, Philippas Tsigas, and Hacène Fouchal (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 62–75.

[6] Sanjoy Baruah. 2016. The Federated Scheduling of Systems of Mixed-Criticality Sporadic DAG Tasks. In *2016 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, Porto, Portugal, 227–236. https://doi.org/10.1109/RTSS.2016.030

[7] Sanjoy Baruah. 2016. Schedulability analysis of mixed-criticality systems with multiple frequency specifications. In *Proceedings of the 13th International Conference on Embedded Software* (Pittsburgh, Pennsylvania) *(EMSOFT '16)*. Association for Computing Machinery, New York, NY, USA, Article 24, 10 pages. https://doi.org/10.1145/2968478.2968488

[8] S.K. Baruah, A. Burns, and R.I. Davis. 2011. Response-Time Analysis for Mixed Criticality Systems. In *2011 IEEE 32nd Real-Time Systems Symposium*. IEEE, Vienna, Austria, 34–43. https://doi.org/10.1109/RTSS.2011.12

[9] Sanjoy K. Baruah, Vincenzo Bonifaci, Gianlorenzo D'Angelo, Alberto Marchetti-Spaccamela, Suzanne van der Ster, and Leen Stougie. 2011. Mixed-Criticality Scheduling of Sporadic Task Systems. In *Algorithms – ESA 2011*, Camil Demetrescu and Magnús M. Halldórsson (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 555–566.

[10] Sanjoy K Baruah, Louis E Rosier, and Rodney R Howell. 1990. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-time systems* 2, 4 (1990), 301–324.

[11] Alan Burns and Robert Ian Davis. 2022. *Mixed Criticality Systems - A Review (13th Edition, February 2022)*. White Rose ePrints repository, Yorkshire, UK. https://eprints.whiterose.ac.uk/183619/ This is the 13th version of this review now updated to cover research published up to the end of 2021..

[12] A. Burns and Cliff B. Jones. 2022. An Approach to Formally Specifying the Behaviour of Mixed-Criticality Systems. In *34th Euromicro Conference on Real-Time Systems (ECRTS 2022) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 231)*, Martina Maggio (Ed.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 14:1–14:23. https://doi.org/10.4230/LIPIcs.ECRTS.2022.14

[13] Yao Chen, Qiao Li, Zheng Li, and Huagang Xiong. 2014. Efficient schedulability analysis for mixed-criticality systems under deadline-based scheduling. *Chinese Journal of Aeronautics* 27, 4 (2014), 856–866.

[14] Laurent Doyen and Jean-François Raskin. 2010. Antichain Algorithms for Finite Automata. In *Tools and Algorithms for the Construction and Analysis of Systems, 16th International Conference, TACAS 2010, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2010, Paphos, Cyprus, March 20-28, 2010. Proceedings (Lecture Notes in Computer Science, Vol. 6015)*, Javier Esparza and Rupak Majumdar (Eds.). Springer, Paphos, Cyprus, 2–22. https://doi.org/10.1007/978-3-642-12002-2_2

[15] Laurent Doyen and Jean-Francois Raskin. 2009. Antichains for the Automata-Based Approach to Model-Checking. *Logical Methods in Computer Science* 5, 1 (3 2009), 1–20. https://doi.org/10.2168/lmcs-5(1:5)2009

[16] Pontus Ekberg and Wang Yi. 2012. Outstanding Paper Award: Bounding and Shaping the Demand of Mixed-Criticality Sporadic Tasks. In *2012 24th Euromicro Conference on Real-Time Systems*. IEEE, Pisa, Italy, 135–144. https://doi.org/10.1109/ECRTS.2012.24

[17] Pontus Ekberg and Wang Yi. 2014. Bounding and shaping the demand of generalized mixed-criticality sporadic task systems. *Real-Time Systems* 50, 1 (01 Jan 2014), 48–86. https://doi.org/10.1007/s11241-013-9187-z

[18] Pontus Ekberg and Wang Yi. 2014. Bounding and shaping the demand of generalized mixed-criticality sporadic task systems. *Real-time systems* 50 (2014), 48–86.

[19] Alexandre Esper, Geoffrey Nelissen, Vincent Nélis, and Eduardo Tovar. 2018. An industrial view on the common academic understanding of mixed-criticality systems. *Real-Time Systems* 54 (2018), 745–795.

