Safety-aware Flexible Schedule Synthesis for Cyber-Physical Systems using Weakly-Hard Constraints

Shengjie Xu, Bineet Ghosh, Clara Hobbs, P. S. Thiagarajan, and Samarjit Chakraborty
Department of Computer Science, The University of North Carolina (UNC) at Chapel Hill, USA

ABSTRACT

With the emergence of complex autonomous systems, multiple control tasks are increasingly being implemented on shared computational platforms. Due to the resource-constrained nature of such platforms in domains such as automotive, scheduling all the control tasks in a timely manner is often difficult. The usual requirement — that all task invocations must meet their deadlines — stems from the isolated design of a control strategy and its implementation (including scheduling) in software. This separation of concerns, where the control designer sets the deadlines, and the embedded software engineer aims to meet them, eases the design and verification process. However, it is not flexible and is overly conservative. In this paper, we show how to capture the deadline miss patterns under which the *safety* properties of the controllers will still be satisfied. The allowed patterns of such deadline misses may be captured using what are referred to as "weakly-hard constraints." But scheduling tasks under these weakly-hard constraints is non-trivial since common scheduling policies like fixed-priority or earliest deadline first do not satisfy them in general. The main contribution of this paper is to automatically synthesize schedules from the safety properties of controllers. Using real examples, we demonstrate the effectiveness of this strategy and illustrate that traditional notions of schedulability, *e.g.*, utility ratios, are not applicable when scheduling controllers to satisfy safety properties.

1 INTRODUCTION

The core functionalities of many emerging autonomous systems, like autonomous vehicles or robots, are implemented as a collection of feedback control loops. Their design starts with mathematically determining a control strategy, followed by implementing that strategy in software [14]. The former belongs to the domain of control theory and the latter to real-time and embedded systems. In this process, the control engineers assume certain deadlines that the control tasks need to satisfy for them to behave as desired, and the embedded systems engineers schedule them to meet those deadlines [16]. While this ensures a clean separation of concerns, allowing the two groups of engineers to work independently, this process is inflexible and overly conservative. It is posing a problem as the volume of software in autonomous systems continues to grow and multiple control tasks now need to be scheduled on shared resources [3, 25]. Meeting *all* deadlines requires making pessimistic decisions because of many reasons [7], such as the difficulty in estimating the worst case execution times (WCET) of tasks [1].

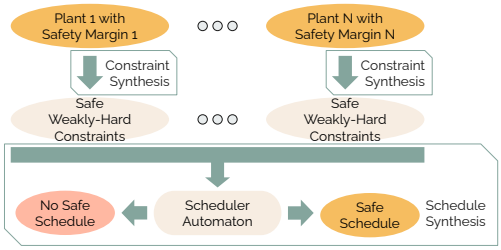

Figure 1: Outline of the proposed scheme.

Here, we show that instead of attempting to meet all deadlines, if the focus is shifted to a "system-level" property like control safety, then certain deadlines can be missed while still satisfying this property. Which deadline hit/miss patterns are acceptable has recently been actively studied from a control stability perspective [13, 15, 21, 22, 26, 28], which may be seen as a form of *safety* property. Other more general safety properties such as the maximum deviation from a nominal behavior have also been studied [5, 10]. While these papers have studied the deadline hit/miss patterns that guarantee the satisfaction of a given safety property, it is not clear *how to schedule* a given set of tasks such that each task respects its corresponding deadline hit/miss pattern. In the absence of such a scheduling policy it is not possible to exploit the flexibility that arises from the fact that deadline misses can be tolerated.

The ***main contribution*** of this paper is a technique for automatically synthesizing such schedules from a collection of feedback controllers and their associated safety properties. Instead of the safety property being *stability*, where our proposed technique may also be applied, we instead focus on a more general safety property defined by the maximum deviation from a *nominal behavior*. Given an initial state of the system (plant + controller) we define the nominal behavior as the trajectory in the state space where all the controller inputs meet their deadlines (*i.e.*, all the feedback control inputs are applied before the end of the sampling period). If some of the control inputs cannot be computed (and hence applied) within the sampling period (*i.e.*, there is a deadline miss) then the trajectory followed by the system will deviate from this nominal trajectory. As long as the deviation is not more than a specified bound, the behavior of the system is deemed to be *safe*. This notion of safe behaviors is a natural one (in the concluding section we point to more complex behavioral notions). For instance, the nominal trajectory might represent the pre-determined path that an autonomous vehicle should follow and safe behaviors will denote the paths the vehicle can take, without hitting any obstacles.

Thiagarajan is also affilifated with the Chennai Mathematical Institute, India. This work was supported by the NSF grant #2038960.

