Stability of Control Systems under Extended Weakly-Hard Constraints

Nils Vreman*, Paolo Pazzaglia†, Jie Wang‡, Victor Magron‡, Martina Maggio†*
* Department of Automatic Control, Lund University, Sweden, {nils.vreman}@control.lth.se
† Department of Computer Science, Saarland University, Germany, {pazzaglia, maggio}@cs.uni-saarland.de
‡ Laboratory for Analysis and Architecture of Systems CNRS, France, {vmagron, jwang}@laas.fr

***Abstract*—Control systems can show robustness to many events, like disturbances and model inaccuracies. It is natural to speculate that they are also robust to alterations of the control signal pattern, due to sporadic late completions (called *deadline misses*) when implemented as a digital task on an embedded platform. Recent research analysed stability when imposing constraints on the maximum number of consecutive deadlines that can be missed. This is only one type of characterization, and results in a pessimistic analysis when applied to more general cases. To overcome this limitation, this paper proposes a comprehensive stability analysis for control systems subject to a set of generic constraints, describing the possible sequences of correct completions and deadline misses. The proposed analysis brings the assessment of control systems robustness to computational problems one step closer to the controller implementation.**

I. Introduction

Robustness is an essential concern in the design of control systems [54]. Robust control design techniques ensure that controlled systems are able to reliably handle nonlinear effects, unmodeled dynamics, and external noise. However, existing methods rarely consider robustness to *computational issues*. This latter aspect is relevant for a vast number of control applications, which rely on the usage of digital controllers implemented as periodic *tasks* in cheap embedded platforms. Such tasks are expected to execute with real-time guarantees, meaning that their execution must be scheduled to complete before a well-defined *deadline*, usually corresponding to the activation of the next periodic instance of the task.

In the case of a digital controller, its execution consists in performing the control signal calculation and dealing with sensing and actuation. In practice, control systems are often designed introducing one step delay, i.e., the control signal computed during one control period is applied to the plant at the beginning of the next period. This helps masking the timing variability experienced using embedded platforms, that often have to dynamically share their limited computational resources among many concurrent functions. It is still possible – and it happens quite often in practice [1] – that tasks do not complete within their execution period, causing what is called a *deadline miss*. The reasons behind those misses can vary, from delays or non-delivered sensor messages, to transient overload conditions in the platform, to bugs in the controller code.

Generally speaking, extensive studies have been made about real-time systems that allow deadline misses, and two main frameworks arose: soft real-time tasks [5] and weakly-hard tasks [3]. The first model is often used for those tasks where completing the execution before the deadline is beneficial, but not strictly required all the time. It is in general associated with a *probabilistic* description of deadline misses, and with the definition of a performance loss function for late completions. Due to its probabilistic nature, this approach has mostly been used to describe multimedia applications, like video encoding and decoding, in which a small delay is not perceived by the end user, as long as it does not happen too frequently. Nonetheless, the soft deadline model has been applied in the past also for the case of control tasks design with some success, e.g., in [6], [8], [13], [14]. The second model – weakly-hard tasks – applies to those cases where it is important to identify worst-case conditions. Weakly-hard tasks are characterized by a maximum number of deadlines that can be missed in a given number of consecutive periods. This can be defined, e.g., by a constraint of the form (m, k) to describe a task that cannot miss more than m deadlines in every k consecutive activations. This second framework has gained a lot of traction recently, especially in the real-time community. Its properties have also been studied for control tasks that occasionally miss deadlines [15], [32], [38], [41], and will also be the focus of this work.

Assessing properties of control systems that miss deadlines is not only important for the embedded systems architects, but it is also an important research area in the control domain [7], [16], [30], [36]. When considering a *control task*, to properly analyse the effect a miss has on the controlled plant, it is necessary to specify also *what happens when the miss is experienced*, both in terms of changes to the control signal and in terms of actions taken to deal with the failed task [37]. In the case of embedded controllers, different approaches are used. For example, a computation instance that experiences a deadline miss can be let continue executing until completion (and possibly used later), while in other applications it is stopped and discarded instead.

There is however quite a mismatch between the guarantees that can be obtained for real-time tasks and platforms [9], [19], [20], [39], and the analysis that is available for control tasks [32], [38]. In particular, few works deal with the *stability* analysis of real-time control tasks subject to deadline misses, and they often lack generality. The analysis presented in [32] (which can be considered the closest work to this paper) is limited to constraints that specify a maximum number of *consecutive* deadline misses. The majority of other state-of-the-art approaches deal with constraints of the form (m, k).

In particular, the works in [30] and [31] study the stability of *networked control systems* where packet dropouts (or system faults) are bounded using the (m, k) constraint model. This model is equivalent to the case where deadline misses represent discarded computations, but its results can not be generalized for the other common case of *late completions*. In other works [23], [28], [29], the authors have studied how to enforce safety guarantees of weakly-hard real-time controllers, with focus also on stability properties. However, such works consider only the case where a deadline miss corresponds to a discarded computation and, in [28], [29], with the additional hypothesis of a known periodic pattern of deadline hits and misses.

This paper aims at filling this gap, by providing a stability analysis that can be applied to a general class of weakly-hard models and strategies at the deadline miss event. To the best of our knowledge, the analysis proposed in this paper is also the first being able to guarantee the stability of systems subject to *multiple* constraints belonging to *any* of the weakly-hard types available in the state-of-the-art. More precisely, the paper advances the state of the art in the following directions:

(i) It formally extends the weakly-hard model to explicitly consider the strategy used to handle miss events. This was intuitively introduced in some prior work, but this paper offers a formal extension, that considers time intervals instead of periodic jobs and sheds some light on the required notation.
(ii) It builds a representation of the control task execution, subject to a set of weakly-hard constraints as a finite state machine. We use the resulting finite state machine to construct a transition matrix that determines the valid state transitions.
(iii) It uses Kronecker lifted switching systems and the joint spectral radius as tools to properly express stability conditions for systems that satisfy a set of weakly-hard constraints.
(iv) It uses the concept of constraint dominance (in which a constraint dominates another if satisfying the first implies always satisfying the second too) to prove analytic bounds on the stability of a weakly-hard constrained system with respect to *less dominant* weakly-hard constraints.
(v) It analyses the stability of the resulting closed-loop systems using `SparseJSR`, an algorithm that exploits the structure (i.e., the sparsity patterns) that naturally arises in the Kronecker lifted representation of the closed-loop systems.

Notably, the analysis presented in this paper calls for modularity and separation of concern, defining the interface between the control engineer and the embedded system designer. It decouples the constraint specification and the control verification. With the proposed method, the embedded system designer can extract a set of constraints that belong to the hardware and software design phase (e.g., the control task is not going to miss more than two consecutive deadlines and not more than three for every subsequent seven activations) and the control engineer can verify that the proposed constraints satisfy all control requirements. This decoupling is one of the main advantages of the proposed solution.

II. Background and Notation

In this section we introduce the necessary background for the paper and introduce the notation used in the following. We also provide a thorough analysis of related research contributions.

A. Control Systems

In this paper, we consider an arbitrary *discrete-time* sampled linear time invariant (LTI) system, expressed as follows:

$$P: \quad \begin{cases} x_{t+1} &= A_p\, x_t + B_p\, u_t \\ y_t &= C_p\, x_t + D_p\, u_t. \end{cases} \tag{1}$$

Here t is a positive integer value, i.e., $t \in \mathbb{N}^{>}$. The system is sampled with sampling period p_τ. The vector $x_t \in \mathbb{R}^n$ contains the plant's internal states, $u_t \in \mathbb{R}^r$ contains the signals controlling the plant, and $y_t \in \mathbb{R}^q$ is the vector of measurement values acquired from the plant at time $t \cdot p_\tau$. Finally A_p, B_p, C_p and D_p are real-valued matrices of appropriate size defining the dynamics of the plant. In line with standard assumptions, we assume the system in (1) to be controllable and the state to be fully observable.

The plant is controlled by a stabilising, LTI, one-step delay,[1] discrete-time controller producing u_t and defined as:

$$C: \quad \begin{cases} z_{t+1} &= A_c\, z_t + B_c\, e_t \\ u_{t+1} &= C_c\, z_t + D_c\, e_t. \end{cases} \tag{2}$$

Here, $z_t \in \mathbb{R}^s$ is the controller's internal state and $e_t \in \mathbb{R}^q$, $e_t = r_t - y_t$ is the tracking error of the controller. For simplicity, and without loss of generality, we will be treating the regulator problem; ergo, no reference tracking ($r_t = 0$).

B. Real-time tasks that may miss deadlines

The controller in (2) is implemented as a real-time task τ, and designed to be executed periodically with period p_τ in a real-time embedded platform. Under nominal conditions the task releases an instance (called *job*) in each period, that should be completed before the release of the next instance. We denote the sequence of activation instants for τ with $(a_i)_{i \in \mathbb{N}^{\geq}}$, such that, in nominal conditions, $a_{i+1} = a_i + p_\tau$, the sequence of completion instants $(f_i)_{i \in \mathbb{N}^{\geq}}$, and the sequence of job deadlines with $(d_i)_{i \in \mathbb{N}^{\geq}}$, such that $d_i = a_i + p_\tau$ (also called *implicit* deadline). This requirement can be either satisfied or not, leading respectively to deadline hits and misses.

Definition 1 (Deadline hit)**.** The i-th job of a periodic task τ with period p_τ (and implicit deadline) hits its deadline when $f_i \leq d_i$.

Definition 2 (Deadline miss)**.** The i-th job of a periodic task τ with period p_τ (and implicit deadline) misses its deadline when $f_i > d_i$.

[1] One-step delay controllers are controllers in which a control signal computed in the k-th interval is actuated at the beginning of the $k+1$-th period. In the real-time systems jargon, this is enforced by adopting the so-called the Logical Execution Time paradigm [12], [25]. From a real-time perspective, it improves the timing predictability, and from a control perspective, it reduces input-output jitter and allows to neglect time-varying computational delays.

We refer to both deadline hits and misses using the term *outcome* of a job. In order to provide some guarantees on how the computation is distributed in the application and therefore how misses and hits are interleaved, *weakly-hard task models* have been introduced in the past [3]. Weakly-hard models impose constraints on the number of deadline hits and misses that can be experienced in a window of $k \geq 1$ consecutive activations. In particular, the following four models have been proposed in literature [3].

Definition 3 (Weakly-hard task models)**.** A task τ may satisfy any combination of these four basic weakly hard constraints:

(i) $\tau \vdash \overline{\binom{m}{k}}$: at most m deadlines are missed, in any window of k consecutive jobs;
(ii) $\tau \vdash \binom{h}{k}$: at least h deadlines are hit, in any window of k consecutive jobs;
(iii) $\tau \vdash \overline{\langle {}^{m}_{k} \rangle}$: at most m *consecutive* deadlines are missed, in any window of k consecutive jobs; and
(iv) $\tau \vdash \langle {}^{h}_{k} \rangle$: at least h *consecutive* deadlines are hit, in any window of k consecutive jobs,

with $m, h, k \in \mathbb{N}^{\geq}$, $m \leq k$, $h \leq k$ and $k \geq 1$.

The model expressed in (ii) was first introduced in [18], but its complementary version (i) has gained most research traction recently, and is often referred to as the (m, k) model. In particular, schedulability analyses have been developed for the (m, k) model [20], [39]. Furthermore, a weakly-hard sensitivity analysis is implemented in a toolchain that allows to derive the strongest satisfied (m, k) constraint from C code [33]. The model in (iii) constrains the maximum number of consecutive deadline misses. As proven in [3], the window size becomes irrelevant in this model,[2] hence it is commonly referred to as $\overline{\langle m \rangle}$ (a notation also adopted in this paper).

These models convey different information about the deadline miss properties of the task, and can also be used jointly, stating that a task τ satisfies a *set* of constraints (possibly of different types). In particular, the possibility of defining sets of constraints, for example by obtaining the worst-case number of deadline misses over different time windows, opens the door to a more detailed (and possibly less pessimistic) timing description of the tasks. This latter aspect has been however generally overlooked by the research literature, which often considers timing models defined by a single weakly-hard constraint. Due to the relative importance and prevalence of (m, k) and $\overline{\langle m \rangle}$ models in the recent literature, we will use such models in our examples. Nonetheless, we remark that the analysis presented in the remainder of the paper is invariant to the choice of weakly-hard model. To simplify the notation, in the following we will refer to a generic weakly-hard constraint using the symbol λ, while a set of L weakly hard constraints will be referred as $\Lambda = \{\lambda_1, \lambda_2, \ldots, \lambda_L\}$.

