Statistical Approach to Efficient and Deterministic Schedule Synthesis for Cyber-Physical Systems

Shengjie Xu$^{1[0000-0003-1784-1386]}$, Bineet Ghosh$^{1[0000-0002-1371-2803]}$, Clara Hobbs$^{1[0000-0001-6046-9511]}$, Enrico Fraccaroli$^{1[0000-0002-9739-6501]}$, Parasara Sridhar Duggirala$^{1[0000-0002-8871-0298]}$, and Samarjit Chakraborty$^{1[0000-0002-0503-6235]}$

University of North Carolina at Chapel Hill, Chapel Hill NC 27599, USA

Abstract. Correctness of control implementations has relied on real-time guarantees that all control tasks would finish execution by the prescribed deadline. However, with increased complexity and heterogeneity in hardware, the worst-case execution time estimates are becoming very conservative. Thus, for efficient usage of hardware resources, some control tasks would have to miss some deadlines. Recent work has shown that a system can still abide by its safety requirements even after missing some of its deadlines. This paper investigates an approach to synthesize a scheduler for control tasks that miss some deadlines without compromising any of the safety requirements. Given that the number of possible schedules would increase combinatorially with the number of tasks involved, our scheduler synthesis uses an efficient automata representation to search for the appropriate schedule. We incorporate statistical verification techniques to construct this automaton and accelerate the search process. Statistical verification is advantageous compared to deterministic verification in the synthesis process in two ways: first, it enables us to synthesize schedules that would not be possible otherwise, and second, it drastically reduces the time taken to synthesize such a schedule. We demonstrate both these advantages through a case study of five controllers with different safety specifications sharing the same computational resource.

1 Introduction

Modern-day cars (and other autonomous systems) have several millions of lines of code deployed on various *electronic control unit* (ECUs). Each ECU implements a feedback control software task managing important components such as engine control, brake control, suspension and vibration control. The typical workflow for implementing these control software tasks is a two-step process. In the first step, a control designer would design the feedback function using principles of control design. In the second step, the embedded systems engineer

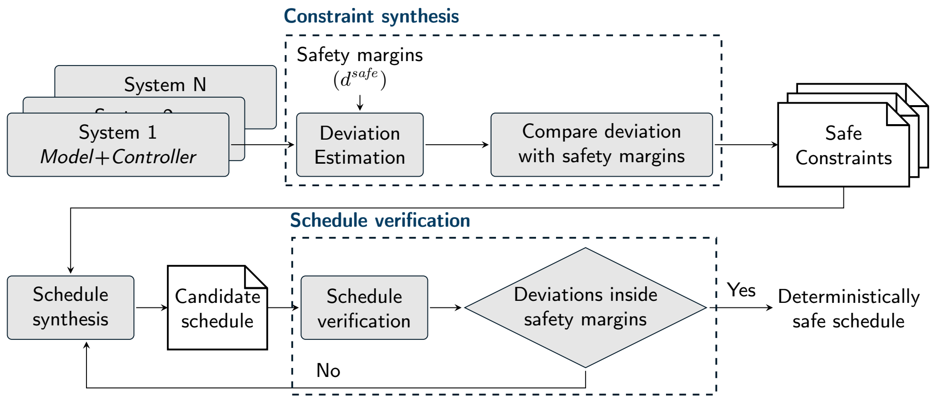

Fig. 1. Overview of the proposed method.

would schedule the control tasks such that each task would meet its prescribed deadline. This separation of concerns allows for communication between control designers and software engineers through task deadlines and allows for parallel development of software architecture and feedback control mechanisms.

However, the rapid increase in the volume of software being deployed in autonomous systems to enable additional *autonomous features* poses a challenge to this design flow. First, due to increased complexity in hardware, estimates of Worst-Case Execution Time (WCET) are overly conservative, and thus, applying traditional design flow with the estimated WCET would result in waste of computational resources. Second, scheduling multiple tasks on a shared computational resource with optimistic estimates of WCET would risk the system not satisfying its performance requirements. As a result, it is challenging to (a) ensure that none of the control tasks miss their deadlines and (b) synthesize a scheduler that tolerates these deadline misses while satisfying performance requirements. Further, automotive in-vehicle architectures are going away from one function per ECU or "federated", to multiple functions sharing resources, or "integrated" architectures. The clear trend is that future architectures will be less static than before, as indicated by developments like AUTOSTAR Adaptive and service-oriented paradigms. Thus, it is necessary to rethink the design flows for autonomous systems with new software and hardware architectures.

This paper proposes a new *correctness-by-construction* approach for synthesizing control implementations. *Our primary observation is that the safety and performance specification can be satisfied even when some of the control jobs miss their deadlines.* This is because the feedback control mechanisms are often robust to delays in sensing and actuation. We leverage this observation for synthesizing the task scheduler that (a) incorporates that control tasks can miss some of the deadlines — specified as weakly-hard constraints and (b) the control performance when the control tasks are scheduled with these weakly-hard constraints meets the specified requirements. The proposed approach thus has two steps. The first step involves checking the compatibility between the weakly-hard constraints over scheduling a given feedback control task and its performance specification. The second step collects the weakly-hard constraints for all the control tasks and synthesizes a scheduler that is compatible with all the weakly hard constraints. This workflow is illustrated in Figure 1 where for each control system, we synthesize a collection of weakly hard constraints, and then a scheduler for the task set is synthesized from the constraints.

One of the important contributions of this paper is the usage of statistical model checking. In the literature, various deterministic techniques for ensuring that a controller implementation is compatible with the scheduling of the underlying control task [8,7,4]. These techniques either employ bounded model checking techniques, abstract interpretation, or software model checking techniques. However, due to the non-deterministic nature of the weakly-hard constraints, deterministic techniques are often overly conservative and suffer from scalability issues. As a result, the landscape of compatibility between the feedback control design and the weakly-hard scheduling constraints is poorly explored. To overcome this challenge, we use statistical model checking approaches for exploring the compatibility between a given weakly-hard constraint and the safety specification of the controller. That is, instead of ensuring that a weakly-hard constraint satisfies the performance specification *always*, we check if a weakly-hard constraint satisfies the specification *with high probability*. This begs the question: how can one provide deterministic performance guarantees when the compatibility is checked using statistical guarantees? We provide deterministic guarantees by checking if the final schedule (that is deterministic) satisfies all the performance requirements of the individual controllers — a sanity check. If the final schedule does not satisfy all the performance requirements, the schedule synthesizer would search the space of other compatible weakly-hard constraints and continue the search process.