ASPDAC '23, January 16–19, 2023, Tokyo, Japan

ACM ISBN 978-1-4503-9783-4/23/01…$15.00
https://doi.org/10.1145/3566097.3567848

Our proposed schedule synthesis scheme is illustrated in Figure 1. Given a plant + controller and a desired safety property (*i.e.*, the maximum allowed deviation from its nominal behavior), we first derive the pattern of allowed deadline hits and misses. This is captured as a *weakly-hard constraint* $\binom{m}{k}$ that specifies how many deadlines m must be met within any window of k consecutive samples. Note that there could be multiple such weakly-hard constraints that satisfy the safety property of a controller. Given a set of controllers and a set of weakly-hard constraints derived for each controller, we develop an automata-theoretic technique to compute a schedule for them. Such a schedule specifies which controllers should be run in each period (and hence meet their deadlines) and which cannot be run (*i.e.*, they miss their deadlines). It is important to note that such schedules may not be captured using standard scheduling policies like earliest deadline first or fixed priority.

This work is broadly related to the topic of scheduling control tasks [6], and in particular to a number of recent studies [5, 10, 15] on *checking* whether a safety property (including control stability) is satisfied for some given deadline hit/miss pattern. Here, we leverage those results and solve the inverse problem, (*viz.*,) synthesizing the hit/miss patterns (or weakly-hard constraints) and further use them for schedule synthesis. As for other related work, the authors of [30] propose systematic methods of specifying weakly-hard constraints. They also carry out a detailed analysis of the dominance relations between various constraint types and present an analysis tool called WeaklyHard.jl. Scheduling related studies involving weakly hard constraints have appeared in [8, 9, 21, 27]. In [8] the problem of scheduling multiple data streams with $\binom{m}{k}$ constraints using a priority based scheme is studied with the aim of reducing the probability of constraint violations. The work in [9] develops a technique for bounding the number of deadline misses in end-to-end task chains that have task dependencies. The problem of verifying if a $\binom{m}{k}$ constraint is being met in a uniprocessor setting for constrained-deadline periodic systems is investigated in [27] and an overapproximation scheme is presented. Finally, the authors of [21] develop a Deadline-Miss-Aware-Controller (DMAC) which estimates deadline misses based on probabilistic execution times derived through simulations. The goal is to provide probabilistic guarantees for the performance of the controller.

The rest of this paper is organized as follows. In Section 2, we introduce the system model and weakly-hard constraints with which we model the deadline hit/miss behaviors. In Section 3, we formally define the two main problems of this work, namely constraint synthesis and schedule synthesis. We then propose solutions to these two problems in Sections 4 and 5, respectively. We evaluate our solutions with a case study in Section 6. Finally, we provide concluding remarks in Section 7.

2 SYSTEM MODELLING

2.1 The State-Space Model

Our system model is a discrete time-invariant linear dynamical system with state feedback control. We assume it to be of the form:

$$x[t+1] = Ax[t] + Bu[t], \tag{1}$$

where $A \in \mathbb{R}^{n\times n}$, and $B \in \mathbb{R}^{n\times p}$. The control input u is computed by a periodic real-time task running on a processor, and is assumed to be of the form:

$$u[t] = Kx[t-1], \tag{2}$$

where $K \in \mathbb{R}^{p\times n}$. We assume that the new control input is always applied at the deadline of the control job, and the deadline is assumed to be one sampling period from the release time of the job [12]. In other words, the system state at time $t-1$ is sampled and used to compute the control input for time t, where the state and the control input are computed according to Eqs. (1) and (2).

2.2 Safe Behaviors

We consider the behavior of the plant only over a finite time horizon H. Thus the states of the plant will be recorded at time points $0, 1, \ldots, H$. We also assume the initial state of the system is $z[0] \in \mathbb{R}^n$. Then the *nominal trajectory* of the plant is the sequence of states of length H of the form $z[0], z[1] \ldots, z[H]$, where $z[t] = Az[t-1] + Bu[t-1]$ and $u[t] = Kz[t-1]$, for $0 < t \leq H$. Intuitively, it is the trajectory that results when there are no deadline misses. We next wish to define the set of behaviors that can be tolerated in terms of how far they can deviate from the nominal trajectory. To start with, we let $\mathcal{T}^H$ be sequences of length H over $\mathbb{R}^n$ of the form $\tau = x[0], x[1] \ldots, x[H]$ with $x[0] = z[0]$. We use $\tau[j]$ to denote the j-th member of the sequence, *i.e.*, $\tau[j] = x[j]$. Since H will be clear from the context we shall write $\mathcal{T}$ instead of $\mathcal{T}^H$ in what follows. Intuitively, $\mathcal{T}$ denotes the set of all possible trajectories of length H in the state space that start from $z[0]$. Clearly the nominal trajectory, denoted from now on as τ_{nom}, is a member of $\mathcal{T}$. To quantify deviations from the nominal trajectory, we use the Euclidean distance, denoted $\mathrm{dis}(\cdot)$ to measure the distance between two points in $\mathbb{R}^n$. In other words, for $x, y \in \mathbb{R}^n$, $\mathrm{dis}(x, y) = (\sum_{i=1}^{n}(x^i - y^i)^2)^{1/2}$, where x^i and y^i denote the i-th element of the vector x and y respectively. This induces a distance between any pair of members of $\mathcal{T}$, also denoted as $\mathrm{dis}(\cdot)$, given by $\mathrm{dis}(\tau, \tau') = \max\{\mathrm{dis}(\tau[j], \tau'[j]) \mid 0 \leq j \leq H\}$.