[20] Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. 2011. Antichains and compositional algorithms for LTL synthesis. *Formal Methods Syst. Des.* 39, 3 (2011), 261–296. https://doi.org/10.1007/S10703-011-0115-3

[21] Gilles Geeraerts, Joël Goossens, and Markus Lindström. 2013. Multiprocessor schedulability of arbitrary-deadline sporadic tasks: Complexity and antichain algorithm. *Real-time systems* 49, 2 (2013), 171–218.

[22] Gilles Geeraerts, Joël Goossens, Thi-Van-Anh Nguyen, and Amélie Stainer. 2018. Synthesising succinct strategies in safety games with an application to real-time scheduling. *Theor. Comput. Sci.* 735 (2018), 24–49. https://doi.org/10.1016/J.TCS.2017.06.004

[23] Pourya Gohari, Jeroen Voeten, and Mitra Nasri. 2024. Reachability-Based Response-Time Analysis of Preemptive Tasks Under Global Scheduling. In *36th Euromicro Conference on Real-Time Systems (ECRTS 2024) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 298)*, Rodolfo Pellizzoni (Ed.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 3:1–3:24. https://doi.org/10.4230/LIPIcs.ECRTS.2024.3

[24] David Griffin, Iain Bate, and Robert I. Davis. 2020. Generating Utilization Vectors for the Systematic Evaluation of Schedulability Tests. In *IEEE Real-Time Systems Symposium, RTSS 2020, Houston, Texas, USA*. IEEE, Houston, Texas, USA, 76–88. http://eprints.whiterose.ac.uk/167646/

[25] Xiaozhe Gu and Arvind Easwaran. 2017. Efficient schedulability test for dynamic-priority scheduling of mixed-criticality real-time systems. *ACM Transactions on Embedded Computing Systems (TECS)* 17, 1 (2017), 1–24.

[26] Cliff B Jones and Alan Burns. 2021. A Rely-Guarantee Specification of Mixed-Criticality Scheduling. arXiv:2012.01493 [cs.SE] https://arxiv.org/abs/2012.01493

[27] Stephen Law, Iain Bate, and Benjamin Lesage. 2019. Industrial Application of a Partitioning Scheduler to Support Mixed Criticality Systems. In *31st Euromicro Conference on Real-Time Systems (ECRTS 2019) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 133)*, Sophie Quinton (Ed.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 8:1–8:22. https://doi.org/10.4230/LIPIcs.ECRTS.2019.8

[28] Haohan Li and Sanjoy Baruah. 2012. Outstanding Paper Award: Global Mixed-Criticality Scheduling on Multiprocessors. In *2012 24th Euromicro Conference on Real-Time Systems*. IEEE, Pisa, Italy, 166–175. https://doi.org/10.1109/ECRTS.2012.41

[29] Markus Lindström, Gilles Geeraerts, and Joël Goossens. 2011. A faster exact multiprocessor schedulability test for sporadic tasks. 9th International Conference on Real-Time and Network Systems, RTNS '11..

[30] Mitra Nasri and Bjorn B. Brandenburg. 2017. An Exact and Sustainable Analysis of Non-preemptive Scheduling. In *2017 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, Paris, France, 12–23. https://doi.org/10.1109/RTSS.2017.00009

[31] Mitra Nasri, Geoffrey Nelissen, and Björn B. Brandenburg. 2018. A Response-Time Analysis for Non-Preemptive Job Sets under Global Scheduling. In *30th Euromicro Conference on Real-Time Systems (ECRTS 2018) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 106)*, Sebastian Altmeyer (Ed.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 9:1–9:23. https://doi.org/10.4230/LIPIcs.ECRTS.2018.9

[32] Mitra Nasri, Geoffrey Nelissen, and Björn B. Brandenburg. 2019. Response-Time Analysis of Limited-Preemptive Parallel DAG Tasks Under Global Scheduling. In *31st Euromicro Conference on Real-Time Systems (ECRTS 2019) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 133)*, Sophie Quinton (Ed.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 21:1–21:23. https://doi.org/10.4230/LIPIcs.ECRTS.2019.21

[33] Geoffrey Nelissen, Joan Marcè i Igual, and Mitra Nasri. 2022. Response-Time Analysis for Non-Preemptive Periodic Moldable Gang Tasks. In *34th Euromicro Conference on Real-Time Systems (ECRTS 2022) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 231)*, Martina Maggio (Ed.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 12:1–12:22. https://doi.org/10.4230/LIPIcs.ECRTS.2022.12