Using a statistical approach for listing all the compatible weakly-hard constraints for a given controller and synthesizing the scheduler using a sanity check has several advantages. First, statistical model checking approaches are highly scalable as they only require *opaque-box* access to the plant and its corresponding feedback controller. Second, statistical model checking techniques are less conservative as the guarantees are based on the behaviors of the controller that are sampled during the verification process. Finally, the scheduler synthesized using statistical model checking is provably at least as good as the schedule synthesized using traditional model checking approaches. We demonstrate the advantages of our method to synthesize a schedule for various control tasks sharing the same computational platform. *To the best of the authors' knowledge, this is the first approach that leverages a statistical model checking approach for synthesizing a deterministic scheduler.*

1.1 Related Work

A number of works have studied the characterization of deadline misses in real-time settings. Notably, the work in [1] proposes a systematic method for characterizing deadline hit/miss patterns. These so-called weakly-hard constraints have been studied in a number of settings, including schedulability analysis, formal verification, and runtime monitoring, with [3,8,13] being some recent examples. A significant body of research exists regarding the verification of control safety properties, such as stability, under specific constraints such as these, with notable examples found in [9,13,14].

In particular, this work has been inspired by a number of recent works [6,7] that relate *quantitative* safety properties—such as the maximum deviation of a system's trajectory from an ideal trajectory—with the maximum number of consecutive deadline misses. The latter uses a deterministic reachable set-based method, while the former relies on Statistical Hypothesis Testing (SHT) to provide a deviation upper bound with a statistical guarantee. However, there has been considerably less focus on the synthesis of task schedules that satisfy control safety properties, particularly those beyond stability, which is the central topic of this paper. The study in [19] delved into scheduling considering safety constraints, but their deterministic deviation estimation method suffers from a tradeoff between exponentially growing execution time and large overestimation of the deviation, similar to [7].

At a philosophical level, the current work is similar to program synthesis using stochastic search [16]and neuro-symbolic synthesis [12]. In these program synthesis approaches, the search for the *correct program* is conducted using a stochastic process such as random search or generative neural network and the final program returned is verified to satisfy the specification.

2 Background

This section introduces the system formulation, its behavior under deadline misses, and the characterization of deadline miss patterns that affect system behavior. These concepts will be used throughout the rest of the paper.

2.1 System formulation

Control systems are dynamic in nature and are often described using differential equations. These are called *state equations*, and each one describes the relationship between the time derivative of a single state variable with respect to other state variables and the system's inputs. For instance, an autonomous vehicle system might be described with equations regarding its velocity, acceleration, and steering angle. A state equation for is of the form

$$\dot{x}(t) = f(x, u, t), \tag{1}$$

where $x(t) \in \mathbb{R}^n$ represents the states of the system and $u(t) \in \mathbb{R}^p$ the inputs to the system.

Certain characteristics of a control system can influence the representational forms of its dynamics. If the differential equations describing the system are time-invariant and linear, then the system dynamics can be expressed by the state-space model

$$\dot{x}(t) = Ax(t) + Bu(t), \tag{2}$$

where $A \in \mathbb{R}^{n\times n}$, and $B \in \mathbb{R}^{n\times p}$ represent the constant *continuous-time* transition matrix and the input matrix, respectively. Equation (2) shows that the rate of change of the system state $\dot{x}(t)$ depends both on the current state $x(t)$ and the control input $u(t)$. When the controller is implemented as a software task, the state-space model needs to be discretized and assumes the form of

$$x[t+1] = A_d x[t] + B_d u[t]. \tag{3}$$

where A_d and B_d represent the *discrete* counterparts of $\boldsymbol{A}$ and $\boldsymbol{B}$ respectively, and are computed as:

$$A_d = e^{AP}, \qquad B_d = \int_0^P e^{A\tau} B \, d\tau \tag{4}$$

Here, P is the sampling period for sensing the environment and actuation. We focus on closed-loop systems, where the state measurement is used to determine the control input of the next actuation. In practice, this is done by a periodic real-time task running on a processor and is assumed to be of the form

$$u[t] = Kx[t-1], \tag{5}$$

where $K \in \mathbb{R}^{p\times n}$ is the *feedback gain.* We follow the logical execution time (LET) paradigm, where the deadline equals the sampling period. A new control input is always applied at the deadline of the control job, *i.e.,* the system state is sampled at time $t-1$ and used to compute the control input for time t, where the state and control input is computed according to Equations (3) and (5).

2.2 System Behavior under Deadline Misses

The correct behavior of the control system described in Section 2.1 relies on the timely computation of the control input u by the end of each period. If the periodic real-time task computing the control input misses its deadline, then Equation (5) no longer holds, and the system may deviate from its ideal behavior and become unsafe. However, the safety of the system depends on the amount of deviation from its ideal behavior, and most systems can tolerate a certain degree of deviation before it becomes unsafe. In this section, we quantify the deviation from a system's ideal behavior and define the set of safe trajectories.

We consider the behavior of the system only over a finite time horizon H. Thus, the states of the system will be recorded at time points $0, 1, \ldots, H$. For ease of exposition, we also assume that the initial state of the system is $z[0] \in \mathbb{R}^n$. Setting the initial state as $z[0]$, we define the *nominal trajectory*, denoted as τ_{nom}, of the system as the trajectory resulting from no deadline misses as follows.

Definition 1 (Nominal Trajectory). *A nominal trajectory (τ_{nom}) of a system is the sequence of states of length $H+1$ of the form $x[0], x[1] \ldots, x[H]$, where $x[0] = z[0]$ is the initial state, $x[t+1]$ is computed with Equation* (3) *and $u[t]$ is computed with Equation* (5)*.*

Let $\mathcal{T}$ be the set of sequences of length $H+1$ over $\mathbb{R}^n$ where the control task has missed some deadlines. For each $\tau \in \mathcal{T}$, $\tau = \tau[0], \tau[1] \ldots, \tau[H]$ where $\tau[i] \in \mathbb{R}^n$ and $\tau[0] = z[0]$. Intuitively, $\mathcal{T}$ denotes the set of *all* possible trajectories of length $H+1$ in the state space starting from the initial state $z[0]$. We wish to find a subset of $\mathcal{T}$ that does not deviate from the nominal trajectory τ_{nom} by more than a safety bound d^{safe}. This requires a way to quantify deviations from the nominal trajectory. With a metric $\mathrm{dis}(\cdot)$ defined between two points in $\mathbb{R}^n$, we define the distance between a pair of trajectories (τ, τ'), also denoted as $\mathrm{dis}(\cdot)$, as follows:

$$\mathrm{dis}(\tau, \tau') = \max_{0 \le t \le H} \mathrm{dis}(\tau[t], \tau'[t]). \tag{6}$$