Stating that $\tau \vdash \lambda$ means that all the possible sequences of outcomes (hits and misses) that τ can experience satisfy the corresponding constraint λ. To this end, we define a string $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_N\}$ of length N. Each element α_i in the string is a symbol that belongs to the alphabet of job outcomes Σ. The most common choices are to use `M` and `H` (or 0 and 1) to indicate respectively a miss and a hit.

[2]A useful result presented in [32] is that $\overline{\langle m \rangle} = (m, m+1)$.

Definition 4 (Alphabet Σ of job outcomes for real-time tasks)**.** The alphabet Σ includes all the possible outcomes of a real-time job. Commonly, $\Sigma = \{\texttt{M}, \texttt{H}\}$ where `M` indicates a deadline miss and `H` indicates a deadline hit.

We use $\alpha \vdash \lambda$ to denote that the string α satisfies the constraint λ. The set of all possible sequences that satisfy a given constraint λ naturally results from the definition of λ, and is formally defined as follows [3].

Definition 5 (Satisfaction set $\mathcal{S}_N(\lambda)$ of a weakly-hard constraint λ)**.** We denote with $\mathcal{S}_N(\lambda)$ the set of sequences of length $N \geq 1$ that satisfy a constraint λ. Formally, $\mathcal{S}_N(\lambda) = \{\alpha \in \Sigma^N \mid \alpha \vdash \lambda\}$.

Taking the limit to infinity, the set $\mathcal{S}_\infty(\lambda)$ contains all the sequences of infinite length that satisfy the constraint λ. We will henceforth denote $\mathcal{S}(\lambda) \equiv \mathcal{S}_\infty(\lambda)$.

By leveraging the definition of satisfaction sets, it is possible to determine how two constraints λ_i and λ_j relate to one another [3], [51]. In particular, we can define a *domination* notion between constraints [3].

Definition 6 (Constraint domination)**.** Given two constraints λ_i and λ_j, we say that λ_i is harder to satisfy than (or dominates) λ_j (denoted by $\lambda_i \prec \lambda_j$) if all sequences that satisfy λ_i also satisfy λ_j, i.e., when $\mathcal{S}(\lambda_i) \subset \mathcal{S}(\lambda_j)$. Similarly, we say that $\lambda_i \preceq \lambda_j$ if $\mathcal{S}(\lambda_i) \subseteq \mathcal{S}(\lambda_j)$.

The notion of constraint domination introduces a partial ordering between constraints [3]. Despite constraint ordering being an active research area (with new relations found recently, e.g., in [51]), the abstraction of properties of the represented physical process by leveraging constraint dominance relationships is still unexplored. In Section V-C, we relate the real-time concept of constraint dominance to the control theoretical notion of stability.

C. Control tasks that may miss deadlines

This paper considers the special case of control tasks subject to weakly-hard constraints, meaning that the task τ implements a control algorithm (e.g., a Proportional Integral and Derivative controller, a Linear Quadratic Regulator, etc).

Since the fundamental properties of stability and performance of a controller depend on the actual pattern of control commands, it is necessary to precisely define what happens when a control deadline is missed. This led to several suggestions in literature for *deadline miss handling strategies* and *actuator modes*. The handling strategies manages the fate of the job that missed the deadline (and possibly the next ones), while the actuator mode deals with the loss of control signal and decides whether it should be held constant, or zeroed [44].[3] A few handling strategies have been proposed in the past [7], [32],

[3]The most common strategy applied by control engineers is *hold*. However, *zero* is sometimes the only viable option, e.g., when holding the previous control signal requires additional computation (like the translation to a different coordinate system) and there is no time to perform such computation.

[37]–[39], the most important being *kill*, *skip-next*, and *queue* (also denoted as *continue*). In particular, the results of [7], [32], [37] suggest that the Queue strategy (i.e., letting each job executing until completion, queuing newly released jobs) may create chain effects of missed deadlines, negatively affecting stability and performance. Thus, Kill and Skip-Next will be the focus of this paper. The following definitions of deadline miss handling strategies hold for a task τ, with period p_τ.

Definition 7 (Kill strategy)**.** The Kill strategy imposes the immediate termination of the job that misses the deadline. A killed job will never complete its execution, meaning that, for an arbitrary i-th job of τ either $f_i \leq d_i$ or $f_i = \infty$. If the i-th job is killed, the $(i+1)$-th job is immediately released, i.e., $a_{i+1} = a_i + p_\tau$.

Definition 8 (Skip-Next strategy)**.** The Skip-Next strategy imposes that a job that misses its deadline is allowed to continue during the following period. When an arbitrary i-th job of τ, released at a_i misses its deadline d_i, a new deadline $d_i^{\succ} = d_i + p_\tau$ is set for the job. The i-th job then becomes a job with activation time a_i and deadline $d_i^{\succ}$. At the same time, the $(i+1)$-th job activation is set to $a_{i+1} = d_i^{\succ}$. The process can be repeated, extending the deadline until the job completion.

D. Existing stability analysis techniques

A control task that may experience deadline misses requires a proper analysis to check its asymptotic stability in all possible timing conditions. In this paragraph we introduce the existing analytic techniques from the control literature, which can be adapted to deal with this problem.

The closest work to this paper is the one in [32], which presents a stability analysis for the $\tau \vdash \overline{\langle m \rangle}$ weakly hard constraint. However, its application for a more general weakly-hard constraint may result in overly pessimistic bounds. More specifically, in [32], the authors identify a set of subsequences of hit and missed deadlines, which can be arbitrarily combined to obtain all possible sequences in $\mathcal{S}\left(\overline{\langle m \rangle}\right)$. After assigning to each subsequence its corresponding dynamic matrix, the stability analysis of the resulting arbitrary switching system can then be obtained by leveraging the *Joint Spectral Radius* (JSR) [24], which is briefly presented here. Given $\ell \in \mathbb{N}^{>}$ and a set of matrices $\mathcal{A} = \{A_1, \ldots, A_\ell\} \subseteq \mathbb{R}^{n \times n}$, under the hypothesis of arbitrary switching, the JSR of $\mathcal{A}$ is defined by:

$$\rho(\mathcal{A}) := \lim_{k \to \infty} \max_{\alpha \in \{1,\ldots,\ell\}^k} \|A_{\alpha_k} \cdots A_{\alpha_2} A_{\alpha_1}\|^{\frac{1}{k}}. \tag{3}$$

The number $\rho(\mathcal{A})$ characterizes the maximal asymptotic growth rate of products of matrices from $\mathcal{A}$ (thus $\rho(\mathcal{A}) < 1$ means that the system is asymptotically stable), and is independent of the norm $\|\cdot\|$ used in (3). When $\mathcal{A}$ is a singleton, the joint spectral radius is equal to the spectral radius, which justifies that the JSR generalizes the spectral radius for the case of multiple matrices.

The JSR was introduced in [43] and is applied in several important contexts, including switched dynamical systems stability and combinatorics. We refer to [24] for more details about theory and applications related to JSR. Existing practical tools such as the JSR Matlab toolbox [46] allow one to compute both upper and lower bounds on $\rho(\mathcal{A})$, leveraging multiple algorithms.

The JSR approach can be generalized for the case of switching sequences constrained by a given graph $\mathcal{G}$, with the so called *constrained joint spectral radius* (CJSR) [10]. Consider again a set of matrices $\mathcal{A}$ and define (with a slight abuse of notation) $\mathcal{S}_k(\mathcal{G})$ as the set of all possible switching sequences of length k that satisfy the constraints of graph $\mathcal{G}$. Then, the CJSR of $\mathcal{A}$ is defined by

$$\rho(\mathcal{A}, \mathcal{G}) := \lim_{k \to \infty} \max_{\alpha \in \mathcal{S}_k(\mathcal{G})} \|A_{\alpha_k} \cdots A_{\alpha_2} A_{\alpha_1}\|^{\frac{1}{k}}. \tag{4}$$

In general, computing or approximating the CJSR is not an easy task, and noticeably fewer works exist for CJSR with respect to JSR. In [40], the authors propose a multinorm-based method to approximate with arbitrary accuracy the CJSR. An ad-hoc branch-and-bound algorithm is proposed in [11]. Another approach [26], [49], [52] is based on the creation of an arbitrary switching system such that its JSR is equal to the CJSR of the original system, based on a Kronecker lifting method. This will be also our approach, as detailed in later sections of the paper.

In [35], the authors propose a hierarchy of relaxations to provide a sequence of upper bounds for the JSR, based on positive polynomials which be can decomposed as *sum of squares* (SOS) of polynomials. This approximation method is particularly interesting in terms of performance and has been chosen as primary approach in our tests. Therefore, to make the paper self-contained, we introduce the basic notation and recall some preliminary material about SOS polynomials. Given $n \in \mathbb{N}^{>}$, let $x = (x_1, \ldots, x_n)$ be a tuple of variables. For $d \in \mathbb{N}^{>}$, let us note $\mathbb{R}[x]$ the ring of real n-variate polynomials and let $\mathbb{R}[x]_{2d}$ be the restriction of $\mathbb{R}[x]$ to the set of homogeneous polynomials of degree $2d$. A polynomial $f \in \mathbb{R}[x]_{2d}$ can be written as $f(x) = \sum_{\nu \in \mathscr{A}} f_\nu x^\nu$ with $f_\nu \in \mathbb{R}, x^\nu = x_1^{\nu_1} \cdots x_n^{\nu_n}$ and $\mathscr{A} \subseteq \mathbb{N}^n$. If there exist homogeneous polynomials $f_1, \ldots, f_t \in \mathbb{R}[x]_d$ such that $f = \sum_{i=1}^t f_i^2$, then we say that f is an SOS polynomial. Let SOS_{2d} be the subset of SOS polynomials in $\mathbb{R}[x]_{2d}$ The core ingredient of the hierarchy of [35] is based on the following result (Theorem 2.2 of [35]).

Theorem 1 (JSR upper bound with homogeneous polynomials)**.** *Given a set of matrices $\mathcal{A} = \{A_1, \ldots, A_\ell\} \subseteq \mathbb{R}^{n \times n}$, let p be a strictly positive homogeneous polynomial of degree $2d$ that satisfies*

$$\gamma^{2d} p(x) - p(A_i x) \geq 0, \quad \forall x \in \mathbb{R}^n, \quad i = 1, \ldots, \ell.$$

Then, $\rho(\mathcal{A}) \leq \gamma$.

By replacing $\gamma^{2d} p(x) - p(A_i x) \geq 0$ with the more tractable condition that $\gamma^{2d} p(x) - p(A_i x)$ is SOS, we obtain a hierarchy of relaxations indexed by $d \in \mathbb{N}^{>}$, which in turn yields a sequence of upper bounds for $\rho(\mathcal{A})$:

$$\rho_{\mathrm{SOS},2d}(\mathcal{A}) := \inf_{p \in \mathbb{R}[x]_{2d}, \gamma} \gamma \tag{5}$$
$$\text{s.t.} \begin{cases} p(x) \in \mathrm{SOS}_{2d}, \\ \gamma^{2d} p(x) - p(A_i x) \in \mathrm{SOS}_{2d}, \ 1 \leq i \leq \ell. \end{cases}$$

The optimal value of this optimization problem can be computed by combining a bisection procedure on γ together with *semidefinite programming* (SDP) [50]. SDP solvers minimize a linear objective function with linear matrix inequality constraints. The value of an SDP problem can be approximated up to a given precision in polynomial time with respect to its input size, while relying on interior-point algorithms, implemented in modern solvers such as Mosek [2]. Finding the coefficients of a polynomial being SOS boils down to solving an SDP [27], [34], [42]. The upper bound $\rho_{\mathrm{SOS},2d}(\mathcal{A})$ satisfies the following inequalities (Theorem 4.3 of [35]).

Theorem 2 (SOS relaxation)**.** *Let $\mathcal{A} = \{A_1, \ldots, A_\ell\} \subseteq \mathbb{R}^{n\times n}$. For any integer $d \geq 1$, $\ell^{-\frac{1}{2d}} \rho_{SOS,2d}(\mathcal{A}) \leq \rho(\mathcal{A}) \leq \rho_{SOS,2d}(\mathcal{A})$.*

Theorem 2 implies the convergence of $\{\rho_{\mathrm{SOS},2d}(\mathcal{A})\}_{d\geq 1}$ to $\rho(\mathcal{A})$ when d goes to infinity. To reduce the computational burden associated with this procedure, a *sparse* variant, called `SparseJSR`, has been proposed in [47] to exploit the sparsity of the input matrices from $\mathcal{A}$, based on the *term sparsity* SOS (TSSOS) framework [48]. This allows the authors to provide SOS decompositions of polynomials with arbitrary sparse support. By contrast, the procedure defined in (5) will be denoted hereafter as *dense*.