We now fix a maximum deviation $d_{max} > 0$, a rational number in $\mathbb{R}$. This leads to the set of safe trajectories $\mathcal{T}_{safe}^{d_{max}} \subset \mathcal{T}$, defined as $\mathcal{T}_{safe}^{d_{max}} = \{\tau \mid \mathrm{dis}(\tau, \tau_{nom}) \leq d_{max}\}$. We shall from now on write $\mathcal{T}_{safe}$ instead of $\mathcal{T}_{safe}^{d_{max}}$ since d_{max} will be clear from context. Clearly, the nominal trajectory is a member of $\mathcal{T}_{safe}$.

2.3 Weakly-Hard Constraints

The control input u is computed by a periodic real-time task running on a processor. Suppose $x[t]$ is the plant state and $u[t]$ is the control input at time t. When $x[t-1]$ is read, a software job is released to compute $u[t]$, which is then applied to the physical plant at time t if the job completes within its deadline. It is also possible that the job will miss its deadline and not compute a control input in time. In this case, the controller must decide on the control signal to be sent to the system. In the literature [15] two policies have been often considered: (1) **Hold**, in which the previous control signal is sent as the current input, and (2) **Zero**, where the control input 0 is sent. In this paper, we focus on the **Hold** policy, though our methods are compatible with **Zero** as well. Clearly, when deadline misses occur, the behavior of the plant trajectory will deviate from the nominal trajectory since the "correct" control inputs are not received.

The work in [2] proposes succinct and systematic methods of limiting how many deadlines can be missed by control software jobs before it is considered to be a violation. Among them, the $\overline{\binom{m}{k}}$ model—which requires that at most m deadlines can be missed in any k consecutive executions of a task—has been studied in a number of settings, including schedulability analysis, formal verification, and runtime monitoring, with [11, 20, 29] as recent examples.

Following the notation in [2], we use constraints of the type $\binom{m}{k}$, which states that at least m deadlines must be met in any k consecutive invocations of the task. This is equivalent to the constraint $\overline{\binom{k-m}{k}}$. If we represent the hit/miss patterns using a bit string, where 0 represents a deadline miss and 1 a hit, then all hit/miss patterns that comply with the constraint $\binom{m}{k}$ form a regular language over the alphabet $\{0, 1\}$ [30]. We denote this language as $\mathcal{L}_{(m,k)}$.

Plant Behaviors under Weakly-Hard Constraints. Our goal is to synthesize weakly-hard constraints of the form $\binom{m}{k}$ under which the plant behavior remains safe. To make this precise, suppose $\sigma \in \{0,1\}^H$ is a sequence of length H representing a pattern of hits and misses. As before, 0 denotes a deadline miss and 1, a hit. Then starting from $z[0]$ we can use Eq. (1) to compute the sequence of plant states and Eq. (2) to compute control inputs (or use the last control input if there was a deadline miss) . We denote the resulting plant trajectory as τ_σ. This leads to $\mathcal{T}_{(m,k)} = \{\tau_\sigma \mid \sigma \in \mathcal{L}_{(m,k)}\}$. We call the plant *safe* under $\binom{m}{k}$ if and only if $\mathcal{T}_{(m,k)} \subseteq \mathcal{T}_{safe}$.

3 PROBLEM STATEMENT

The problems we study in this work can now be stated as follows.

Problem 1 (Constraint Synthesis). *Given a dynamical system with the initial state $z[0]$ as specified above, a time horizon H, and the allowed maximum deviation d_{max} from the nominal behavior, find a set of weakly-hard constraints of the form $\binom{m}{k}$ such that the plant behavior is safe under each of these constraints.*

Since H may be large, allowing all weakly-hard constraints is not realistic. Hence, we assume that we are given a maximum window size k_{max} such that $k_{max} \ll H$. Thus we are required to synthesize all constraints of the form $\binom{m}{k}$ with $m \leq k$ and $k \leq k_{max}$. Suppose we are given a set of plants, each with its own maximum allowed deviation. For convenience, we assume they have a common time horizon H. Solving Problem 1 results in a set of weakly-hard constraints for each plant. Then, we wish to also solve the following scheduling problem.

Problem 2 (Schedule Synthesis). *Given a set of N controllers $\{\mathbb{C}_i\}$, each with a set of weakly-hard constraints, and an implementation platform where at most $J < N$ controllers can be scheduled in each time slot, determine if a schedule exists where all the controllers can be scheduled without violating their safety constraints over the time horizon H. Furthermore, synthesize a schedule if one exists.*

We propose solutions to these problems in the next two sections. Figure 1 shows how the two problems below are connected.