We now fix a safety margin $d^{safe} > 0$. This leads to the set of safe trajectories $\mathcal{T}_{safe} \subset \mathcal{T}$, defined as

$$\mathcal{T}_{safe} = \{\tau \mid \mathrm{dis}(\tau, \tau_{nom}) \le d^{safe}\}. \tag{7}$$

This is the set of trajectories that do not exceed the safety margin around the nominal trajectory, *i.e.*, trajectories that do not deviate more than d^{safe} from the nominal trajectory. Clearly, the nominal trajectory is also a member of $\mathcal{T}_{safe}$.

2.3 Characterizing Deadline Miss Patterns

We now characterize the pattern of deadline misses and its connection to deviation in system behavior. The *weakly hard constraints*, proposed in [1], provide an alternative to the traditional hard/soft classification of real-time systems and have been studied in a number of settings including schedulability analysis and formal verification [8,3,13]. The $\binom{m}{k}$ constraint is one of the four types of constraints proposed in [1] and states that out of any k consecutive deadlines of the task, *at least* m of them must be met. A notable result from [1] is that weakly hard constraints are regular languages over $\{0, 1\}$, provided that a deadline hit is represented by 1 and miss by 0. We denote the regular language representing $\binom{m}{k}$ as $\mathcal{L}_{(m,k)}$.

Finally, suppose $\gamma \in \{0, 1\}^H$ is a sequence of length H representing a pattern of deadline hits and misses. A unique trajectory τ_γ is defined for γ where $\tau_\gamma[0] = z[0]$, $\tau_\gamma[t+1]$ is computed with Equation (3), and

$$u[t] = \begin{cases} K\tau_\gamma[t-1], & \gamma[t] = 1 \\ u[t-1], & \gamma[t] = 0. \end{cases} \tag{8}$$

This leads to

$$\mathcal{T}_{(m,k)} = \{\tau_\gamma \mid \gamma \in \mathcal{L}_{(m,k)}\}. \tag{9}$$

In other words, $\mathcal{T}_{(m,k)}$ is the set of all trajectories resulting from deadline hit/miss patterns in the regular language $\mathcal{L}_{(m,k)}$. We call the system *safe* under $\binom{m}{k}$ if and only if $\mathcal{T}_{(m,k)} \subseteq \mathcal{T}_{safe}$, which is equivalent to checking the following inequality

$$\max_{\tau \in \mathcal{T}_{(m,k)}} \operatorname{dis}(\tau, \tau_{nom}) \leq d^{safe}. \tag{10}$$

Intuitively, it means the system is safe under $\binom{m}{k}$ if and only if the maximum deviation of all trajectories $\tau \in \mathcal{T}_{(m,k)}$ is less than or equal to the safety margin of the system.

3 Statistical Hypothesis Testing

Computing the exact maximum deviation pertaining to a given constraint (*e.g.*, weakly hard constraints), in the worst case, might require computing deviation of 2^H trajectories, where H is the time horizon. This is clearly infeasible for practical values of H. To address this issue, [7] proposed a deterministic technique that employs reachable sets to compute an upper bound on the maximum deviation for a given constraint, rather than the exact maximum deviation. This approach enables safe deviation bounds (*i.e.*, an upper bound) to be computed for large time bounds. However, this technique has two main issues. Firstly, because it relies on reachable-set-based methods, the resulting upper bound is often overly conservative, rendering it ineffective for the safety verification in some instances. Secondly, because computing reachable sets can be computationally intensive, it was unable to compute a bound in a reasonable amount of time (1 hour) for some applications, limiting its applicability.

In contrast to deterministic methods, a novel method presented in [6] uses statistical hypothesis testing (SHT) to compute a bound on the maximum deviation with a high level of confidence that is determined by the user. This technique demonstrates better performance than the deterministic method proposed in [7], both in terms of the tightness of the computed bound and computation time. In the rest of this section, we briefly review the technique proposed in [6] and present an illustrative example of the technique.

The proposed method in [6] employs a statistical hypothesis testing framework—specifically, Jeffreys's Bayes factor-based method—alongside a counterexample-based refinement strategy to compute a deviation upper bound with probabilistic guarantees. It comprises three main modules, namely the `Hypothesizer` module, the `Verifier` module, and the `Refiner` module. First, the inputs—the system model, the initial state, and the *nominal trajectory* of the system—are provided to the `Hypothesizer` module, which then makes an initial *guess* of the upper bound on the maximum deviation. Subsequently, the guessed upper bound is forwarded to the `Verifier` module, which uses SHT to verify its correctness. If the guessed deviation bound is incorrect, the `Verifier` module generates a counterexample and passes it on to the `Refiner` module. The `Refiner` module refines the guessed deviation based on the counterexample and sends it back to the `Verifier` module for re-verification. This iterative process continues until a

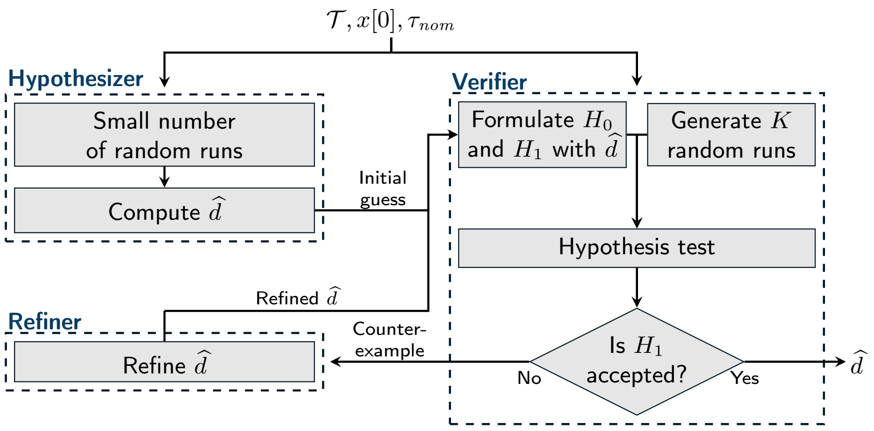

Fig. 2. Proposed statistical hypothesis testing approach [6]. The `Hypothesizer` module makes an initial guess for the upper bound on the maximum deviation and sends it to the `Verifier` module for verification. If the verification fails, the counterexample is generated and sent to the `Refiner` module, which refines the guessed deviation based on the counterexample and sends it back to the `Verifier` module. This iteration continues until a successful verification is achieved, at which point the computed bound $\widehat{d}$ on the maximum deviation is returned.

successful verification occurs, with the desired level of confidence specified by the user. At this point, the computed bound on the maximum deviation is returned. The method is illustrated in Figure 2.