Next, we briefly recall some preliminary material from [47] to explain how the `SparseJSR` algorithm works. The *support* of f is defined by $\mathrm{supp}(f) := \{\nu \in \mathscr{A} \mid f_\nu \neq 0\}$. Let $\mathbb{R}[\mathscr{A}] \subset \mathbb{R}[x]_{2d}$ be the set of polynomials whose supports are contained in $\mathscr{A}$, and $\mathrm{SOS}[\mathscr{A}] \subset \mathrm{SOS}_{2d}$ be the set of SOS polynomials whose supports are contained in $\mathscr{A}$. The authors in [47] restrict the support of the unknown polynomial p from Equation (5) to obtain a sparse SOS program, choosing $p_0(x) = x_1^{2d} + x_2^{2d} + \ldots + x_n^{2d}$ to be the power sum of coordinates with a fixed degree $2d$, $\mathscr{A}_0 = \mathrm{supp}(p_0)$, and

$$\mathscr{A} := \mathscr{A}_0 \cup \bigcup_{i=1}^{\ell} \mathrm{supp}\left(p_0\left(A_i x\right)\right). \tag{6}$$

Let $\mathscr{A}_i = \mathscr{A} \cup \mathrm{supp}(p(A_i x))$ for $i = 1, \ldots, \ell$. To take into account the sparsity of the matrices from $\mathcal{A}$, SOS_{2d} is replaced by either by $\mathrm{SOS}_{\mathscr{A}}$ or $\mathrm{SOS}_{\mathscr{A}_i}$ in (5), obtaining:

$$\rho_{\mathrm{TSSOS},2d}(\mathcal{A}) := \inf_{p\in\mathbb{R}[\mathscr{A}],\gamma} \gamma \tag{7}$$
$$\text{s.t.} \begin{cases} p(x) \in \mathrm{SOS}_{\mathscr{A}}, \\ \gamma^{2d} p(x) - p(A_i x) \in \mathrm{SOS}_{\mathscr{A}_i},\ 1 \leq i \leq \ell. \end{cases}$$

As for $\rho_{\mathrm{SOS},2d}(\mathcal{A})$, the optimal value $\rho_{\mathrm{TSSOS},2d}(\mathcal{A})$ of (7) can be obtained from a bisection procedure. From Theorem 4.1 of [47], this gives $\rho(\mathcal{A}) \leq \rho_{\mathrm{SOS},2d}(\mathcal{A}) \leq \rho_{\mathrm{TSSOS},2d}(\mathcal{A})$, yielding a hierarchy of upper bounds for the JSR $\rho(\mathcal{A})$. By contrast with the dense case, the sparse upper bound can be obtained significantly faster if the matrices from $\mathcal{A}$ are sparse. This is relevant for the specific application we are considering, as the matrices that we have to analyse (introduced in Section V) are sparse and it is possible to exploit their structure to reduce computation times. We remark that it is also possible to substitute SOS with other algorithms such as branch and bound or ellipsoid [46] to compute bounds on the value of the JSR.

III. Extended Weakly-Hard Task Model

An important point raised in previous works [32], [37], [38], is that the original weakly-hard model is not sufficiently descriptive to model the dynamics of a control task executing on an embedded platform. The missing piece of the puzzle is represented by the actual strategy used to handle the control task's deadline misses. In fact, different strategies may have dramatically different effects on the physical equations and properties of the control system. For instance, under Kill strategy, a missed deadline represents a job that will never be completed. However, in the case of Skip-Next strategy, a missed deadline represents a job that will complete *later*. As a consequence, the chosen strategy directly affects the actual pattern of the control signal when jobs miss their deadlines.

A model that only counts the number of missed deadlines is leaving out pieces of information about how those deadline misses affect the controlled system. Indeed, what we are really interested in is *if* and *when* a job is completed. This Section presents our proposal to giving practical shape of such intuition, by extending the concepts related to weakly-hard constraints (Section II-B) to also include the deadline handling strategy.

To provide a comprehensive analysis framework, we need to examine what occurs in each time interval $(\pi_i)_{i\in\mathbb{N}^{\geq}}$, with $\pi_i = [a_0 + i \cdot p_\tau, a_0 + (i+1) \cdot p_\tau)$. In fact, depending on the strategy that is used to handle deadline misses, the activation rate of jobs may be decoupled by the nominal periodic pattern. In this context, the alphabet Σ introduced in Definition 4 and the weakly-hard model need to be extended to account for the specific deadline handling procedure – denoted hereafter with the symbol $\mathcal{H}$. This is done by directly including the deadline miss handling strategy $\mathcal{H}$ in the model, and defining a richer characterization of what happens at each time interval π_i of the control task τ, as follows.

Definition 9 (Extended Weakly Hard Task $\tau \vdash \lambda_{\mathcal{H}}$)**.** An extended weakly-hard task τ satisfies any combination of these four constraints:

(i) $\tau \vdash \overline{\binom{m}{k}}_{\mathcal{H}}$: at most m intervals lack a job completion, in any window of k consecutive jobs;
(ii) $\tau \vdash \binom{h}{k}_{\mathcal{H}}$: at least h intervals contain a job completion, in any window of k consecutive jobs;
(iii) $\tau \vdash \overline{\langle {}^{m}_{k} \rangle}_{\mathcal{H}}$: at most m *consecutive* intervals lack a job completion, in any window of k consecutive jobs; and
(iv) $\tau \vdash \langle {}^{h}_{k} \rangle_{\mathcal{H}}$: at least h *consecutive* intervals contain a job completion, in any window of k consecutive jobs,

with $m, h, k \in \mathbb{N}^{\geq}$, $m \leq k$, $h \leq k$ and $k \geq 1$, while using strategy $\mathcal{H}$ to handle potential deadline misses.

The definition above differs from Definition 3 in two points: firstly, it focuses on the presence of a new control command at the end of each time interval π_i, instead of checking the outcome (deadline hit or miss) of a job; secondly, it explicitly introduces the handling strategy by adding the symbol $\mathcal{H}$. The dependency of the new extended weakly-hard model on the strategy $\mathcal{H}$ is thus required by the necessity of introducing a

more expressive alphabet $\Sigma(\mathcal{H})$ to characterize the behaviour of task τ in each possible time interval. In the case under analysis in this paper, for both the strategies Kill and Skip-Next, in each interval π_i at most one job is activated and at most one job is completed. This restricts the possible behaviours to three cases, defined by the following symbols:

(i) a time interval in which the same job is both released and completed is denoted by `H` (*hit*);
(ii) a time interval in which either one or zero jobs are released, but no job is completed is denoted by `M` (*miss*);
(iii) a time interval in which no job is released, but a job (released in a previous interval) is completed, is denoted by `R` (*recovery*).

Therefore, we can build an alphabet for each possible strategy, namely $\Sigma(\mathcal{H})$. In the case of Kill and Skip-Next we obtain:

(i) $\Sigma(\text{Kill}) = \{\texttt{M}, \texttt{H}\}$, and
(ii) $\Sigma(\text{Skip-Next}) = \{\texttt{M}, \texttt{H}, \texttt{R}\}$.

Trivially, $\Sigma(\text{Kill})$ uses the basic alphabet from Definition 4. In fact, exactly one job is activated at the beginning of each time interval when Kill is used – and that job has a binary termination outcome. The additional recovery character `R` is used in the Skip-Next alphabet to identify the late completion of a job that was not activated at the beginning of the current time interval. As a consequence, recalling Definition 9, `R` is treated equivalently to `H` when checking the extended weakly hard constraint. Similarly, an alphabet can be created for any arbitrary handling strategy, by checking all unique combinations of job activations and completions in each interval. Notably, the definition of such an alphabet does not require any additional information about the actuator modes introduced in Section II-C. However, the actuator mode will be fundamental later in Section V-A in order to properly express the resulting dynamic matrices.

We can now extend the algebra presented in Section II-B to the new alphabet. We assign a character of the alphabet $\Sigma(\mathcal{H})$ to each interval π_i. A string $\alpha = \{\alpha_1, \alpha_2, \ldots, \alpha_N\} \in \Sigma(\mathcal{H})^N$ is used to represent a sequence of outcomes (hits, misses and recoveries) for task τ, spanning any N consecutive time intervals, with α_t representing the outcome associated to the interval π_t. Without loss of generality, we always consider ideal startup conditions, i.e., $\alpha_t = \texttt{H}$, $\forall t \leq 0$.

In principle, the ordered sequence α could be any combination of characters in $\Sigma(\mathcal{H})$. However, we are only interested in analysing the set of *feasible sequences* of the task τ. To this end, we introduce an order constraint for the `R` character.

Rule 1. *For any arbitrary string $\alpha \in \Sigma(\textit{Skip-Next})^N$, `R` may only directly follow a miss `M`, or be the initial element of the string.*

The rule is trivial when noticing that a recovery `R` at period π_i corresponds to the completion of a job that was activated in an interval π_j with $j < i$, that is not completed yet. Hereafter, we will consider each arbitrary sequence $\alpha \in \Sigma(\text{Skip-Next})^N$ to always satisfy Rule 1.

The extended weakly hard model presented so far also inherits all the properties of the original weakly hard model. In particular, the satisfaction set of $\lambda_{\mathcal{H}}$ can be defined for $N \geq 1$ as $\mathcal{S}_N(\lambda_{\mathcal{H}}) = \{\alpha \in \Sigma(\mathcal{H})^N \mid \alpha \vdash \lambda_{\mathcal{H}}\}$, and the constraint domination still holds as $\lambda_{\mathcal{H},i} \preceq \lambda_{\mathcal{H},j}$ if $\mathcal{S}(\lambda_{\mathcal{H},i}) \subseteq \mathcal{S}(\lambda_{\mathcal{H},j})$. As a consequences, when the context is clear, to avoid heavy notation, we will hereafter inherit the same symbol λ to identify a constraint in the extended form, i.e., $\lambda \equiv \lambda_{\mathcal{H}}$, with implicit dependency on $\mathcal{H}$.

IV. State Machine Realisation

In this section we propose a way to model a set of extended weakly-hard constraints as a finite state machine, by utilising the extended alphabet $\Sigma(\mathcal{H})$ and the concept of *dominant set*. The presented approach is invariant to the actual control system dynamics. For this reason, the resulting model can be used as an interface that separates the software design phase from the stability analysis (Section V), allowing the complete architecture analysis to be decoupled.

A. Constraint graph

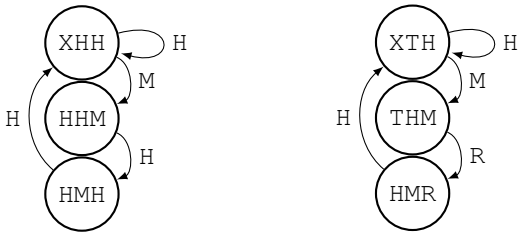
An extended weakly-hard constraint, as presented in Definition 9, can be systematically represented using a minimal *Finite State Machine* (FSM) and the corresponding directed graph. We denote the corresponding graph of a task $\tau \vdash \lambda$ (recall $\lambda \equiv \lambda_{\mathcal{H}}$) with the symbol $\mathcal{G}_\lambda = (V_\lambda, E_\lambda)$. Here, V_λ represents the set of *nodes* in the graph and E_λ represents the set of *transitions*.

Each node in V_λ corresponds to a *word* $w_i \in \mathcal{S}_k(\lambda)$. A *slice* $w_i(a..b)$ of a word is a new word obtained from w_i by extracting only the characters starting at position a and ending at position b (both included). We assume the initial position in the string to be position 1. Furthermore, we use $w_i(a..\text{end})$ to indicate the slice from position a till the end of the word. A transition $c_{i,j} \in E_\lambda$ takes us from node w_i to node w_j. The transition is labelled by a character $c \in \Sigma(\mathcal{H})$. Node w_j is said to be a direct successor of w_i if concatenating character c to the end of $w_i(2..end)$ gives w_j.