4 CONSTRAINT SYNTHESIS

We are seeking to synthesize all constraints of the form $\binom{m}{k}$ with $m < k$ and $k \leq k_{max}$ under which the plant is safe. This set of constraints can be narrowed down based on the following observations (see [2] for a more detailed analysis). Suppose $\binom{m}{j}$ and $\binom{m}{k}$ are two constraints with $j < k$. Then clearly $\mathcal{L}_{(m,j)} \subset \mathcal{L}_{(m,k)}$, and hence if the plant is safe under $\binom{m}{k}$ then it will also be safe under $\binom{m}{j}$. Next, suppose $\binom{m}{k}$ and $\binom{\ell}{k}$ are two constraints such that $m < \ell$. Then $\mathcal{L}_{(\ell,k)} \subset \mathcal{L}_{(m,k)}$. Hence, if the plant is safe under $\binom{m}{k}$, then it will also be safe under $\binom{\ell}{k}$.
Therefore, we wish to synthesize a set of constraints:

$$\left\{\binom{m_k}{k} \mid k \in \{1, \ldots, k_{max}\}\right\}$$

such that for all k, if the plant is safe under $\binom{m'}{k}$, then $m' \geq m_k$.

Suppose we are trying to check if the given plant is safe under a generic constraint $\binom{m}{k}$. This entails checking if $\mathcal{T}_{(m,k)} \subseteq \mathcal{T}_{safe}$. This is difficult to do directly since there will be exponentially (in H) many strings to be checked. Hence we will follow a scheme developed in [10] to check this in an overapproximate manner. In other words, if we decide that the plant is safe under $\binom{m}{k}$, this is guaranteed to be the case but there may be a smaller value $m' < m$ such that the plant is also safe under $\binom{m'}{k}$. To explain this scheme, we first define $d\left(\binom{m}{k}\right) = \max\{\text{dis}(\tau, \tau_{nom}) \mid \tau \in \mathcal{T}_{(m,k)}\}$.

The work in [10] proposes several methods for computing an upper bound $\widehat{d}$ for $d\left(\binom{m}{k}\right)$. We use the `BoundedTree` algorithm and make the following modification to suit our problem. The weakly-hard constraints considered in [10] are in the form of $\langle \ell \rangle$, indicating that no more than ℓ deadline misses can occur consecutively in a trajectory of length H. This can be easily modified to work with weakly-hard constraints of the form $\binom{m}{k}$. Using the modified `BoundedTree` algorithm, we approximate an upper bound $\widehat{d} \geq d\left(\binom{m}{k}\right)$. If $\widehat{d} \leq d_{max}$ then we conclude that the plant is safe under $\binom{m}{k}$. Otherwise we conclude that it is not.

4.1 Constructing Safety Constraints List

We can now tackle the problem of estimating the set of constraints:

$$\left\{\binom{m_1}{1}, \binom{m_2}{2}, \ldots, \binom{m_{k_{max}}}{k_{max}}\right\}$$

such that for j in $\{1, 2, \ldots, k_{max}\}$, if the plant is safe under $\binom{m'}{j}$ then $m' \geq m_j$. We propose Algorithm 1 for computing this set of constraints. The heart of the algorithm is the overapproximation function `deviationUB`, using which an upper bound is computed for each candidate constraint $\binom{m_j}{j}$. A naïve way of building $\left\{\binom{m_1}{1}, \binom{m_2}{2}, \ldots, \binom{m_{k_{max}}}{k_{max}}\right\}$ is to loop over k from 1 to k_{max} and m from 1 to k. Our observations made earlier in the section suggest two improvements: (1) if $\binom{m}{k}$ is unsafe, then $\binom{m}{k'}$ is unsafe for any $k' > k$; (2) if $\binom{m}{k}$ is safe, then $\binom{m'}{k}$ is safe for any $m' > m$. In both cases no further computation is required. We also note that $m = k$ implies that there are no deadline misses, i.e., $\binom{j}{j}$ is the nominal behavior. Thus we can further restrict that $m < k$ and $k \geq 2$.

In Algorithm 1, the outer loop at Line 4 loops over k from 2 to k_{max}, while the inner loop at Line 5 loops over m from 1 to k. Note that m increases monotonically for all values of k. For each combination of m and k, the deviation upper bound $\widehat{d}$ is computed for $\binom{m}{k}$ (Line 6) and compared with maximum deviation d_{max} (Line 7). If $\widehat{d} \leq d_{max}$, then $\binom{m}{k}$ is added to results and k is incremented (Line 8); otherwise, m is incremented (Line 10). Some examples are shown in Table 1 in Section 6. For now, if we focus on

Algorithm 1: Constructing the set of safe constraints for a given controller

```
function constraintSynthesis(ℂ, d_max, z[0], k_max)
    input  : The plant, maximum deviation d_max, the initial
             state z[0], and maximum window size k_max
    output : List of weakly-hard constraints that satisfies
             safety requirement
    result ← {} ;
    m ← 1 ;
    for k ← 2 to k_max do
        while m < k do
            d̂ ← deviationUB(m, k, ℂ, z[0]) ;
            if d̂ ≤ d_max then
                result ← append( result, (m k)) ;
                break
            m ← m + 1 ;
    return result ;
```

just the first model, namely, the RC network, k_{max} for this system is 6. ✓ denotes a safe constraint (for instance, $\binom{4}{2}$) and × an unsafe constraint (for instance, $\binom{5}{2}$).