We now give a more detailed review of the statistical hypothesis testing procedure employed within the `Verifier` module. The purpose of the `Verifier` module is to verify, using SHT, whether a given bound on the maximum deviation $\widehat{d}$ is indeed correct with the specified confidence level. For a given value of confidence c, the null and alternate hypotheses can be formulated as follows:

$$H_0 : \mathcal{P}rob\left[\mathcal{T}, x[0], \tau_{nom}, \widehat{d}\right] < c \tag{11}$$

$$H_1 : \mathcal{P}rob\left[\mathcal{T}, x[0], \tau_{nom}, \widehat{d}\right] \geq c \tag{12}$$

where $\mathcal{P}rob\left[\mathcal{T}, x[0], \tau_{nom}, \widehat{d}\right]$ is the probability that a randomly selected trajectory τ has a deviation that remains within the deviation upper bound $\widehat{d}$. Intuitively, the null hypothesis H_0 represents the rejection of $\widehat{d}$ as the correct deviation upper bound, while the alternative hypothesis H_1 represents the acceptance of $\widehat{d}$. The `Verifier` tests the two hypotheses by first generating a set of K samples from the assumed distribution of executions, denoted by $X = \{\tau_1, \tau_2, \ldots, \tau_K\}$. The sample size K is derived from the Bayes factor B and confidence level c selected by the user. It then examines whether *all* members of X satisfy the upper bound constraint of $\widehat{d}$ (*i.e.*, the deviation of all the trajectories in X, from the nominal trajectory τ_{nom}, is less than $\widehat{d}$). If all members of X

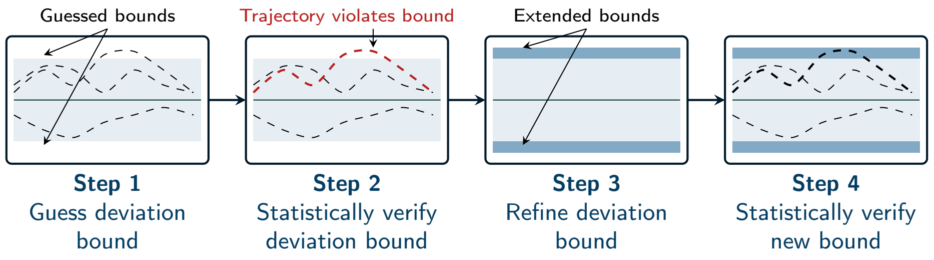

Fig. 3. Steps to compute deviation using the SHT method [6]. The nominal trajectory is the solid line in the center, while randomly generated trajectories are shown as dashed lines. The light blue envelope represents the calculated upper bound on deviation.

satisfy this condition, we accept the alternative hypothesis H_1 and report $\widehat{d}$ as the estimated bound. However, if at least one counterexample exists, then the `Verifier` rejects *devub* and send the counterexample to the `Refiner` module.

3.1 Example of Statistical Hypothesis Testing

In this section, we demonstrate how the SHT framework from [6] is used to compute the upper bound on the deviation $\widehat{d}$ for a specific weakly hard constraint. Figure 3 illustrates the overall process. We use the linearized motion of an F1Tenth [11] model car as an example, discretized with the period $P = 20\,\text{ms}$:

$$x[t+1] = \begin{bmatrix} 1.000 & 0.1300 \\ 0 & 1.0000 \end{bmatrix} x[t] + \begin{bmatrix} 0.0256 \\ 0.3937 \end{bmatrix} u[t]$$

In this example, we compute the deviation upper bound for the weakly hard constraint $\binom{1}{3}$ from a nominal trajectory with no deadline misses, starting with the initial state $x[0] = [1\ 1]^T$. We assume a time horizon of $H = 5$ periods. When a deadline is missed, the overrun job is killed and the control input from the previous period is held, consistent with the *Hold&Kill* policy [9]. The SHT framework compute the deviation upper bound $\widehat{d}$ with the following steps:

1. The `Hypothesizer` *guesses* an upper bound by considering a small sample of random sequences of deadline hit/miss that satisfy the weakly hard constraint $\binom{1}{3}$: 10110, 11001 (0 indicates miss, 1 indicates hit). The maximum deviation from the two random samples, 0.0482, is used as the initial guess $\widehat{d_0}$.
2. The guessed upper bound $\widehat{d_0} = 0.0482$ is verified by the `Verifier`, which returns *False*. *i.e.*, a counterexample is found whose deviation from nominal trajectory $d = 0.3157$ exceeds the initial guess $\widehat{d_0} = 0.0482$.
3. Since the guessed bound $\widehat{d_0}$ was not verified, the `Refiner` takes the counterexample produced by the `Verifier` and updates the previous deviation upper bound, $\widehat{d_0} = 0.0482$, to the deviation obtained from the counterexample, $\widehat{d_1} = 0.3157$.

4. The refined upper bound $\widehat{d_1} = 0.3157$ is again sent to the `Verifier` module for re-checking. This time, the `Verifier` module accepts the $\widehat{d_1} = 0.3157$ as a valid upper bound up to the desired probabilistic guarantees, and terminates the procedure

We note that the $\widehat{d} = 0.3157$ computed from the SHT is the same as the deviation produced by the reachability analysis method used in [19] for this example. This is because the small number of total deadline hit/miss sequences (as a result of the small time horizon $H = 5$) enables both methods to find the exact trajectory corresponding to the deviation upper bound. However, this is no longer the case as H increases due to the exponentially growing number of total deadline hit/miss sequences.

4 Proposed Schedule Synthesis

The schedule synthesis problem we wish to solve is as follows:

Problem 1 (Schedule Synthesis). Given a set of N controller tasks with the same period, their respective safety margins d^{safe}, and an implementation platform where at most $J < N$ controllers can be scheduled in each period, determine if a schedule over the time horizon H exists where all the controller tasks can be scheduled without deviating more than their safety margin. Furthermore, synthesize a schedule if one exists.