Intuitively, a graph $\mathcal{G}_\lambda$ has *at most* a number of nodes equal to the cardinality of the set of feasible words $\mathcal{S}_k(\lambda)$, or formally $|V_\lambda| \leq |\mathcal{S}_k(\lambda)|$. Building the graph $\mathcal{G}_\lambda$ with a number of states $|V_\lambda| = |\mathcal{S}_k(\lambda)|$ is always possible, but it would unnecessarily worsen the computational complexity of the problem.

A *minimal state machine realisation* $\mathcal{G}^*_\lambda = (V^*_\lambda, E^*_\lambda)_{\mathcal{H}}$ is easily obtained from $\mathcal{G}_\lambda$ using standard techniques [21]. For each node w_i, we observe its successors $\{w_k\}$. Given two nodes w_i and w_j, if they have the same successors with the same transition events c they are considered equivalent and thereby combined. The differing characters in w_i and w_j are replaced by an *any character* token, which we denote with `X`. This process is repeated until the state machine can no longer be reduced. When considering the Skip-Next strategy, an additional auxiliary token `T` is introduced to represent all events where a job is completed (ergo; $\texttt{T} = \{\texttt{H}, \texttt{R}\}$). This token is necessary to describe $\mathcal{G}_\lambda$, since job completions from `H` and `R` events are equivalent for any arbitrary $\lambda_{\text{Skip-Next}}$ constraint.

Example 1. *Given an extended constraint $\lambda = (1,3)_{\text{Kill}}$, the minimal realisation $\mathcal{G}^*_\lambda$ is shown in the left-hand side of Figure 1. The node `XHH` is obtained by merging `HHH` and*



(Example 1) $\lambda = (1,3)_{\text{Kill}}$ (Example 2) $\lambda = (1,3)_{\text{Skip-Next}}$

Fig. 1. Minimal graph $\mathcal{G}^*_\lambda$ for Examples 1 and 2.

MHH. It can be interpreted as the first character not affecting the possible transitions.

Example 2. *Given an extended constraint $\lambda = (1,3)_{\text{Skip-Next}}$, the minimal realisation $\mathcal{G}^*_\lambda$ is shown in the right-hand side of Figure 1.*

The similarities between Kill and Skip-Next under the extended notation become apparent as soon as we compare the minimal state machines in Figure 1. The resulting graphs have identical number of nodes but slightly different transition events. However, this difference becomes significant when analysing the corresponding closed-loop systems, as will be clear in Section V.

B. Dealing with multiple constraints

The approach presented above for constructing a minimal FSM can be extended to the case where the task τ is subject to a set of multiple extended weakly hard constraints, in any combination of the forms presented in Definition 9. Such a set is formally denoted as $\Lambda_{\mathcal{H}}$, but again to simplify the notation, for the remainder of the paper $\Lambda \equiv \Lambda_{\mathcal{H}}$, unless explicitly stated otherwise. In order to optimise the problem of building the minimal FSM of the corresponding system, it is beneficial that the set Λ contains only constraints that cannot be further reduced using the hardness relations of Definition 6. For this reason, we first introduce the new concept of *dominant constraint set* (or simply dominant set).

Definition 10 (Dominant set)**.** Given a set of constraints Λ under strategy $\mathcal{H}$, the set $\Lambda^\star \subseteq \Lambda$ is called the *dominant set* of Λ if:

(i) $\lambda_i, \lambda_j \in \Lambda^\star \implies \lambda_i \not\preceq \lambda_j,\ \forall i \neq j,$
(ii) $\lambda_i \in \Lambda \setminus \Lambda^\star \implies \lambda_j \preceq \lambda_i,\ \exists \lambda_j \in \Lambda^\star.$

Building upon Definition 5, the satisfaction set of $\Lambda^\star$ can be derived as follows, for $N \geq 1$:

$$\mathcal{S}_N(\Lambda^*) \equiv \bigcap_{\lambda_i \in \Lambda^\star} \mathcal{S}_N(\lambda_i). \tag{8}$$

We can then obtain a minimal graph $\mathcal{G}^*_{\Lambda^\star} = (V^*_{\Lambda^\star}, E^*_{\Lambda^\star})$ following a procedure similar to the one presented for a single constraint λ in Section IV. As a first step, the length of the words in each node of the corresponding graph must be defined. Since $\Lambda^\star$ may contain constraints with different window values, the choice is not straightforward. We propose a safe assumption about the words length $k^\star$, defined as $k^\star = \max_k \lambda,\ \forall \lambda \in \Lambda^\star$ (recall that each weakly-hard constraint is window dependent). An algorithm can then be built that, recursively, adds nodes $w_i \in \mathcal{S}_{k^\star}(\Lambda^\star)$ to the graph, until a minimal realisation is generated. The obtained graph is finally passed through a post-processing step (strategy-dependent) that assigns appropriate transitions $c_{i,j}$ in order to obtain a correct minimal FSM realisation $\mathcal{G}^*_{\Lambda^\star}$.

Example 3. *Given two extended constraints $\lambda_1 = \overline{\langle 2 \rangle}_{\text{Kill}}$ and $\lambda_2 = (3,\,5)_{\text{Kill}}$, the minimal realisation graphs $\mathcal{G}^*_{\lambda_1}$ and $\mathcal{G}^*_{\lambda_2}$ are shown as the leftmost and middle graphs of Figure 2. Generating the graph $\mathcal{G}^*_{\Lambda^\star}$ from the dominant set $\Lambda^\star = \{\lambda_1, \lambda_2\}$ results in the rightmost graph of Figure 2, satisfying both λ_1 and λ_2. A similar result can be obtained for the* Skip-Next *strategy, by using its corresponding alphabet.*

Building a minimal FSM by choosing the dominant set $\Lambda^\star$ is useful to improve the computational performance of the analysis algorithms. Nonetheless, the analysis presented in the following sections can be applied to any graph built from a given set Λ. To limit complexity in the notation, and for the sake of generality, all future steps will be defined using a generic graph $\mathcal{G}_\Lambda$.

C. Dynamic model of a graph

An arbitrary walk of N steps in the state machine $\mathcal{G}_\Lambda$ corresponds to a distinct sequence $\alpha \in \mathcal{S}_N(\Lambda)$. Extracting all the transitions $c_{i,j} \in E_\Lambda$ corresponding to a particular event c yields what is generally known as a *directed adjacency matrix*, denoted here as *transition matrix* due to its connection to the transition event c.

Definition 11 (Transition matrix)**.** Given a graph $\mathcal{G}_\Lambda$, the directed adjacency matrix (or *transition matrix*), $F_c \in \mathbb{R}^{|V_\Lambda| \times |V_\Lambda|}$ with $c \in \Sigma(\mathcal{H})$, is computed as $F_c = \{f_{i,j}(c)\}$ with

$$f_{i,j}(c) = \begin{cases} 1, & \text{if } c_{j,i} \in E_\Lambda \\ 0, & \text{otherwise} \end{cases}$$

The walk (i.e., the completion sequence) can then be expressed in terms of a left-multiplication between the corresponding transition matrices. Since there can only exist *at most one* successor from each node with the transition event c, the transition matrix F_c will thus have a column sum of either 1 or 0. Finally, we introduce a vector $q_t \in \mathbb{R}^{|V_\Lambda|}$ called *G-state* (for graph state), representing the state of the given graph $\mathcal{G}_\Lambda$, which is associated to the interval π_t. This vector is formally defined as follows.

Definition 12 (G-state q_t)**.** Given a graph $\mathcal{G}_\Lambda = (V_\Lambda, E_\Lambda)$ and a sequence $\alpha \in \Sigma(\mathcal{H})^N$, for $k = |w|,\ w \in V_\Lambda$, we define $q_t \in \mathbb{R}^{|V_\Lambda|}$, where the i-th element, $q_t(i)$, is defined as:

$$q_t(i) = \begin{cases} 1, & \text{if } \alpha(t-k..t-1) \equiv w_i \in V_\Lambda \\ 0, & \text{otherwise.} \end{cases}$$

In other words, the G-state q_t is the vector representation of the index corresponding to the node we are *leaving* at step t. Particularly, for all feasible sequences $\alpha \in \mathcal{S}_N(\Lambda)$, q_t contains *exactly* one 1 at index i, where $w_i \equiv \alpha(t-k..t-1) \in V_\Lambda$ (all

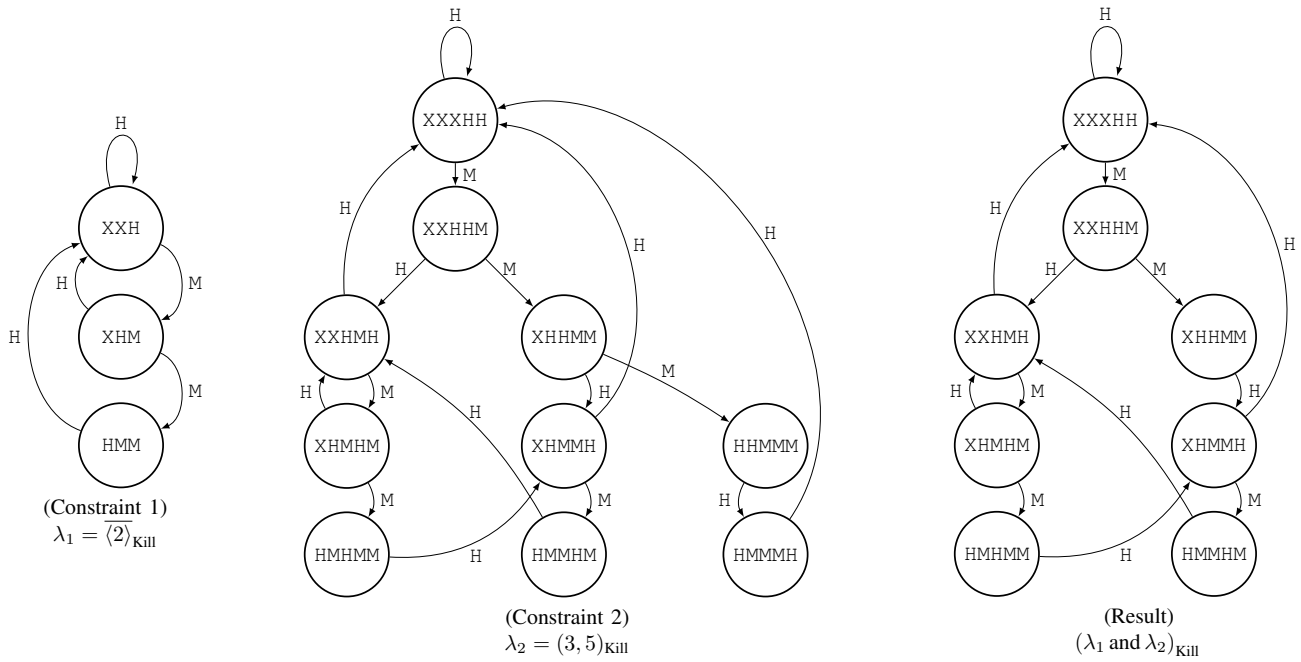


Fig. 2. Minimal graphs $\mathcal{G}^*_{\lambda_1}$, $\mathcal{G}^*_{\lambda_2}$, and $\mathcal{G}^*_{\Lambda^\star}$ for Example 3.

other values are 0). However, for an arbitrary sequence $\alpha \in \Sigma(\mathcal{H})^N$, the G-state contains *at most* one 1. In this definition, $q_t = 0$ means that the transition $\alpha(t-1)$ was infeasible for the graph. The G-state dynamics, given an arbitrary sequence α, can then easily be defined as $q_{t+1} = F_c \cdot q_t$ with $c = \alpha_t$. Trivially the following property from [53] holds:

Lemma 1 (Transition matrix of an infeasible sequence)**.** *If* $\alpha \notin \mathcal{S}_N(\Lambda)$*, then* $F_\alpha = F_{\alpha_N} \cdots F_{\alpha_2} F_{\alpha_1} = 0$

Thus, if $q_t = 0$ for any t, then $q_{t'} = 0$ for $t' \geq t$.

V. Stability Analysis

In this Section we illustrate the dynamics of the closed-loop system under weakly-hard constraints, together with an efficient way to implement a stability analysis and some interesting properties of the resulting system.