5 SCHEDULE SYNTHESIS

As noted earlier, a weakly-hard constraint can be associated with a regular language over $\{0,1\}$ where the strings in this language are hit/miss patterns that satisfy this constraint. Clearly, the set of safety constraints we have synthesized for a plant can be represented as regular language which is the union of the regular languages representing the constraints. In this section, we first build an automaton-based representation $\mathbb{A}_i$ of the weakly-hard constraints for each controller $\mathbb{C}_i$. We then construct a scheduler automaton $\mathbb{A}^S$ to check for the existence of safe schedules and generate one such schedule if they exist.

5.1 Controller Automaton

The automaton $\mathbb{A}_{(m,k)}$, accepting the language $\mathcal{L}_{(m,k)}$, will be of the form $\langle L, \Sigma, T, L_f, \ell_0 \rangle$ where:

- L set of locations, $L = \{0,1\}^k$;
- Σ input alphabet, $\Sigma = \{0,1\}$;
- T transition function, $T : L \times \Sigma \to L$;
- L_f accepting locations of the automaton, $L_f \subset L$;
- ℓ_0 initial location of the automaton with $\ell_0 = \{1\}^k$.

The locations of the automaton $\ell \in \{0,1\}^k$ are strings representing the sliding window of size k over the hit/miss patterns, corresponding to the k consecutive invocations of the software task. The starting location $\ell_0 = \{1\}^k$ assumes that there has been no deadline misses at system start. Let $\texttt{countOnes}(s)$ be a function that returns the number of 1s in a string s over $\{0,1\}$. The accepting locations L_f are the ones satisfying the weakly-hard constraint for that specific window, *i.e.*, $\ell \in L_f$ if and only if $\texttt{countOnes}(\ell) \geq m$. The transition function T is defined in the expected way. As an example, the automaton corresponding to the the weakly-hard constrain of $\binom{1}{2}$ is shown in Fig. 2.

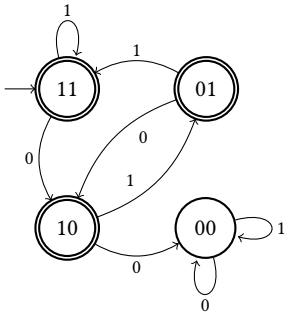

Figure 2: The automaton modelling the weakly-hard constraint $\binom{1}{2}$.

For each controller $\mathbb{C}_i$, a single $\mathbb{A}_i$ can be constructed by taking the union of all languages corresponding to weakly-hard safe constraints that have been synthesized for that controller (as per Problem 1). We assume that we have N plants and controllers. Thus we will have a set of automata $\{\mathbb{A}_i\}$. These automata will then be used to construct the scheduler automaton. Our construction will ensure that a schedule exists if and only if the language accepted by the scheduler automaton is non-empty.

5.2 Scheduler Automaton

Our goal is to schedule the jobs of each controller's task. We assume that the processor makes available H time slots one after the another and in each slot at most J jobs can run concurrently. As a first attempt at scheduling tasks with weakly-hard constraints, we assume a restricted setting where all the controllers have the same period and the deadline is its period for each controller. Thus if the job tsk_i^j, namely the j-th job belonging to the task tsk_i is not scheduled in the j-th slot then it suffer a deadline miss. On the other hand, if it is scheduled then it will meet its deadline. Based on this setting, we define the scheduler automaton as follows.

Definition 1. *A **scheduler automaton** $\mathbb{A}^S$ for a set of N controllers whose constraints are represented by the automata of the form $\mathbb{A}_i = \langle L^i, \Sigma, T^i, L^i_f, \ell^i_0 \rangle$, where at most J controllers can be scheduled in each time slot, is defined as an automaton $\langle L^S, \Sigma^S, T^S, L^S_f, \ell^S_0 \rangle$:*

- L^S *set of locations,* $L^S = \prod_i L^i$;
- Σ^S *input alphabet,* $\Sigma^S \subset \{0,1\}^N$. *A sequence* $\sigma \in \{0,1\}^N$ *is in* Σ^S *if and only if* $\texttt{countOnes}(\sigma) \leq J$;
- T^S *transition function,* $T^S(\ell, \sigma) = \prod_i T^i(\ell^i, \sigma^i)$;
- L^S_f *accepting locations of the automaton,* $L^S_f = \prod_i L^i_f$;
- ℓ^S_0 *initial location of the automaton,* $\ell^S_0 = \prod_i \ell^i_0$.