We propose an efficient solution to Problem 1, using the deviation upper bound estimation methods proposed in [6]. Our approach involves three stages:

1. *Compute* the collection of all weakly hard constraints that are statistically safe.
2. *Synthesize* a candidate schedule using the list of safe constraints from each task.
3. *Verify* the safety of the candidate schedule. If it is unsafe, go back to step 2; if it is safe, exit with the safe schedule.

Our approach is similar to the two-stage schedule synthesis scheme proposed by [19]. However, our method is different from theirs in two important aspects: First, the method in [19] deploys a deterministic technique to determine a list of safe constraints. This causes issues where lengthy execution time and substantial overestimation of deviation upper bounds significantly restrict the pool of weakly hard constraints available for schedule synthesis considerably. In contrast, our scheduler synthesis method capitalizes on the speed and tightness advantages of the SHT-based method proposed in [6]. Second, given that the deviation guarantee provided by the SHT-based method is probabilistic, our method incorporates an additional schedule verification phase. This step verifies the safety of the schedule produced by the schedule synthesis step. It is important to note that, because the exact deviations for the trajectories corresponding to the final schedule can be exactly determined, the schedules generated by our method are *deterministically* safe despite its reliance on a statistical method for constraint checking. This is different from the [6], where the gain in performance and tightness is at the price of only obtaining a probabilistic guarantee (*e.g.*, with confidence $c = 0.99$).

Constraint Checking Given a control task T_i and its safety margin d_i^{safe}, constraint checking determines the set of weakly-hard constraints under which the system is safe. This amounts to checking if $\mathbf{d}(m,k) \le d_i^{safe}$, where $\mathbf{d}(m,k)$ is the maximum deviation of the trajectories in $\mathcal{T}_{(m,k)}$ from the nominal trajectory of T_i. More precisely,

$$\mathbf{d}(m,k) = \max\big\{\mathrm{dis}(\tau, \tau_{nom}) \bigm| \tau \in \mathcal{T}_{(m,k)}\big\}. \tag{13}$$

However, checking this directly is expensive, due to the exponential number of hit/miss patterns of length H. To get around this, we compute an upper bound $\widehat{d}(m,k)$ on $\mathbf{d}(m,k)$ with confidence c using the SHT-based method in [6]. It then suffices to check that $\widehat{d}(m,k) \le d^{safe}$ to derive a probabilistic guarantee of the safety of the system under $\binom{m}{k}$. We then iterate through all weakly-hard constraints (up to a maximum window size $k_{max} \ll H$) and compute $\widehat{d}(m,k)$ using the SHT-based method for each constraint $\binom{m}{k}$. If $\widehat{d}(m,k) \le d_i^{safe}$, we conclude that the system is safe under $\binom{m}{k}$ with confidence c and add $\binom{m}{k}$ to the set of safe constraints.

Schedule Synthesis As introduced in Section 2.3, a weakly-hard constraint $\binom{m}{k}$ is a regular language $\mathcal{L}(m,k)$ over $\{0,1\}$, where a string represents a hit/miss pattern satisfying $\binom{m}{k}$. We denote such an automaton for the control task T_i by $\mathbb{A}_i = \langle L^i, \Sigma, \delta^i, F^i, \ell_0^i\rangle$, where L^i is a set of states, $\Sigma = \{0,1\}$ is the input alphabet (miss/hit), $\delta^i = L^i \times \Sigma \to L^i$ is the transition function, F^i is the set of accepting states, and ℓ_0^i is the initial state. With this construction, an accepting run of $\mathbb{A}_i$ is a hit/miss pattern that satisfies at least one safe weakly-hard constraint for the corresponding controller task T_i. We use these individual finite automata for each task i to construct a *scheduler automaton* as follows:

Definition 2 (Scheduler Automaton). *A* ***scheduler automaton*** $\mathbb{A}^S$ *for a set of N controllers whose constraints are represented by the automata of the form $\mathbb{A}_i = \langle L^i, \Sigma, \delta^i, F^i, \ell_0^i\rangle$, where at most J controllers can be scheduled in each time slot, is defined as an automaton $\langle L^S, \Sigma^S, \delta^S, F^S, \ell_0^S\rangle$:*

- L^S *set of states, $L^S = \prod_i L^i$;*
- 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 $\sum_i \sigma^i \le J$;*
- δ^S *transition function, $\delta^S(\ell, \sigma) = \prod_i \delta^i(\ell^i, \sigma^i)$;*
- F^S *accepting states of the automaton, $F^S = \prod_i F^i$;*
- ℓ_0^S *initial state of the automaton, $\ell_0^S = \prod_i \ell_0^i$.*

The set of states L^S is a Cartesian product of the controller automaton states: $L^S = L^1 \times L^2 \times \cdots \times L^N$, where each state $\ell \in L^S$ is a tuple of individual states 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, and an action $\sigma = \langle \sigma^1, \sigma^2, \ldots, \sigma^N \rangle$ is valid if and only if $\sum_i \sigma^i \leq J$. For example, $N = 3, J = 1$ results in $\Sigma^S = \{000, 001, 010, 100\}$, where $\sigma = 010$ indicates that only the second controller is scheduled. The transition function δ^S is the Cartesian product of individual transition functions $\prod_i \delta^i$. Concretely, assuming $\sigma \in \Sigma^S$ is a valid action for the scheduler automaton $\mathbb{A}^S$, the transition function δ^S becomes:

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

The set of accepting states F is the Cartesian product of the individual accepting states. A state $\ell = \langle \ell^1, \ell^2, \ldots, \ell^N \rangle$ is an accepting state if and only if $\ell^i \in F^i$ 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) state, the scheduler automaton will also transition to a non-accepting state.

Under this formulation, an accepting run of length $H + 1$ of the scheduler automaton is a schedule that satisfies at least one weakly hard constraint in the set of safe constraints for each control task. The existence of safe schedules can be checked by running emptiness checking on the scheduler automaton, and schedules can be generated using breadth-first search (BFS).