A. Closed-loop system

Using the alphabet $\Sigma(\mathcal{H})$, defined for a given deadline miss strategy $\mathcal{H}$, and the chosen actuator mode (Zero or Hold), we now compute the closed-loop system behaviour of plant (1) with controller (2). In particular, we identify one matrix for each distinct dynamics corresponding to a character $c \in \Sigma(\mathcal{H})$, building the set $\mathcal{A}_\mathcal{H}$. Again, we treat the cases of Kill and Skip-Next, but the same approach can be extended to other strategies.

Kill: The alphabet for the Kill strategy includes two characters, $\Sigma(\text{Kill}) = \{\mathtt{H}, \mathtt{M}\}$. We define $\tilde{x}^{\mathtt{K}}_t = \begin{bmatrix} x_t^T & z_t^T & u_t^T \end{bmatrix}^T$ as the state vector for the closed-loop system. We compute the one-step closed-loop system dynamics $A^{\mathtt{K}}_{\mathtt{H}}$, corresponding to the character $\mathtt{H}$ (i.e., when the job released in the given period π_t successfully completes within its deadline) as follows.

$$\tilde{x}^{\mathtt{K}}_{t+1} = A^{\mathtt{K}}_{\mathtt{H}} \tilde{x}^{\mathtt{K}}_t, \quad A^{\mathtt{K}}_{\mathtt{H}} = \begin{bmatrix} A_p & 0 & B_p \\ -B_c C_p & A_c & -B_c D_p \\ -D_c C_p & C_c & -D_c D_p \end{bmatrix}$$

On the other hand, for the case of $\mathtt{M}$ (i.e., when the job released in π_t misses its deadline) the controller terminates its execution prematurely by killing the job, thus not updating its states ($z_{t+1} = z_t$). In this case, the controller output is forced to a default value, determined by the actuator mode and is thus either zeroed ($u_{t+1} = 0$) or held ($u_{t+1} = u_t$). The resulting closed-loop system in state-space form is denoted with $A^{\mathtt{K}}_{\mathtt{M}}$ and defined as follows.

$$\tilde{x}^{\mathtt{K}}_{t+1} = A^{\mathtt{K}}_{\mathtt{M}} \tilde{x}^{\mathtt{K}}_t, \quad A^{\mathtt{K}}_{\mathtt{M}} = \begin{bmatrix} A_p & 0 & B_p \\ 0 & I & 0 \\ 0 & 0 & \Delta \end{bmatrix}$$

Here, I is the identity matrix of appropriate size, and Δ assume values depending on the actuation mode, i.e., $\Delta = I$ if the control signal is held and $\Delta = 0$ if it is zeroed. The set of dynamic matrices that a controlled system under Kill strategy may experience is $\mathcal{A}_{\text{Kill}} = \left\{A^{\mathtt{K}}_{\mathtt{H}}, A^{\mathtt{K}}_{\mathtt{M}}\right\}$.

Skip-Next: When considering constraints under the Skip-Next strategy, characterizing the resulting closed-loop dynamics requires a different state vector model. In this case, when a control task released in an arbitrary interval π_t misses a deadline, it is allowed to continue its execution until completion. Once completed, the control state will then be updated with a control command computed using *old* measurement values. Thus, we introduce two additional states $\hat{x}_t$ and $\hat{u}_t$ that store those old values while the controller awaits an update. The resulting state vector then becomes $\tilde{x}^{\mathtt{S}}_t = \begin{bmatrix} x_t^T & z_t^T & u_t^T & \hat{x}_t^T & \hat{u}_t^T \end{bmatrix}^T$.

When π_t is associated to H (i.e., a job is both activated and completed in π_t), the two augmented states mirror the behaviour of the states of which they are storing data. The resulting closed-loop system is described using $A^{\mathrm{S}}_{\mathrm{H}}$ as follows:

$$\tilde{x}^{\mathrm{S}}_{t+1} = A^{\mathrm{S}}_{\mathrm{H}}\,\tilde{x}^{\mathrm{S}}_{t},\;\; A^{\mathrm{S}}_{\mathrm{H}} = \begin{bmatrix} A_p & 0 & B_p & 0 & 0 \\ -B_cC_p & A_c & -B_cD_p & 0 & 0 \\ -D_cC_p & C_c & -D_cD_p & 0 & 0 \\ A_p & 0 & B_p & 0 & 0 \\ -D_cC_p & C_c & -D_cD_p & 0 & 0 \end{bmatrix}.$$

For the case of M in π_t (i.e., when a job is not completed) the two augmented states hold their previous values. The resulting closed-loop system is described by $A^{\mathrm{S}}_{\mathrm{M}}$:

$$\tilde{x}^{\mathrm{S}}_{t+1} = A^{\mathrm{S}}_{\mathrm{M}}\,\tilde{x}^{\mathrm{S}}_{t},\;\; A^{\mathrm{S}}_{\mathrm{M}} = \begin{bmatrix} A_p & 0 & B_p & 0 & 0 \\ 0 & I & 0 & 0 & 0 \\ 0 & 0 & \Delta & 0 & 0 \\ 0 & 0 & 0 & I & 0 \\ 0 & 0 & 0 & 0 & I \end{bmatrix}.$$

Finally, for the case of recovery R in π_t (i.e., when no control job is activated, but an old one is completed) the new control command is calculated using the old measurement and control values stored in $\hat{x}_t$ and $\hat{u}_t$. The resulting closed-loop system is described by $A^{\mathrm{S}}_{\mathrm{R}}$ as follows:

$$\tilde{x}^{\mathrm{S}}_{t+1} = A^{\mathrm{S}}_{\mathrm{R}}\,\tilde{x}^{\mathrm{S}}_{t},\;\; A^{\mathrm{S}}_{\mathrm{R}} = \begin{bmatrix} A_p & 0 & B_p & 0 & 0 \\ 0 & A_c & 0 & -B_cC_p & -B_cD_p \\ 0 & C_c & 0 & -D_cC_p & -D_cD_p \\ A_p & 0 & B_p & 0 & 0 \\ 0 & C_c & 0 & -D_cC_p & -D_cD_p \end{bmatrix}.$$

The set of dynamic matrices under Skip-Next strategy can then be finally defined as $\mathcal{A}_{\text{Skip-Next}} = \left\{A^{\mathrm{S}}_{\mathrm{H}}, A^{\mathrm{S}}_{\mathrm{M}}, A^{\mathrm{S}}_{\mathrm{R}}\right\}$.

B. Kronecker lifted switching system

We have so far derived the possible switching patterns under a set of constraints Λ in terms of a graph $\mathcal{G}_\Lambda$ (Section III), and the closed-loop switching dynamics $\mathcal{A}_\mathcal{H}$ of the controlled system (Section V-A). To analyse the system stability under any switching pattern constrained by Λ, we are then required to combine the set of system dynamics $\mathcal{A}_\mathcal{H}$ with the FSM describing the allowed switches $\mathcal{G}_\Lambda$. A straightforward approach would be configuring a CJSR problem in the form of Equation (4), obtaining $\rho\left(\mathcal{A}_\mathcal{H}, \mathcal{G}_\Lambda\right)$.

Conversely, building upon the recent work of [52], we seek to obtain an equivalent model of the system based on Kronecker lifting. The model obtained with this approach, characterized by a set of matrices that we denote with $\mathcal{A}_\Lambda$, behaves as an *arbitrary switching system*, such that the JSR of the set $\mathcal{A}_\Lambda$ is equal to the CJSR of the constrained system $\rho\left(\mathcal{A}_\mathcal{H}, \mathcal{G}_\Lambda\right)$, i.e., $\rho\left(\mathcal{A}_\Lambda\right) = \rho\left(\mathcal{A}_\mathcal{H}, \mathcal{G}_\Lambda\right)$. In this way, we can use powerful algorithms applicable to arbitrary switching system – such as the ones implemented in [46] and the `SparseJSR` algorithm [47] – to find tight stability bounds of our target controlled system. For the sake of brevity, we will give only an intuitive explanation here about the Kronecker lifting approach presented in [52]. The interested reader is referred to [52] for a more detailed analysis.

The Kronecker product is formally defined as follows [22].

Definition 13 (Kronecker product)**.** The Kronecker product between two real-valued matrices A and B of any size is defined as:

$$A \otimes B := \begin{bmatrix} a_{11}\cdot B & a_{12}\cdot B & \dots \\ a_{21}\cdot B & a_{22}\cdot B & \dots \\ \vdots & \vdots & \ddots \end{bmatrix}.$$

Differently from the common matrix product, the Kronecker product does not require the equivalence between the number of rows of A and columns of B.

The Kronecker product is used here to build a set of lifted matrices (P_c) that includes the information of both the system dynamics (closed-loop matrix $A^{\mathcal{H}}_c$) and the possible transitions (transition matrix F_c) of a given outcome $c \in \Sigma\left(\mathcal{H}\right)$. To avoid heavy notation, we will consider $A^{\mathcal{H}}_c \equiv A_c$ henceforth, with implicit dependency on $\mathcal{H}$.

We introduce the *lifted discrete-time state* $\xi_t \in \mathbb{R}^{n|V_\Lambda|}$ defined as

$$\xi_t = q_t \otimes x_t.$$

By construction, ξ_t is a vector composed of $|V_\Lambda|$ blocks of size n, where at most one block is equal to x_t and all the other blocks are equal to the 0 vector. All the possible lifted closed-loop system dynamic matrix associated to ξ_t are defined as

$$P_c = F_c \otimes A_c, \quad c \in \Sigma\left(\mathcal{H}\right), \tag{9}$$

where $P_c \in \mathbb{R}^{n|V_\Lambda| \times n|V_\Lambda|}$. The lifted dynamics of the closed loop systems then become

$$\xi_{t+1} = P_c \xi_t, \quad c = \alpha_t.$$

To better understand how this works, we here apply the theory to Example 1.

Example 4. *Consider the graph of Example* 1 *in Figure 1 (left). The transition matrices F_H and F_M, computed using Definition 11, are:*

$$F_{\mathrm{H}} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \quad F_{\mathrm{M}} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}. \tag{10}$$

The lifted matrices corresponding to H and M are then computed using Equation (9) *as follows:*

$$P_{\mathrm{H}} = \begin{bmatrix} A_{\mathrm{H}} & 0 & A_{\mathrm{H}} \\ 0 & 0 & 0 \\ 0 & A_{\mathrm{H}} & 0 \end{bmatrix}, \quad P_{\mathrm{M}} = \begin{bmatrix} 0 & 0 & 0 \\ A_{\mathrm{M}} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}. \tag{11}$$

Let consider an initial G-state $q_1 = [1,\,0,\,0]^T$ and an initial plant state x_1. Given a sequence $\alpha =$ HM, we obtain:

$$\begin{aligned} \xi_3 &= P_{\mathrm{M}}\,P_{\mathrm{H}}\,\xi_1 \\ &= \begin{bmatrix} 0 & 0 & 0 \\ A_{\mathrm{M}} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} A_{\mathrm{H}} & 0 & A_{\mathrm{H}} \\ 0 & 0 & 0 \\ 0 & A_{\mathrm{H}} & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ A_{\mathrm{M}}\,A_{\mathrm{H}}\,x_1 \\ 0 \end{bmatrix} \end{aligned}$$

Instead, if the sequence $\alpha =$ MM was given, which is infeasible for this example ($\alpha \notin \mathcal{S}\left((1,3)_{\text{Kill}}\right)$), the dynamics will converge to $\xi = 0$.

$$\xi_3 = P_{\mathrm{M}}\,P_{\mathrm{M}}\,\xi_1 = \begin{bmatrix} 0 & 0 & 0 \\ A_{\mathrm{M}} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ A_{\mathrm{M}} & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Formally, we obtain a system composed of a set of switching dynamic matrices, $\mathcal{A}_\Lambda$, which is defined as follows.