Specifically, the new set of locations L^S is a Cartesian product of the controller automaton locations: $L^S = L^1 \times L^2 \times \cdots \times L^N$, and each location $\ell \in L^S$ is a tuple of individual locations from each controller: $\ell = \langle \ell^1, \ell^2, \ldots, \ell^N \rangle$. The set of actions $\Sigma^S \subset \{0,1\}^N$ now captures hits and misses for all controllers. Below, we use superscripts ℓ^i to denote i-th element of ℓ, and σ^i to denote the i-th position of σ. The i-th controller is scheduled when $\sigma^i = 1$. Since we can only schedule J jobs in each slot, an action σ is valid if and only if $\texttt{countOnes}(\sigma) \leq J$. For example, $N = 3, J = 1$ results in $\Sigma^S = \{000, 001, 010, 100\}$. $\sigma = 010$ indicates that only the second controller is scheduled. The transition function T^S returns the

product of individual transition functions $\prod_i T^i$. Let $\sigma \in \Sigma^S$ be a valid action for the scheduler automaton $\mathbb{A}^S$, then the transition function T^S becomes:

$$T^S(\ell, \sigma) = \langle T^1(\ell^1, \sigma^1), T^2(\ell^2, \sigma^2), \ldots, T^N(\ell^N, \sigma^N) \rangle.$$

The set of accepting locations L_f is also a Cartesian product of the individual accepting locations. A location $\ell \in L_f^S$ if and only if $\ell^i \in L_f^i$ for all $i \in [1, N]$. Intuitively, this means that the schedule is valid only if all the controllers operate within their safety margin; if any of the controller automaton $\mathbb{A}_i$ transition to a non-accepting (unsafe) location, the scheduler automaton will also transition to a non-accepting location. Once we have constructed the scheduler automaton, we can check for the existence of schedules by running emptiness check on the scheduler automaton.

6 EXPERIMENTAL RESULTS

We have implemented the constraint synthesis and scheduler synthesis techniques using *Julia*. We used the 5 systems described in Section 6.1 for our experiments. Given a maximum allowed deviation for each system, we used our constraint synthesis technique to compute a set of safe weakly-hard constraints for each controller. We then checked, using our scheduler synthesis technique, if the corresponding controllers with their synthesized constraints could be scheduled on a shared platform. In doing so, we assumed that only two jobs can be scheduled in any single slot (*i.e.*, $J = 2$).

6.1 Plant Models

We note that the controllers of all the five systems, described in the rest of the section, have the same period, namely, $p = 20ms$

6.1.1 RC Network (RC). Our first model is a resistor-capacitor network [4] with the following model.

$$x[t+1] = \begin{bmatrix} 0.8870 & 0.01871 \\ 0.003743 & 0.9861 \end{bmatrix} x[t] + \begin{bmatrix} 0.09433 \\ 0.01012 \end{bmatrix} u[t]$$

6.1.2 F1Tenth Car (F1). Our second model is the linearized motion of an F1Tenth model car [18]:

$$x[t+1] = \begin{bmatrix} 1 & 0.13 \\ 0 & 1 \end{bmatrix} x[t] + \begin{bmatrix} 0.02559 \\ 0.3937 \end{bmatrix} u[t]$$

Our next three plant models are selected from [23] and represent components in the automotive domain.

6.1.3 DC Motor (DC). Our third model is the speed control for DC motor adapted from [17]:

$$x[t+1] = \begin{bmatrix} 0.8187 & 0.01776 \\ -0.0003551 & 0.9608 \end{bmatrix} x[t] + \begin{bmatrix} 0.0003696 \\ 0.03921 \end{bmatrix} u[t]$$

6.1.4 Car Suspension (CS). Our fourth model is a suspension system adapted from [19]:

$$x[t+1] = \begin{bmatrix} 0.9988 & 0.01937 & 0.000923 & 0.000549 \\ -0.1111 & 0.9432 & 0.06715 & 0.04547 \\ 0.01082 & 0.00549 & 0.978 & 0.01165 \\ 0.8878 & 0.4547 & -1.82 & 0.3012 \end{bmatrix} x[t] + \begin{bmatrix} 0.0113 \\ 0.9534 \\ 0.2441 \\ -13.04 \end{bmatrix} u[t]$$

Table 1: Synthesized constraints for the 5 controllers from Section 6.1.

Model	Window Size (k)	Minimum Hits (m) 1	2	3	4	5
RC network	2	✓	–	–	–	–
	3	✓	✓	–	–	–
	4	×	✓	✓	–	–
	5	×	×	✓	✓	–
	6	×	×	×	✓	✓
F1 Tenth	2	✓	–	–	–	–
	3	×	✓	–	–	–
	4	×	×	✓	–	–
	5	×	×	×	✓	–
	6	×	×	×	×	✓
DC Motor	2	✓	–	–	–	–
	3	✓	✓	–	–	–
	4	✓	✓	✓	–	–
	5	×	✓	✓	✓	–
	6	×	×	✓	✓	✓
Car Suspension	2	✓	–	–	–	–
	3	×	✓	–	–	–
	4	×	✓	✓	–	–
	5	×	✓	✓	✓	–
	6	×	×	✓	✓	✓
Cruise Control	2	✓	–	–	–	–
	3	✓	✓	–	–	–
	4	✓	✓	✓	–	–
	5	✓	✓	✓	✓	–
	6	×	✓	✓	✓	✓

6.1.5 Cruise Control (CC). Our final model is a cruise control system adapted from [24]:

$$x[t] = \begin{bmatrix} 1.0 & 0.01999 & 0.0001996 \\ -0.001207 & 0.9989 & 0.01995 \\ -0.1206 & -0.1066 & 0.9942 \end{bmatrix} x[t] + \begin{bmatrix} 3.298 \times 10^{-6} \\ 0.0004945 \\ 0.0494 \end{bmatrix} u[t]$$

Each plant model was discretized with a common period of $p = 20$ ms. Controllers were then designed for the discretized state-space models with a one-period delay.