Schedule Verification Given a synthesized schedule, we verify if the actual deviation of each system is within its safety margin. Toward this, we first calculate the exact trajectory of each system τ^i using the synthesized schedule for that system γ^i from the previous stage. The actual deviation of that system can be then calculated using the with $d^i = \text{dis}(\tau^i, \tau^i_{nom})$. If $d^i < d_i^{safe}$ for all systems, the schedule is verified to be safe. Note that since the exact deviation for each system is determined by the schedule, the schedule is *deterministically* safe despite the probabilistic guarantee given by the SHT-based method in Constraint Synthesis. Otherwise, the verification fails and a different schedule must be synthesized. If no schedule has passed the verification step and no more candidate schedules are available from schedule synthesis, the process terminates and returns `No Schedule`.

4.1 Comparison with Deterministic Method Proposed in [19]

We now demonstrate that our scheduler synthesis technique that uses stochastic hypothesis testing for generating all the weakly-hard constraints is at least as good as scheduler synthesis that uses deterministic verification process.

Lemma 1. *Consider the controller T_i that is scheduled satisfying the weakly-hard constraint $\binom{m}{k}$ leading to at most $d(m,k)$ deviation from the nominal trajectory. For such controller, if the SHT estimates the deviation $\hat{d}(m,k)$ and* *a deterministic verification method provides an upper bound of $\tilde{d}(m,k)$, then, $\hat{d}(m,k) \leq d(m,k) \leq \tilde{d}(m,k)$.*

Since the deterministic verification techniques would compute a conservative overapproximation of all behaviors, $d(m,k) \leq \tilde{d}(m,k)$. Since the estimation of the deviation during SHT is generated from one of the counterexamples, there exists at least one behavior with deviation $\hat{d}(m,k) \leq d(m,k)$.

Theorem 1. *If the scheduler synthesis procedure using the deterministic verification technique for collecting the set of all the safe weakly-hard constraints successfully generates a schedule, then the SHT-based constraint generation would also eventually generate it.*

Proof. Suppose that the deterministic verification method returned a safe schedule where the weakly-hard constraint for each task T_i is $\binom{m}{k}_i$. From Lemma 1, we know that the same weakly-hard constraint $\binom{m}{k}_i$ would also be considered safe using the SHT method. Therefore, the scheduler automaton generated using constraints using SHT would either terminate early with a safe schedule or eventually construct the scheduler automaton with weakly-hard constraints $\binom{m}{k}_i$ for the task i. Therefore, the scheduler automaton, in the worst case, returns the same schedule obtained using deterministic verification.

5 Evaluation

To evaluate the effectiveness of our approach in scheduler synthesis, we implemented the proposed method in *Julia* [2] and conducted experiments on a system where five different controllers share the same computational platform. Our goal is to answer the following research questions:

1. Is our approach capable of synthesizing safe schedules where existing methods cannot?
2. How does the execution time of our approach compare to existing methods?

5.1 Benchmarks

We use five dynamical system models from the automotive domain. All systems are discretized with a period $P = 20\,\text{ms}$, and controllers for each system are computed with LQR using a one-period delay.

RC Network (RC) Our first model is a resistor-capacitor network [5] with the following model:

$$\dot{x}(t) = \begin{bmatrix} -6.0 & 1.0 \\ 0.2 & -0.7 \end{bmatrix} x(t) + \begin{bmatrix} 5.0 \\ 0.5 \end{bmatrix} u(t).$$

Table 1. Initial states and safety margins of the five models used in experiments.

	Initial State	Safety Margin
RC Network	$[1\ 1]^T$	0.07
F1 Tenth Car	$[1\ 1]^T$	0.56
DC Motor	$[100\ 100]^T$	0.1
Car Suspension	$[100\ 100\ 100\ 100]^T$	0.8
Cruise Control	$[1\ 1\ 1]^T$	0.06

F1Tenth Car (F1) Our second model is the linearized motion of an F1Tenth model car [11]:

$$\dot{x}(t) = \begin{bmatrix} 0 & 6.5 \\ 0 & 0 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 19.685 \end{bmatrix} u(t).$$

Our next three plant models are selected from [15] and also represent subsystems from the automotive domain.

DC Motor (DC) Our third model is the speed controller for a DC motor adapted from [18]:

$$\dot{x}(t) = \begin{bmatrix} -10 & 1 \\ -0.02 & -2 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 2 \end{bmatrix} u(t).$$

Car Suspension (CS) Our fourth model is a suspension system adapted from [17]:

$$\dot{x}(t) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -8 & -4 & 8 & 4 \\ 0 & 0 & 0 & 1 \\ 80 & 40 & -160 & -60 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 80 \\ 20 \\ -1120 \end{bmatrix} u(t).$$

Cruise Control (CC) Our final model is a cruise control system adapted from [10]:

$$\dot{x}(t) = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -6.0476 & -5.2856 & -0.238 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 0 \\ 2.4767 \end{bmatrix} u(t).$$

5.2 Experiments

We compared the effectiveness and performance of the proposed SHT-based method with the existing deterministic method [19]. For each method, the five models in Section 5.1 are used with a starting state x_0 that is offset from the origin in the state-space. The goal of all controllers is to bring the state x to the origin. The starting states and safety margins for the five systems are shown

Table 2. Deviation upper bounds $\widehat{d}$ for each system and weakly hard constraint, computed by the SHT-based method.

	Window	Minimum Hits (m)					
	Size (k)	1	2	3	4	5	6
	1	0.0	–	–	–	–	–
	2	0.036	0.0	–	–	–	–
RC Network	3	0.0656	0.036	0.0	–	–	–
$d^{safe} = 0.07$	4	0.0899	0.0656	0.036	0.0	–	–
	5	0.11	0.0899	0.0656	0.036	0.0	–
	6	0.126	0.11	0.0899	0.0656	0.036	0.0
	1	0.0	–	–	–	–	–
	2	0.179	0.0	–	–	–	–
F1 Tenth Car	3	0.364	0.179	0.0	–	–	–
$d^{safe} = 0.56$	4	0.557	0.364	0.179	0.0	–	–
	5	0.75	0.557	0.364	0.179	0.0	–
	6	0.949	0.75	0.557	0.364	0.179	0.0
	1	0.0	–	–	–	–	–
	2	0.0546	0.0	–	–	–	–
DC Motor	3	0.107	0.0546	0.0	–	–	–
$d^{safe} = 0.1$	4	0.156	0.107	0.0546	0.0	–	–
	5	0.204	0.157	0.107	0.0546	0.0	–
	6	0.248	0.204	0.157	0.107	0.0546	0.0
	1	0.0	–	–	–	–	–
	2	0.16	0.0	–	–	–	–
Car Suspension	3	0.34	0.16	0.0	–	–	–
$d^{safe} = 0.8$	4	0.53	0.34	0.159	0.0	–	–
	5	0.729	0.529	0.339	0.159	0.0	–
	6	0.908	0.717	0.526	0.338	0.158	0.0
	1	0.0	–	–	–	–	–
	2	0.0138	0.0	–	–	–	–
Cruise Control	3	0.0298	0.0115	0.0	–	–	–
$d^{safe} = 0.06$	4	0.0368	0.0212	0.0117	0.0	–	–
	5	0.0455	0.0323	0.0194	0.0115	0.0	–
	6	0.0584	0.0423	0.0262	0.0201	0.0116	0.0