Definition 14 (Lifted switching set $\mathcal{A}_\Lambda$)**.** Given a set of dynamic matrices $\mathcal{A}_\mathcal{H}$ and a graph $\mathcal{G}_\Lambda$, the switching set $\mathcal{A}_\Lambda$ is defined as the set of all lifted matrices corresponding to $\mathcal{A}_\mathcal{H}$. Formally,

$$\mathcal{A}_\Lambda = \{F_c \otimes A_c \mid c \in \Sigma(\mathcal{H}),\, A_c \in \mathcal{A}_\mathcal{H}\}.$$

Proving then that $\rho(\mathcal{A}_\Lambda) = \rho(\mathcal{A}_\mathcal{H}, \mathcal{G}_\Lambda)$ requires first finding a proper norm for the lifted system. For an arbitrary sequence $\alpha \in \Sigma(\mathcal{H})$, the mixed-product property [22] of $\otimes$ guarantees the following relationship:

$$\begin{aligned} P_\alpha &= P_{\alpha_N} \cdots P_{\alpha_2} P_{\alpha_1} \\ &= (F_{\alpha_N} \cdots F_{\alpha_2} F_{\alpha_1}) \otimes (A_{\alpha_N} \cdots A_{\alpha_2} A_{\alpha_1}) \end{aligned}$$

Then, we introduce a submultiplicative norm $|||\cdot|||$ such that, for an arbitrary matrix M, it holds that

$$|||M||| = \max_j \sum_i \|m_{i,j}\|.$$

Finally, recalling from Section IV-C that there may be at most one entry 1 in each column of F_α, and from Lemma 1 that $F_\alpha = 0$ if $\alpha \notin \mathcal{S}(\Lambda)$, we obtain that:

$$|||P_\alpha||| = \begin{cases} \|A_\alpha\| & \text{when } \alpha \in \mathcal{S}(\Lambda) \\ 0 & \text{when } \alpha \notin \mathcal{S}(\Lambda) \end{cases} \tag{12}$$

As a consequence, it follows that $\rho(\mathcal{A}_\Lambda) = \rho(\mathcal{A}_\mathcal{H}, \mathcal{G}_\Lambda)$, which proves the viability of our approach.

C. Extended weakly hard and JSR properties

In Section II-D we recalled existing stability results on the $\overline{\langle m \rangle}$ model obtained with the JSR [32]. We now provide a general relation between *all* extended weakly-hard constraints in terms of the joint spectral radii, by leveraging the model presented earlier in the paper.

Theorem 3 (JSR dominance)**.** *Given two arbitrary weakly-hard constraints λ_1 and λ_2, if $\lambda_2 \preceq \lambda_1$ then*

$$\rho(\mathcal{A}_{\lambda_2}) \leq \rho(\mathcal{A}_{\lambda_1}).$$

Proof. From Equation (3), for a generic λ,

$$\rho(\mathcal{A}_\lambda) = \lim_{\ell\to\infty} \rho_\ell(\mathcal{A}_\lambda), \quad \rho_\ell(\mathcal{A}_\lambda) = \max_{\alpha\in\mathcal{S}_\ell(\lambda)} \|A_\alpha\|^{1/\ell}. \tag{13}$$

Definition 6 gave us that $\lambda_2 \preceq \lambda_1$ if and only if $\mathcal{S}(\lambda_2) \subseteq \mathcal{S}(\lambda_1)$. Thus, if an arbitrary sequence is in the satisfaction set of λ_2, i.e., $\beta \in \mathcal{S}_\ell(\lambda_2)$, this means β also belongs to the satisfaction set of λ_1, i.e., $\beta \in \mathcal{S}_\ell(\lambda_1)$. The set of all possible A_β is thus included in the set of all possible A_α, $\alpha \in \mathcal{S}_\ell(\lambda_1)$. As a consequence it holds that

$$\max_{\beta\in\mathcal{S}_\ell(\lambda_2)} \|A_\beta\|^{1/\ell} \leq \max_{\alpha\in\mathcal{S}_\ell(\lambda_1)} \|A_\alpha\|^{1/\ell}, \quad \forall \ell \in \mathbb{N}^{>}.$$

The theorem follows immediately when $\ell \to \infty$. □

The intuition that a sequence of control activations containing a large number of deadline misses is less robust than one including fewer errors, is well-established. However, Theorem 3 is the first result that provides a clear, analytic, correlation between the control theoretical analysis and real-time implementation. Primarily, it implies that the constraint dominance from Definition 6 also carries on to the JSR, giving us a notion of *JSR dominance*. One of the most important consequences arising from the JSR dominance is the relation between the implementation and analysis of a weakly-hard constrained system. If stability under a specific constraint is shown, Theorem 3 guarantees stability for all constraints which are harder to satisfy. This result also applies to the stability analysis presented in [32], giving the user options for which methodology to use. Additionally, we wish to emphasise that the results of Theorem 3 are strategy independent (as long as λ_1 and λ_2 use the same strategy $\mathcal{H}$), further reducing the coupling between the control analysis and implementation approach.

Although Theorem 3 holds for any weakly-hard constraint λ_1 and λ_2, we here present a valuable special case of the theorem. Due to the $\overline{\langle m \rangle}_\mathcal{H}$ and $(m,k)_\mathcal{H}$ constraints being the two most used models, we derive some practical relations between them in terms of the corresponding joint spectral radii.

Corollary 1 ($(m,k)_\mathcal{H}$ dominance)**.** *Given a constraint $\lambda_1 = (m,k_1)_\mathcal{H}$ and a constraint $\lambda_2 = (m,k_2)_\mathcal{H}$, if $k_1 \leq k_2$ then*

$$\rho(\mathcal{A}_{\lambda_2}) \leq \rho(\mathcal{A}_{\lambda_1}).$$

Proof. According to [51] it holds that $(m,k_2)_\mathcal{H} \preceq (m,k_1)_\mathcal{H}$ if $k_1 \leq k_2$. The corollary follows directly from [51] and Theorem 3. □

Corollary 2 ($\overline{\langle m \rangle}_\mathcal{H}$ dominance)**.** *Given a constraint $\lambda_1 = \overline{\langle m \rangle}_\mathcal{H}$ and a constraint $\lambda_2 = (m,k)_\mathcal{H}$, $k > m$, then*

$$\rho(\mathcal{A}_{\lambda_2}) \leq \rho(\mathcal{A}_{\lambda_1}).$$

Proof. According to [32] it holds that $\overline{\langle m \rangle}_\mathcal{H} \equiv (m, m+1)_\mathcal{H}$. The corollary follows directly from [32] and Corollary 1. □

The conclusions drawn from Theorem 3 are theoretical and its practical applicability depends on the algorithms used to find lower and upper bounds for the JSR value ρ^{LB} and ρ^{UB}. Using these bounds we can determine the stability of the switching system. However, it is not necessarily true that the upper bounds found using different algorithms follow the ordering presented above. Still, we can bound the switching stability as

$$\rho^{LB}(\mathcal{A}_{\lambda_2}) \leq \rho(\mathcal{A}_{\lambda_2}) \leq \rho(\mathcal{A}_{\lambda_1}) \leq \rho^{UB}(\mathcal{A}_{\lambda_1}).$$

Regardless of the algorithm used to find the bounds, we can generally conclude that if $\rho^{UB}(\mathcal{A}_\lambda) < 1$ we know that the constrained system is switching stable. Similarly, if $\rho^{LB}(\mathcal{A}_\lambda) > 1$ the system is unstable. Thus, if $\lambda_2 \preceq \lambda_1$, where $\rho^{UB}(\mathcal{A}_{\lambda_1}) < 1$, we know that the system under λ_2 constraints is switching stable. A similar relation holds for the lower bound.

We now extend the results of Theorem 3 by relating the joint spectral radius of a single constraint to sets of weakly-hard constraints.

Theorem 4. *Given an arbitrary weakly-hard constraint λ, it holds that*

$$\rho\left(\mathcal{A}_{\Lambda}\right) \leq \rho\left(\mathcal{A}_{\lambda}\right), \ \forall \Lambda \ni \lambda.$$

Proof. A relationship between the satisfaction set of a constraint set and the individual constraints's satisfaction sets was presented in Equation (8). Consequently, for an arbitrary constraint λ and constraint set Λ, where $\Lambda = \{\Lambda', \lambda\}$, it holds that

$$\mathcal{S}_{\ell}\left(\Lambda\right) = \mathcal{S}_{\ell}\left(\Lambda'\right) \cap \mathcal{S}_{\ell}\left(\lambda\right) \subseteq \mathcal{S}_{\ell}\left(\lambda\right). \tag{14}$$

Thus, if an arbitrary sequence is in the satisfaction set of Λ, i.e., $\beta \in \mathcal{S}_{\ell}\left(\Lambda\right)$, this means β also belongs to the satisfaction set of λ, i.e., $\beta \in \mathcal{S}_{\ell}\left(\lambda\right)$. The set of all possible A_{β} is thus included in the set of all possible $A_{\alpha},\ \alpha \in \mathcal{S}_{\ell}\left(\lambda\right)$. As a consequence it holds that

$$\max_{\beta \in \mathcal{S}_{\ell}(\Lambda)} \|A_{\beta}\|^{1/\ell} \leq \max_{\alpha \in \mathcal{S}_{\ell}(\lambda)} \|A_{\alpha}\|^{1/\ell}, \quad \forall \ell \in \mathbb{N}^{>}.$$

The theorem follows immediately when $\ell \to \infty$. □

Following the same intuition as Theorem 3, the more we restrict the execution pattern of the task constrained by λ, the lower the JSR will be. In the case of constraint sets, the notion of JSR dominance is not as evident as it was for a single constraint. However, the JSR dominance extension to sets of weakly-hard constraints further improves the relation between constraint specification and control verification.

One of the most important implications of Theorem 4 is the practical significance it has on the switching stability of the system. Specifically, enforcing tighter weakly-hard constraints to a stable system will *never* destabilise it. Thus, if a system has been shown to be stable under a distinct constraint λ, the system implementation may be changed arbitrarily as long as the control task's execution still satisfies λ. The result is formally given by the following corollary.

Corollary 3. *Given an arbitrary weakly-hard constraint λ, if $\rho\left(\mathcal{A}_{\lambda}\right) < 1$ then*

$$\rho\left(\mathcal{A}_{\Lambda}\right) < 1, \ \forall \Lambda \ni \lambda.$$

Proof. The corollary follows immediately from Theorem 4 when $\rho\left(\mathcal{A}_{\lambda}\right) < 1$. □

VI. Evaluation

We here apply the lifted dynamics model presented in Section V to two representative case studies. The corresponding controllers are designed to stabilise the closed loop in ideal conditions, i.e., without deadline misses. We perform numerical experiments to analyse the stability of the two control systems, when they are subject to different constraints, particularly in the $\lambda = (m, k)_{\mathcal{H}}$ form (due to its prevalence). We consider both the Kill and Skip-Next strategy, and we also vary the actuator mode, being either Zero or Hold.

For each combination of plant and constraint λ, a minimal FSM is built and the resulting closed loop systems are expressed in the lifted form of Section V, each yielding a specific set of matrices $\mathcal{A}_{\lambda}$. The stability of the system is computed by approximating the JSR of $\mathcal{A}_{\lambda}$, namely $\rho\left(\mathcal{A}_{\lambda}\right)$, using three different algorithms. First, a lower and upper bound of $\rho\left(\mathcal{A}_{\lambda}\right)$ are computed using the `JSR toolbox` [46] in Matlab. We compare these bounds with an upper bound of the JSR obtained via SOS relaxations, as described in Section II-D, both with the *dense* and *sparse* algorithm from the `SparseJSR` toolbox [47]. Concretely, we compute upper bounds on the JSR given by the optimal value $\rho_{\text{SOS},2d}\left(\mathcal{A}_{\lambda}\right)$ of the dense SOS relaxation (5) as well the optimal value $\rho_{\text{TSSOS},2d}\left(\mathcal{A}_{\lambda}\right)$ of the sparse relaxation (7) implemented in the `SparseJSR` algorithm. For efficiency, we run experiments at the first relaxation order $d = 1$.

The `JSR toolbox` provides an accurate lower bound and a coarse upper bound in a few seconds. In contrast, the dense SOS-based method usually finds a good upper bound but takes more time. The sparse/dense upper bounds are obtained with the `SparseJSR` Julia package, based on the `TSSOS` package used in [48] and the SDP solver MOSEK [2]. Since `JSR toolbox` and `SparseJSR` are implemented in different programming languages (Matlab and Julia) and rely on different SDP solvers (SDPT3/SeDuMi and MOSEK), it is not meaningful to compare their respective timings. However, the time it takes to run the dense and sparse SOS methods in Julia is compared. All numerical examples have been computed on an Intel Core i5-8265U@1.60GHz CPU with 8GB RAM memory.