6.2 Safe Schedule Synthesis

We first synthesized the set of safe weakly-hard constraints for each controller. Then, we checked for a schedule that respects the constraints for all controllers. We exhibit one such schedule below.

Constraint Synthesis. For each controller, we ran the method outlined in Section 4, with time horizon $H = 100$. The maximum window size k_{max} was set to be 6 for all the controllers. The results are presented in Table 1, where we also report the maximum deviation d_{max} used to synthesize the constraints, for each controller.

Schedule Synthesis. We next used the controllers of the five systems and attempted to synthesize a schedule (with $J = 2$). The results are presented in Table 2, showing an execution sequence of the scheduler automaton. As shown, we have successfully found a *schedule* for the five controllers, where each controller satisfies its corresponding safety constraint. As shown in Table 2, one example of a valid schedule is as follows: From $t = 1$ to $t = 19$, the schedule is the corresponding action $\sigma[t]$, where the i-th controller is scheduled at time step t if $\sigma[t]^i = 1$. From $t = 20$ to $t = 100$, the schedule is repeated from $t = 6$ to $t = 19$, *viz.*, the schedule at time

Table 2: Synthesized schedule for the five controllers outlined in Section 6.1. A 1 at the i-th position of action $\sigma[t]$ indicates that the i-th controller is scheduled at time step t.

Step (t)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Action ($\sigma[t]$)	01010	10100	01001	10010	01100	**10010**	01100	10001	01010	10100	01010	10001	01100	10010	01010	10100	01001	10010	01100	(same as $\sigma[6]$)

$t \in [20, 100]$ can be analytically given as $\sigma[((t-20) \bmod 14) + 6]$. We note that this is not the *only* valid schedule for this particular set of controllers. The accepting runs of the scheduler automaton represent the set of all the valid schedules.

Interestingly, this case study highlights that when weakly-hard constraints are involved, traditional expectations involving utilization, scheduling policies and schedulability do not apply, even in our simple setting. To illustrate this, assume that each controller has just one weakly-hard constraint. Then, it is not the case that the schedulability of $\{\mathbb{C}_i\}$ is implied by the utilization ratio satisfying $\sum_{\mathbb{C}\in\{\mathbb{C}_i\}} \mathrm{U}(\mathbb{C}) < J$, where $\mathrm{U}(\mathbb{C})$ is defined as $\frac{m}{k}$ for the controller with constraint $\binom{m}{k}$. Consider five controllers each with one weakly-hard constraint: $\binom{1}{2}, \binom{1}{2}, \binom{1}{3}, \binom{2}{5}, \binom{1}{4}$. Here $\sum_{\mathbb{C}\in\{\mathbb{C}_i\}} \mathrm{U}(\mathbb{C}) = \frac{1}{2} + \frac{1}{2} + \frac{1}{3} + \frac{2}{5} + \frac{1}{4} = \frac{119}{60} < 2$. However, it is easy to check this set of tasks is not schedulable. On the other hand, one may expect that if a schedule $\mathbb{S}$ exists, then the earliest deadline first (EDF) scheduler (where a deadline refers to the maximum number of misses until the constraint is violated) will produce a valid schedule. However, considering the constraints $\binom{1}{2}, \binom{1}{2}, \binom{1}{4}, \binom{3}{6}, \binom{1}{5}$, it is easy to check that scheduling under EDF will fail, but our scheduler synthesis method produces a valid schedule.

7 CONCLUDING REMARKS

In this work, we have linked the safety properties of control systems to the schedules of their controllers implemented on a shared platform. Specifically, we consider safe behaviors of a system to be those in which the system deviates from its nominal behavior (induced by its control software always delivering its results on time) at most by a given bound. We then automatically synthesize a schedule for the controllers of a set of such systems so their behaviors are guaranteed to be safe. The bridge connecting the safe behaviors and schedules are weakly-hard constraints, which specify the pattern of deadline misses that a system can tolerate while maintaining the safety of its behaviors. Concretely, we first synthesize a set of weakly hard constraints for each controller under which the behavior of the system is guaranteed to remain safe. We do so in a safe, overapproximate manner guaranteeing safety of the plant, since exactly determining the constraints is computationally infeasible. Then given a set of weakly-hard constraints for each controller, we synthesize a schedule for a set of controllers on a shared platform, so long as such a schedule exists.