in Table 1. We assume that the controller tasks of the five systems are implemented on the same processor. During each period of $P = 20\,\text{ms}$, we also assume that only two out of the five tasks can be run (*i.e.*, $J = 2$). The limitation that at most two tasks can be executed in each slot is very similar to the scheduling in AUTOSTAR Adaptive, where several tasks with the same period are combined together for scheduling purposes.

We used Julia 1.8 for all experiments. The parameters used in the experiments are as follows: The maximum window size for weakly hard constraint k_{max} is 6; the Bayes factor B is $4.15 * 10^5$; the time horizon H is 100; the confidence level c for statistical hypothesis testing is 0.99. We used Euclidean distance for all deviation estimation.

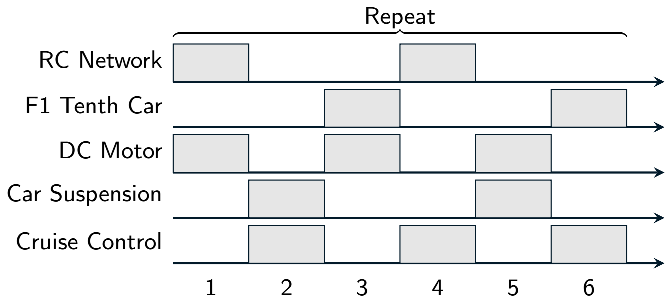

Fig. 4. Schedule synthesized by the SHT-based method.

5.3 RQ1: Effectiveness of the Proposed Approach to Synthesize Safe Schedules

To answer RQ1, we applied our SHT-based constraint and schedule synthesis method to the systems in Section 5.1. We also ran the deterministic method in [19] with iteration parameter $n = 15$. **Result**: the proposed SHT-based method was able to synthesize a safe schedule, shown in Figure 4, while the deterministic method failed to find a safe schedule.

Detailed results of constraint and schedule synthesis are outlined in Table 2 and Table 3, respectively. In Table 2, the value at row k and column m in Table 2 represents the deviation upper bound $\widehat{d}(m,k)$ associated with weakly hard constraint $\binom{m}{k}$, computed by the SHT-based method. If $\widehat{d}(m,k) < d^{safe}$, then $\binom{m}{k}$ is added to the safe list of constraints for that system. The safe values are highlighted in Table 2, where values highlighted with 🟩 are safe according to both the SHT-based method and the deterministic method ($\widehat{d}, \tilde{d} < d^{safe}$), and values highlighted with light green 🟩 are safe according to the SHT-based only and unsafe according to the deterministic method ($\widehat{d} \leq d^{safe} \leq \tilde{d}$). As shown in Lemma 1, any constraint deemed safe by the deterministic method is always deemed safe by the SHT-based method too. We observe from Table 2 that while the deterministic method and the SHT-based method performed similarly for the RC Network and DC Motor systems, the SHT-based method is able to produce tighter estimates for the F1 Tenth Car, Car Suspension and Cruise Control systems. The Car Suspension and Cruise Control systems especially see a large increase in safe constraints.

Table 3 shows the exact deviation for each system under the schedule in Figure 4, per the schedule verification procedure described in Section 4. We observe that the schedule for each individual system closely matches one of the safe constraints for that system. For example, the schedule for RC Network matches the weakly hard constraint $\binom{1}{3}$. We compared the actual deviation with the estimated $\widehat{d}(m,k)$ for that constraint. For two out of five systems, the deterministic method estimates deviation upper bounds $\tilde{d}$ higher than the safety margin d^{safe}; the SHT-based method estimates values closer to the actual deviation that

Table 3. Exact deviation values for each system when scheduled according to Figure 4. All systems have deviation values within their respective safety margin, *i.e.*, the schedule is deterministically safe.

	Safety Margin	Closest Constraint	Estimated $\widehat{d}(m,k)$	Estimated $\tilde{d}(m,k)$	Actual Deviation
RC Network	0.07	$\binom{1}{3}$	0.0656	0.0656	0.0092
F1 Tenth Car	0.56	$\binom{1}{3}$	0.364	0.364	0.303
DC Motor	0.1	$\binom{1}{2}$	0.0546	0.0546	0.157
Car Suspension	0.8	$\binom{1}{3}$	0.34	1.04	0.13
Cruise Control	0.06	$\binom{1}{2}$	0.0138	0.0712	0.00578

are all within the safety margin. This matches the result in Table 2, where the deterministic method greatly overestimates the deviation upper bound for the Car Suspension and Cruise Control systems. Finally, we note that the scheduler passed the schedule verifier on the first try. This is expected, given the high confidence level $c = 0.99$ used during the constraint synthesis step.

These results suggest that our SHT-based approach effectively produces *deterministically* safe schedules, even when the existing method fails to do so.

5.4 Reduction of Execution Time using the Proposed Approach

In addition to the two configurations used in Section 5.3, we also ran an additional configuration of the deterministic approach in [19] with parameter $n = 18$. We ran this additional configuration because a higher n value correlates to a tighter deviation estimation, and we incremented the n value progressively until the deterministic method returns a valid schedule with $n = 18$. The execution times for the constraint synthesis and schedule synthesis using the two methods are outlined in Table 4.

We observe that the SHT-based method delivers a 30× to 600× speed up for constraint synthesis of individual systems compared to the deterministic method. In total, the SHT-based method is 55× and 394× faster than the deterministic method with $n = 15$ and $n = 18$, respectively. Second, the procedure of schedule synthesis and verification accounts for a relatively small fraction of the overall execution time. Third, our method takes roughly the same amount of time for each system, whereas the deterministic method takes much more time to compute the Car Suspension system than other systems. Finally, we note that although the deterministic method is able to synthesize a safe schedule for the benchmarks with parameter $n = 18$, it does so using exponentially more time as n increases [7]. The SHT-based method eliminates the need for finding the suitable n value, which in itself is a time-consuming process.