A. Process industrial plant

We here analyse a stable discrete-time plant P_1, representative of the process industry [17], controlled using a PI-controller C_1 (sampled using the sampling period $p_{\tau} = 0.5$ s):

$$P_1 : \begin{cases} x_{t+1} &= \begin{bmatrix} 0.606 & 0.304 & 0.076 \\ 0 & 0.606 & 0.304 \\ 0 & 0 & 0.606 \end{bmatrix} x_t + \begin{bmatrix} 0.014 \\ 0.091 \\ 0.394 \end{bmatrix} u_t \\ y_t &= \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} x_t \end{cases}$$

$$C_1 : \begin{cases} z_{t+1} &= z_t + 0.359 y_t \\ u_{t+1} &= 0.454 z_t + 0.633 y_t \end{cases}$$

Table I display the results obtained with the distinct strategies (Kill and Skip-Next) combined with the actuator modes (either Zero or Hold), for the initially stable plant. Lower and upper bounds are denoted with "LB" and "UB". The symbol "×" stands for the speedup factor of the time required to obtain the sparse bound w.r.t. the dense one. The symbol "−" means that the SDP solver runs out of memory, and the test is interrupted. Bold values represent stable systems under corresponding constraint for the given strategy and actuator mode. The starred values represent stable systems infered from Corollary 1.

All the upper bounds computed by `JSR toolbox` are greater than 1, while all lower bounds are *below* 1, thus we cannot draw any conclusion about the stability of the considered system using the `JSR toolbox`. However, for all constraints $(m, k)_{\mathcal{H}}$ where $m = 1$ and $2 < k \leq 6$ the dense/sparse SOS upper bounds allow us to infer that the system is stable for all combinations of strategy and actuator mode, and also for $k = 2$ under the Skip-Next strategy. As a consequence of Corollary 1 the stability will hold also for all $(m, k)_{\mathcal{H}}$ with $m = 1$ and $k > 6$. The computing time and the speedup ratio are growing when k increases, yielding a particularly high benefit of exploiting sparsity for the Skip-Next strategy

TABLE I

RESULTS OBTAINED FOR THE STABLE SYSTEM P_1, WHEN CONTROLLED USING C_1.

	Kill and Zero					**Kill and Hold**					**Skip-Next and Zero**					**Skip-Next and Hold**				
	JSR [46]		Dense	Sparse		JSR [46]		Dense	Sparse		JSR [46]		Dense	Sparse		JSR [46]		Dense	Sparse	
(m,k)	LB	UB	UB	UB	×	LB	UB	UB	UB	×	LB	UB	UB	UB	×	LB	UB	UB	UB	×
$(1,2)$	0.960	1.094	1.070	1.070	0.86	0.926	1.094	1.029	1.029	0.83	0.922	1.086	**0.924**	**0.924**	5.40	0.958	1.083	**0.958**	**0.958**	4.43
$(1,3)$	0.920	1.062	**0.995**	**0.995**	0.83	0.894	1.053	**0.971**	**0.971**	0.77	0.898	1.077	**0.974**	**0.974**	10.5	0.917	1.077	**0.988**	**0.988**	10.4
$(1,4)$	0.890	1.038	**0.945**	**0.996**	1.06	0.894	1.021	**0.957**	1.025*	1.25	0.898	1.057	**0.963**	**0.963**	18.2	0.890	1.063	**0.940**	**0.940**	15.9
$(1,5)$	0.890	1.011	**0.922**	**0.983**	1.96	0.894	1.011	**0.948**	1.008*	2.25	0.898	1.026	**0.954**	**0.954**	17.6	0.890	1.039	**0.929**	**0.929**	20.8
$(1,6)$	0.890	1.012	**0.920**	**0.975**	4.36	0.894	1.016	**0.942**	**0.995**	3.68	0.898	1.016	**0.946**	**0.947**	20.9	0.890	1.023	**0.927**	**0.927**	25.8
$(2,3)$	0.983	1.148	1.124	1.124	0.67	0.956	1.152	1.085	1.085	0.80	0.953	1.145	1.034	1.039	4.45	0.982	1.148	1.070	1.076	5.91
$(2,4)$	0.960	1.155	1.079	1.079	0.74	0.927	1.160	1.039	1.039	0.86	0.922	1.165	1.033	1.040	23.9	0.958	1.167	1.079	1.086	24.2
$(2,5)$	0.939	1.156	1.039	1.142	2.09	0.905	1.156	1.002	1.105	1.58	0.898	1.186	**0.999**	1.005	77.8	0.937	1.182	1.038	1.043	58.1
$(2,6)$	0.920	1.150	1.007	1.096	12.3	0.903	1.145	**0.974**	1.080	19.2	0.907	1.184	–	1.007	–	0.917	1.182	–	**0.991**	–
$(3,4)$	0.990	1.186	1.133	1.133	0.76	0.967	1.192	1.098	1.098	1.69	0.967	1.177	1.072	1.082	6.59	0.990	1.191	1.106	1.117	5.02
$(3,5)$	0.975	1.210	1.109	1.109	0.77	0.946	1.215	1.071	1.071	1.74	0.942	1.234	1.071	1.080	34.3	0.975	1.233	1.116	1.125	35.2
$(3,6)$	0.960	1.247	1.082	1.227	2.61	0.928	1.252	1.043	1.182	3.25	0.921	1.246	–	1.118	–	0.959	1.242	–	1.072	–
$(4,5)$	0.994	1.198	1.130	1.130	1.06	0.976	1.206	1.099	1.099	0.82	0.974	1.189	1.122	1.134	5.43	0.993	1.121	1.088	1.100	5.16
$(4,6)$	0.983	1.260	1.120	1.120	0.68	0.957	1.267	1.084	1.084	0.64	0.953	1.267	–	1.143	–	0.983	1.265	–	1.100	–

TABLE II

RESULTS OBTAINED FOR THE UNSTABLE SYSTEM P_2, WHEN CONTROLLED USING C_2.

	Kill and Zero					**Kill and Hold**					**Skip-Next and Zero**					**Skip-Next and Hold**				
	JSR [46]		Dense	Sparse		JSR [46]		Dense	Sparse		JSR [46]		Dense	Sparse		JSR [46]		Dense	Sparse	
(m,k)	LB	UB	UB	UB	×	LB	UB	UB	UB	×	LB	UB	UB	UB	×	LB	UB	UB	UB	×
$(1,2)$	0.995	1.163	1.148	**0.995**	0.02	0.995	1.133	1.149	**0.998**	0.01	0.995	1.187	**0.995**	**0.995**	1.87	0.995	1.178	**0.995**	**0.995**	1.82
$(1,3)$	0.995	1.128	1.116	1.116*	0.78	0.995	1.109	1.116	1.116*	0.75	0.995	1.166	1.109*	1.109*	3.00	0.995	1.147	1.110*	1.110*	2.96
$(1,4)$	0.995	1.110	1.095	1.169*	1.38	0.995	1.098	1.095	1.169*	1.28	0.995	1.145	1.093*	1.093*	4.05	0.995	1.134	1.093*	1.093*	5.05
$(1,5)$	0.995	1.099	1.080	1.145*	2.17	0.995	1.076	1.081	1.145*	2.66	0.995	1.130	1.079*	1.079*	3.90	0.995	1.120	1.080*	1.080*	4.84
$(1,6)$	0.995	1.088	1.070	1.128*	3.88	0.995	1.079	1.070	1.128*	4.79	0.995	1.126	1.069*	1.069*	4.04	0.995	1.117	1.070*	1.070*	4.61
$(2,3)$	0.995	1.217	1.162	**0.997**	0.01	0.995	1.194	1.166	1.166	0.88	0.995	1.278	1.090	1.095	1.66	0.995	1.289	1.094	1.100	1.63
$(2,4)$	0.995	1.226	1.148	1.148*	0.91	0.995	1.204	1.149	1.149	0.80	0.995	1.293	1.150	1.159	2.96	0.995	1.282	1.152	1.161	4.60
$(2,5)$	0.995	1.224	1.131	1.195*	1.74	0.995	1.205	1.132	1.195	1.87	0.995	1.287	1.134	1.142	7.79	0.995	1.269	1.135	1.143	8.70
$(2,6)$	0.995	1.216	1.118	1.195*	7.49	0.995	1.201	1.118	1.195	12.4	0.995	1.274	1.120	1.135	15.8	0.995	1.264	–	1.136	–
$(3,4)$	0.999	1.262	1.154	1.154	0.91	0.995	1.243	1.159	1.159	0.86	0.998	1.345	1.123	1.133	1.09	0.995	1.354	1.127	1.135	1.31
$(3,5)$	0.995	1.279	1.153	1.153	0.86	0.995	1.262	1.156	1.156	0.76	0.995	1.381	1.163	1.175	2.31	0.995	1.378	1.166	1.177	3.30
$(3,6)$	0.995	1.314	1.144	1.195	2.67	0.995	1.299	1.146	1.218	2.76	0.995	1.357	–	1.163	–	0.995	1.360	–	–	–
$(4,5)$	1.000	1.275	1.147	1.147	0.91	0.995	1.263	1.149	1.149	0.90	1.000	1.365	1.138	1.148	1.25	0.995	1.377	1.140	1.149	1.26
$(4,6)$	0.995	1.340	1.148	1.148	0.71	0.995	1.328	1.153	1.153	0.60	0.995	1.419	1.166	1.178	2.12	0.995	1.414	–	–	–

and Zero actuation. For instance, with $m = 1$ the computing time of the dense SOS upper bound is between 2.32 seconds (for $k = 2$) and 273 seconds (for $k = 6$), while obtaining the sparse SOS upper bound takes from 0.43 to 13.1 seconds. Additionally, increasing the value of m result in harder analyses. For the Skip-Next strategy and Zero actuation, it takes more than 1 hour to compute the dense SOS upper bounds with $m = 2$ or $m = 3$ and $k = 5$, and less than 2 minutes to obtain the corresponding sparse upper bounds. This follows from the higher number of nodes in the corresponding FSM, thus increasing the sizes of the matrices in $\mathcal{A}_\lambda$. As a consequence, we managed to complete the tests with the dense SOS only up to $(m, k)_{\mathcal{H}} = (2, 6)$ with the Kill strategy and Hold actuation and up to $(m, k)_{\mathcal{H}} = (2, 5)$ with the Skip-Next strategy and Zero actuation.

To further distinguish this work from the state-of-the-art [32], we performed additional tests on P_1 and C_1, comparing the lower and upper bounds of the JSR for the $\overline{\langle m \rangle}_{\mathcal{H}}$ constraint model. Both methodologies were tested using the `JSR toolbox` for all combinations of strategy and actuator mode, with $5 \leq m \leq 10$. The results show that our lifted model yields an average improvement of bound accuracy of 10% as well as an average speedup factor equal to 6.

B. Ballistic missile

Our second case study treats the stability analysis of the altitude control on a ballistic missile [4], [45]. The dynamics are given by an unstable discrete-time model P_2, which is stabilised using an LQR-controller C_2 (sampled using the sampling period $p_\tau = 0.01$ s):

$$P_2 : \begin{cases} x_{t+1} &= \begin{bmatrix} 0.999 & 0.012 & -5.5e^{-4} \\ 0.020 & 1 & -5.5e^{-6} \\ 5.0e^{-5} & 0.005 & 1 \end{bmatrix} x_t + \begin{bmatrix} 0.020 \\ 2.0e^{-4} \\ 3.3e^{-7} \end{bmatrix} u_t \\ y_t &= I x_t \end{cases}$$

$$C_2 : \quad u_{t+1} = -\begin{bmatrix} 3.380 & 3.417 & 1.846 \end{bmatrix} x_t - 0.322 u_t$$

Table II display the results obtained by running the `SparseJSR` toolbox on each of the strategies combined with the actuation modes. Again, applying Corollary 1, the stability of the case $(1, 2)_{\mathcal{H}}$ guarantees that the system is stable for $m = 1$ and $k > 2$, under both the Kill and Skip-Next strategies. Almost all reported sparse SOS upper bounds have been obtained with the first relaxation order $d = 1$, using the same notation as for Table I. However, we extend the notation by underlining values computed with the second relaxation order $d = 2$. These values correspond to tighter upper bounds on the joint spectral radii, but come with a much higher computational cost. For instance, we remark that $(m, k)_{\text{Kill}} = (1, 2)_{\text{Kill}}$ is stable using either actuation mode (invisible using $d = 1$), a result acquired at the cost of a factor 100 increase in computation time.

In Example 3, we presented the FSM corresponding to the constraint set $\Lambda = \{\lambda_1, \lambda_2\}$ consisting of $\lambda_1 = \overline{\langle 2 \rangle}_{\text{Kill}}$ and $\lambda_2 = (3, 5)_{\text{Kill}}$. Since $\overline{\langle m \rangle}_{\mathcal{H}} = (m, m + 1)_{\mathcal{H}}$ and $(2, 3)_{\text{Kill}}$ is stable, according to Table II, applying Corollary 3 allows us to deduce that the ballistic missile is stable under the weakly-hard constraint set Λ.