An important observation here is that scheduling under weakly-hard constraints is inherently challenging and does not seem to be amenable to analyses developed for standard scheduling policies, like earliest deadline first. As our examples have shown, even in the simple setting that we considered, traditional ideas based on utility ratios and the earliest deadline first policy do not work. It will be interesting to study this issue systematically in the future. In addition, we have only considered a safety property of plants here. For autonomous systems, one often additionally requires *liveness* properties, such as, *"the robot should visit each designated station at least once (while avoiding obstacles) when it moves from the starting point to the destination."* It will therefore be fruitful to extend the our framework to such richer settings.

Acknowledgements: We thank the anonymous reviewers for their useful feedback that helped in improving the paper.

REFERENCES

[1] P. Axer, et al. 2014. Building Timing Predictable Embedded Systems. *ACM Trans. Embed. Comput. Syst.* (2014).
[2] G. Bernat, et al. 2001. Weakly hard real-time systems. *IEEE Trans. Comput.* 50, 4 (2001).
[3] W. Chang and S. Chakraborty. 2016. Resource-aware Automotive Control Systems Design: A Cyber-Physical Systems Approach. *Found. Trends Electron. Des. Autom.* 10, 4 (2016).
[4] R. A. Gabel and R. A. Roberts. 1980. *Signals and Linear Systems* (second ed.). John Wiley & Sons.
[5] B. Ghosh, et al. 2022. Statistical Hypothesis Testing of Controller Implementations Under Timing Uncertainties. In *RTCSA*.
[6] D. Goswami, et al. 2011. Re-engineering cyber-physical control applications for hybrid communication protocols. In *DATE*.
[7] D. Goswami, et al. 2014. Relaxing Signal Delay Constraints in Distributed Embedded Controllers. *IEEE Trans. Control. Syst. Technol.* 22, 6 (2014).
[8] M. Hamdaoui and P. Ramanathan. 1995. A dynamic priority assignment technique for streams with (m, k)-firm deadlines. *IEEE Trans. Comput.* 44, 12 (1995).
[9] Z. Hammadeh, et al. 2017. Bounding Deadline Misses in Weakly-Hard Real-Time Systems with Task Dependencies. In *DATE*.
[10] C. Hobbs, et al. 2022. Safety Analysis of Embedded Controllers under Implementation Platform Timing Uncertainties. In *EMSOFT*.
[11] C. Huang, et al. 2019. Formal Verification of Weakly-Hard Systems. In *HSCC*.
[12] L. Ju, et al. 2008. Performance debugging of Esterel specifications. In *CODES+ISSS*.
[13] S. Linsenmayer and F. Allgöwer. 2017. Stabilization of networked control systems with weakly hard real-time dropout description. In *CDC*.
[14] M. Lukasiewycz, et al. 2013. System architecture and software design for electric vehicles. In *DAC*.
[15] M. Maggio, et al. 2020. Control-System Stability Under Consecutive Deadline Misses Constraints. In *ECRTS*.
[16] A. Masrur, et al. 2010. VM-Based Real-Time Services for Automotive Control Applications. In *RTCSA*.
[17] W. C. Messner and D. M. Tilbury. 1998. Control tutorials for MATLAB and Simulink: a web-based approach. http://ctms.engin.umich.edu/CTMS
[18] M. O'Kelly, et al. 2020. F1tenth: An open-source evaluation environment for continuous control and reinforcement learning. *Proceedings of Machine Learning Research* 123 (2020).
[19] K. Osman, et al. 2009. Modelling and controller design for a cruise control system. *CSPA* (2009).
[20] P. Pazzaglia, et al. 2021. Adaptive Design of Real-Time Control Systems subject to Sporadic Overruns. In *DATE*.
[21] P. Pazzaglia, et al. 2019. DMAC: Deadline-Miss-Aware Control. In *ECRTS*.
[22] P. Pazzaglia, et al. 2018. Beyond the Weakly Hard Model: Measuring the Performance Cost of Deadline Misses. In *ECRTS*.
[23] D. Roy, et al. 2016. Multi-Objective Co-Optimization of FlexRay-Based Distributed Control Systems. In *RTAS*.
[24] R. Schneider, et al. 2011. Constraint-Driven Synthesis and Tool-Support for FlexRay-Based Automotive Control Systems. In *CODES+ISSS*.
[25] T. Sehnke, et al. 2017. Temporal properties in automotive control software. In *RTNS*.
[26] D. Soudbakhsh, et al. 2018. Co-Design of Arbitrated Network Control Systems With Overrun Strategies. *IEEE Trans. Control. Netw. Syst.* 5, 1 (2018).
[27] Y. Sun and M. D. Natale. 2017. Weakly Hard Schedulability Analysis for Fixed Priority Scheduling of Periodic Real-Time Tasks. *ACM Trans. Embed. Comput. Syst.* 16, 5s, Article 171 (2017).
[28] E. P. van Horssen, et al. 2016. Performance analysis and controller improvement for linear systems with (m, k)-firm data losses. In *ECC*.
[29] G. von der Brüggen, et al. 2018. Efficiently approximating the probability of deadline misses in real-time systems. In *ECRTS*.
[30] N. Vreman, et al. 2022. WeaklyHard.jl: Scalable Analysis of Weakly-Hard Constraints. In *RTAS*.