In summary, the deterministic method takes orders of magnitudes more time than the SHT-based method to execute and requires a suitable n value to synthesize a safe schedule. The SHT-based is able to synthesize a schedule much

Table 4. Execution times for the SHT-based and deterministic methods.

	SHT ($c = 0.99$)	DET ($n = 15$)	DET ($n = 18$)
RC Network	2.17 s	115.35 s	818.92 s
F1 Tenth Car	2.09 s	115.96 s	817.70 s
DC Motor	2.78 s	112.95 s	820.18 s
Car Suspension	3.10 s	292.35 s	2071.46 s
Cruise Control	3.33 s	111.65 s	799.89 s
Schedule Synthesis	0.017 s	0.106 s	0.012 s
Schedule Verification	0.008 s	–	–
Total	13.50 s	748.37 s	5328.16 s
Schedulable?	Yes	No	Yes

faster while eliminating the need to find the n value through trial and error, further reducing execution time.

6 Conclusions

Ensuring traditional real-time guarantees that no task misses its deadline has become increasingly challenging because of (a) the increased volume of software being deployed and (b) the increased complexity of hardware of autonomous systems. Consequently, some feedback control tasks may miss their deadlines, and consequently, their behavior would deviate from nominal behavior. This causes a divergence between the design and implementation of autonomous systems, posing a major hurdle in their certification. Our approach to overcome this hurdle is to synthesize a *correct by construction* control system implementation for all the control tasks sharing the computational resources. In this paper, we demonstrated that incorporating probabilistic model checking for collecting a collection of weakly-hard constraints in the scheduler synthesis (1) enables us to schedule tasks that couldn't be scheduled using deterministic verification techniques and (2) would decrease the computational effort in synthesizing such schedules. We have demonstrated these two advantages by scheduling a task set on a system where five controllers share the same computational resource.

Currently, our work requires that all the controllers have the same period and are scheduled on a uni-processor. While these restrictions are compatible with the AUTOSTAR Dynamic framework of scheduling groups of processes that share a control period, we plan to extend this work to controllers that have different sampling periods as a part of future work. We also plan to extend the probabilistic verification to account for hyperproperties such as stability and synthesize schedules such that the feedback loops are not only safe, but also stable.

References

1. Bernat, G., Burns, A., Liamosi, A.: Weakly hard real-time systems. IEEE Transactions on Computers (2001)
2. Bezanson, J., Edelman, A., Karpinski, S., Shah, V.B.: Julia: A Fresh Approach to Numerical Computing. SIAM Review **59**(1) (2017)
3. von der Brüggen, G., et al.: Efficiently approximating the probability of deadline misses in real-time systems. In: Euromicro Conference on Real-Time Systems. ECRTS (2018)
4. Duggirala, P.S., Viswanathan, M.: Analyzing real time linear control systems using software verification. In: 2015 IEEE Real-Time Systems Symposium, RTSS 2015, San Antonio, Texas, USA, December 1-4, 2015 (2015)
5. Gabel, R.A., Roberts, R.A.: Signals and Linear Systems. Wiley (1987), google-Books-ID: yOhSAAAAMAAJ
6. Ghosh, B., Hobbs, C., Xu, S., Duggirala, P.S., Anderson, J.H., Thiagarajan, P.S., Chakraborty, S.: Statistical Hypothesis Testing of Controller Implementations Under Timing Uncertainties. In: 2022 IEEE 28th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA) (2022)
7. Hobbs, C., Ghosh, B., Xu, S., Duggirala, P.S., Chakraborty, S.: Safety Analysis of Embedded Controllers Under Implementation Platform Timing Uncertainties. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **41**(11) (2022), conference Name: International Conference on Embedded Software EMSOFT
8. Huang, C., Li, W., Zhu, Q.: Formal verification of weakly-hard systems. In: Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control. HSCC '19 (2019)
9. Maggio, M., Hamann, A., Mayer-John, E., Ziegenbein, D.: Control-System Stability Under Consecutive Deadline Misses Constraints. In: 32nd Euromicro Conference on Real-Time Systems (ECRTS 2020) (2020)
10. Osman, K., Rahmat, M.F., Ahmad, M.A.: Modelling and controller design for a cruise control system. In: 2009 5th International Colloquium on Signal Processing & Its Applications (2009)
11. O'Kelly, M., Zheng, H., Karthik, D., Mangharam, R.: F1TENTH: An Open-source Evaluation Environment for Continuous Control and Reinforcement Learning. In: Proceedings of the NeurIPS 2019 Competition and Demonstration Track. PMLR (2020), https://proceedings.mlr.press/v123/o-kelly20a.html
12. Parisotto, E., Mohamed, A.r., Singh, R., Li, L., Zhou, D., Kohli, P.: Neuro-symbolic program synthesis. arXiv preprint arXiv:1611.01855 (2016)
13. Pazzaglia, P., et al.: Adaptive design of real-time control systems subject to sporadic overruns. In: Design Automation and Test in Europe. DATE (2021)
14. Pazzaglia, P., Mandrioli, C., Maggio, M., Cervin, A.: DMAC: Deadline-Miss-Aware Control. In: Euromicro Conference on Real-Time Systems. ECRTS (2019)
15. Roy, D., Zhang, L., Chang, W., Goswami, D., Chakraborty, S.: Multi-Objective Co-Optimization of FlexRay-Based Distributed Control Systems. In: 2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) (2016)
16. Schkufza, E., Sharma, R., Aiken, A.: Stochastic superoptimization. ACM SIGARCH Computer Architecture News **41**(1) (2013)
17. Schneider, R., Goswami, D., Zafar, S., Chakraborty, S., Lukasiewycz, M.: Constraint-driven synthesis and tool-support for FlexRay-based automotive control

systems. In: 2011 Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS) (2011)

18. Tilbury, D., Messner, B.: Control Tutorials for MATLAB and Simulink, https://ctms.engin.umich.edu/CTMS/index.php?aux=Home
19. Xu, S., Ghosh, B., Hobbs, C., Thiagarajan, P.S., Chakraborty, S.: Safety-Aware Flexible Schedule Synthesis for Cyber-Physical Systems Using Weakly-Hard Constraints. In: Proceedings of the 28th Asia and South Pacific Design Automation Conference ASP-DAC. ASPDAC '23 (2023)