C. Discussion

We would like to conlude the experimental evaluation with a remark. The qualitative and quantitative difference between the two tables is not suprising, as the two controllers are different in nature. We controlled the stable plant in Section VI-A with a PI controller, that has a state corresponding to the integral error, whose update is affected by the deadline misses. On the contrary, the LQR controller developed for the unstable plant in Section VI-B does not have a state and therefore recovers immediately. This explains the similarity between the values of the lower bounds found using the JSR toolbox in Table II.

VII. Conclusion

This paper proposes a switching stability analysis framework for control systems subject to weakly-hard constraints. The existing weakly-hard models are extended by introducing the choice of deadline handling strategy as part of the model. The main contributions of the paper are twofold: (i) an analytic bound on the switching stability for control systems subject to a set of constraints, relating the hardness of the implementation to the stability of the system, and (ii) a decoupled framework where the real-time implementation and control stability analysis can be performed separately. We applied the analysis to multiple examples, with different dynamics and implementations, to show the wide applicability of the approach.

References

[1] Benny Akesson, Mitra Nasri, Geoffrey Nelissen, Sebastian Altmeyer, and Robert Ian Davis. An empirical survey-based study into industry practice in real-time systems. In *2020 IEEE Real-Time Systems Symposium (Proceedings)*. York, 2020.

[2] Erling D. Andersen and Knud D. Andersen. The mosek interior point optimizer for linear programming: an implementation of the homogeneous algorithm. In *High performance optimization*, pages 197–232. Springer, 2000.

[3] Guillem Bernat, Alan Burns, and Albert Liamosi. Weakly hard real-time systems. *IEEE Transactions on Computers*, 50:308 – 321, 2001.

[4] John H. Blakelock. *Automatic Control of Aircraft and Missiles*. Wiley, 2nd edition, 1991.

[5] Giorgio Buttazzo, Giuseppe Lipari, Luca Abeni, and Marco Caccamo. *Soft Real-Time Systems*. Springer, 2005.

[6] Marco Caccamo, Giorgio Buttazzo, and Lui Sha. Handling execution overruns in hard real-time control systems. *IEEE Transactions on Computers*, 51(7):835–849, 2002.

[7] Anton Cervin. Analysis of overrun strategies in periodic control tasks. *IFAC Proceedings Volumes*, 38(1):219–224, 2005.

[8] Anton Cervin and Johan Eker. Control-scheduling codesign of real-time systems: The control server approach. *Journal of Embedded Computing*, 1(2):209–224, 2005.

[9] Hyunjong Choi, Hyoseung Kim, and Qi Zhu. Job-class-level fixed priority scheduling of weakly-hard real-time systems. In *2019 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 241–253. IEEE, 2019.

[10] Xiongping Dai. A gel'fand-type spectral radius formula and stability of linear constrained switching systems. *Linear Algebra and its Applications*, 436(5):1099–1113, 2012.

[11] Fabio Dercole and Fabio Della Rossa. A simple tree-based algorithm for deciding the stability of discrete-time switched linear systems. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 5298–5303. IEEE, 2017.

[12] Rolf Ernst, Stefan Kuntz, Sophie Quinton, and Martin Simons. The Logical Execution Time Paradigm: New Perspectives for Multicore Systems (Dagstuhl Seminar 18092). *Dagstuhl Reports*, 8:122–149, 2018.

[13] Daniele Fontanelli, Luca Greco, and Luigi Palopoli. Soft real-time scheduling for embedded control systems. *Automatica*, 49(8):2330–2338, 2013.

[14] Daniele Fontanelli, Luca Greco, and Luigi Palopoli. Optimal mean square control using the continuous stream model of computation. In *2015 54th IEEE Conference on Decision and Control (CDC)*, pages 1958–1965, 2015.

[15] Goran Frehse, Arne Hamann, Sophie Quinton, and Matthias Woehrle. Formal analysis of timing effects on closed-loop properties of control software. In *2014 IEEE Real-Time Systems Symposium*, pages 53–62, December 2014.

[16] Saurav Kumar Ghosh, Soumyajit Dey, Dip Goswami, Daniel Mueller-Gritschneder, and Samarjit Chakraborty. Design and validation of fault-tolerant embedded controllers. In Jan Madsen and Ayse K. Coskun, editors, *Design, Automation & Test in Europe Conference Exhibition (DATE), 2018*. IEEE, 2018.

[17] Tore Hägglund and Karl-Johan Åström. Revisiting the ziegler-nichols tuning rules for pi control. *Asian Journal of Control*, 4:364 – 380, 2002.

[18] Moncef Hamdaoui and Parameswaran Ramanathan. A dynamic priority assignment technique for streams with (m, k)-firm deadlines. *IEEE Transactions on Computers*, 44(12):1443–1451, December 1995.

[19] Zain A. H. Hammadeh, Rolf Ernst, Sophie Quinton, Rafik Henia, and Laurent Rioux. Bounding deadline misses in weakly-hard real-time systems with task dependencies. In *Design, Automation & Test in Europe Conference Exhibition (DATE), 2017*, pages 584–589, March 2017.

[20] Zain A. H. Hammadeh, Sophie Quinton, and Rolf Ernst. Weakly-hard real-time guarantees for earliest deadline first scheduling of independent tasks. *ACM Trans. Embed. Comput. Syst.*, 18(6), 2019.

[21] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2nd edition, 2001.

[22] Roger A. Horn and Charles R. Johnson. *Matrix analysis*. Cambridge university press, 2012.

[23] Chao Huang, Kacper Wardega, Wenchao Li, and Qi Zhu. Exploring weakly-hard paradigm for networked systems. In *Proceedings of the Workshop on Design Automation for CPS and IoT*, pages 51–59, 2019.

[24] Raphaël Jungers. *The Joint Spectral Radius: Theory and Applications*. Lecture Notes in Control and Information Sciences. Springer Berlin Heidelberg, 2009.

[25] Christoph Kirsch and Ana Sokolova. *The Logical Execution Time Paradigm*, pages 103–120. Springer Berlin Heidelberg, 10 2012.

[26] Victor Kozyakin. The berger–wang formula for the markovian joint spectral radius. *Linear Algebra and its Applications*, 448:315–328, 2014.

[27] Jean B. Lasserre. Global optimization with polynomials and the problem of moments. *SIAM Journal on optimization*, 11(3):796–817, 2001.

[28] Hengyi Liang, Zhilu Wang, Ruochen Jiao, and Qi Zhu. Leveraging weakly-hard constraints for improving system fault tolerance with functional and timing guarantees. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9. IEEE, 2020.

[29] Hengyi Liang, Zhilu Wang, Debayan Roy, Soumyajit Dey, Samarjit Chakraborty, and Qi Zhu. Security-driven codesign with weakly-hard constraints for real-time embedded systems. In *2019 IEEE 37th International Conference on Computer Design (ICCD)*, pages 217–226. IEEE, 2019.

[30] Steffen Linsenmayer and Frank Allgower. Stabilization of networked control systems with weakly hard real-time dropout description. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 4765–4770, 2017.

[31] Steffen Linsenmayer, Michael Hertneck, and Frank Allgower. Linear weakly hard real-time control systems: Time-and event-triggered stabilization. *IEEE Transactions on Automatic Control*, 2020.

[32] Martina Maggio, Arne Hamann, Eckart Mayer-John, and Dirk Ziegenbein. Control-System Stability under Consecutive Deadline Misses Constraints. In Marcus Völp, editor, *32nd Euromicro Conference on Real-Time Systems (ECRTS 2020)*, Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[33] Saranya Natarajan, Mitra Nasri, David Broman, Björn B Brandenburg, and Geoffrey Nelissen. From code to weakly hard constraints: A pragmatic end-to-end toolchain for timed c. In *2019 IEEE Real-Time Systems Symposium (RTSS)*, pages 167–180, 2019.

[34] Pablo A. Parrilo. *Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization*. PhD thesis, California Institute of Technology, 2000.

[35] Pablo A. Parrilo and Ali Jadbabaie. Approximation of the joint spectral radius using sum of squares. *Linear Algebra and its Applications*, 428(10):2385–2402, 2008.

[36] Paolo Pazzaglia, Arne Hamann, Dirk Ziegenbein, and Martina Maggio. Adaptive design of real-time control systems subject to sporadic overruns. In *Design, Automation & Test in Europe Conference Exhibition (DATE), 2021*, 2021.

[37] Paolo Pazzaglia, Claudio Mandrioli, Martina Maggio, and Anton Cervin. DMAC: Deadline-Miss-Aware Control. In Sophie Quinton, editor, *31st Euromicro Conference on Real-Time Systems (ECRTS 2019)*, volume 133 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 1:1–1:24, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[38] Paolo Pazzaglia, Luigi Pannocchi, Alessandro Biondi, and Marco Di Natale. Beyond the Weakly Hard Model: Measuring the Performance Cost of Deadline Misses. In *30th Euromicro Conference on Real-Time Systems (ECRTS 2018)*, volume 106 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 10:1–10:22, 2018.

[39] Paolo Pazzaglia, Youcheng Sun, and Marco Di Natale. Generalized weakly hard schedulability analysis for real-time periodic tasks. *ACM Transactions on Embedded Computing Systems (TECS)*, 20(1):1–26, 2020.

[40] Matthew Philippe, Ray Essick, Geir E Dullerud, and Raphaël M Jungers. Stability of discrete-time switching systems with constrained switching sequences. *Automatica*, 72:242–250, 2016.

[41] Parameswaran Ramanathan. Overload management in real-time control applications using (m, k)-firm guarantee. *IEEE Transactions on parallel and distributed systems*, 10(6):549–559, 1999.

[42] Bruce Reznick et al. Extremal psd forms with few terms. *Duke mathematical journal*, 45(2):363–374, 1978.

[43] Gian-Carlo Rota and W Strang. A note on the joint spectral radius. 1960.

[44] L. Schenato. To zero or to hold control inputs with lossy links? *IEEE Transactions on Automatic Control*, 54(5):1093–1099, 2009.

[45] R. Padma Sree and M. Chidambaram. *Control of Unstable Systems*. Alpha Science, illustrated edition, 2006.

[46] Guillaume Vankeerberghen, Julien Hendrickx, and Raphaël M Jungers. Jsr: A toolbox to compute the joint spectral radius. In *Proceedings of the 17th international conference on Hybrid systems: computation and control*, pages 151–156, 2014.

[47] J. Wang, M. Maggio, and V. Magron. SparseJSR: A Fast Algorithm to Compute Joint Spectral Radius via Sparse SOS Decompositions. *Proceedings of the American Control Conference (ACC) New-Orleans*, 2021. Accepted for publication.

[48] Jie Wang, Victor Magron, and Jean-Bernard Lasserre. TSSOS: A Moment-SOS hierarchy that exploits term sparsity. *SIAM Journal on Optimization*, 31(1):30–58, 2021.

[49] Yu Wang, Nima Roohi, Geir E Dullerud, and Mahesh Viswanathan. Stability of linear autonomous systems under regular switching sequences. In *53rd IEEE Conference on Decision and Control*, pages 5445–5450. IEEE, 2014.

[50] Henry Wolkowicz, Romesh Saigal, and Lieven Vandenberghe. *Handbook of semidefinite programming: theory, algorithms, and applications*, volume 27. Springer Science & Business Media, 2012.

[51] Shih-Lun Wu, Ching-Yuan Bai, Kai-Chieh Chang, Yi-Ting Hsieh, Chao Huang, Chung-Wei Lin, Eunsuk Kang, and Qi Zhu. Efficient system verification with multiple weakly-hard constraints for runtime monitoring. In Jyotirmoy Deshmukh and Dejan Ničković, editors, *Runtime Verification*, pages 497–516. Springer International Publishing, 2020.

[52] Xiangru Xu and Behcet Acikmese. Approximation of the constrained joint spectral radius via algebraic lifting. *IEEE Transactions on Automatic Control*, 2020.

[53] Xiangru Xu and Yiguang Hong. Matrix expression and reachability analysis of finite automata. *Journal of Control Theory and Applications*, 10(2):210–215, 2012.

[54] Kemin Zhou and John Comstock Doyle. *Essentials of Robust Control*. Prentice Hall international editions. Prentice Hall, 1998.