[34] Suhail Nogd, Geoffrey Nelissen, Mitra Nasri, and Björn B. Brandenburg. 2020. Response-Time Analysis for Non-Preemptive Global Scheduling with FIFO Spin Locks. In *2020 IEEE Real-Time Systems Symposium (RTSS)*. IEEE, Houston, Texas, USA, 115–127. https://doi.org/10.1109/RTSS49844.2020.00021

[35] Antonio Paolillo, Paul Rodriguez, Vladimir Svoboda, Olivier Desenfans, Joël Goossens, Ben Rodriguez, Sylvain Girbal, Madeleine Faugère, and Philippe Bonnot. 2017. Porting a safety-critical industrial application on a mixed-criticality enabled real-time operating system. Workshop on Mixed Criticality Systems.

[36] Simon Picard and Antonio Paolillo. 2024. *MC Graph Explorer*. GitHub. https://github.com/simonpicard/mixed-criticality-graph-xp/

[37] Simon Picard, Antonio Paolillo, Gilles Geeraerts, and Joël Goossens. 2024. Exact schedulability test for sporadic mixed-criticality real-time systems using antichains and oracles. arXiv:2410.18308 [cs.OS] https://arxiv.org/abs/2410.18308

[38] Sayra Ranjha, Pourya Gohari, Geoffrey Nelissen, and Mitra Nasri. 2023. Partial-order reduction in reachability-based response-time analyses of limited-preemptive DAG tasks. *Real-Time Systems* 59, 2 (01 Jun 2023), 201–255. https://doi.org/10.1007/s11241-023-09398-x

[39] Sayra Ranjha, Geoffrey Nelissen, and Mitra Nasri. 2022. Partial-Order Reduction for Schedule-Abstraction-based Response-Time Analyses of Non-Preemptive

Tasks. In *2022 IEEE 28th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE, Milano, Italy, 121–132. https://doi.org/10.1109/RTAS54340.2022.00018

[40] Jean-François Raskin, Krishnendu Chatterjee, Laurent Doyen, and Thomas A. Henzinger. 2007. Algorithms for Omega-Regular Games with Imperfect Information. *Log. Methods Comput. Sci.* 3, 3 (2007), 1–20. https://doi.org/10.2168/LMCS-3(3:4)2007

[41] J. Augusto Santos, George Lima, and Konstantinos Bletsas. 2015. Considerations on the Least Upper Bound for Mixed-Criticality Real-Time Systems. In *2015 Brazilian Symposium on Computing Systems Engineering (SBESC)*. IEEE, Foz do Iguacu, Brazil, 58–63. https://doi.org/10.1109/SBESC.2015.18

[42] Tianning She, Zhishan Guo, and Kecheng Yang. 2022. Scheduling Constrained-Deadline Tasks in Precise Mixed-Criticality Systems on a Varying-Speed Processor. In *Proceedings of the 30th International Conference on Real-Time Networks and Systems* (Paris, France) *(RTNS '22)*. Association for Computing Machinery, New York, NY, USA, 94–102. https://doi.org/10.1145/3534879.3534897

[43] Tianning She, Sudharsan Vaidhun, Qijun Gu, Sajal Das, Zhishan Guo, and Kecheng Yang. 2021. Precise Scheduling of Mixed-Criticality Tasks on Varying-Speed Multiprocessors. In *Proceedings of the 29th International Conference on Real-Time Networks and Systems* (NANTES, France) *(RTNS '21)*. Association for Computing Machinery, New York, NY, USA, 134–143.

[44] Steve Vestal. 2007. Preemptive Scheduling of Multi-criticality Systems with Varying Degrees of Execution Time Assurance. In *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*. IEEE, Tucson, Arizona, USA, 239–243. https://doi.org/10.1109/RTSS.2007.47

[45] Martin De Wulf, Laurent Doyen, Thomas A. Henzinger, and Jean-François Raskin. 2006. Antichains: A New Algorithm for Checking Universality of Finite Automata. In *Proceedings of Computer Aided Verification, 18th International Conference, CAV (Lecture Notes in Computer Science, Vol. 4144)*, Thomas Ball and Robert B. Jones (Eds.). Springer, Seattle, WA, USA, 17–30. https://doi.org/10.1007/11817963_5

[46] Beyazit Yalcinkaya, Mitra Nasri, and Björn B. Brandenburg. 2019. An Exact Schedulability Test for Non-Preemptive Self-Suspending Real-Time Tasks. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, Florence, Italy, 1228–1233. https://doi.org/10.23919/DATE.2019.8715111